MDP-based Network Friendly Recommendations

Theodoros Giannakas, Anastasios Giovanidis, Thrasyvoulos Spyropoulos

MDP-based Network Friendly Recommendations

THEODOROS GIANNAKAS, Eurecom, France
ANASTASIOS GIOVANIDIS, Sorbonne University, CNRS-LIP6, France
THRASYVOULOS SPYROPOULOS, Eurecom, France

Controlling the network cost by delivering popular content to users, as well as improving streaming quality and overall user experience, have been key goals for content providers in recent years. While proposals to improve performance, through caching or other mechanisms (DASH, multicasting, etc.) abound, recent works have proposed to turn the problem on its head and complement such efforts. Instead of trying to reduce the cost to deliver *every* possible content to a user, a potentially very expensive endeavour, one could leverage omni-present recommendations systems to nudge users towards content of low(er) network cost, regardless of where this cost is coming from. In this paper, we focus on this latter problem, namely optimal policies for "Network Friendly Recommendations" (NFR). A key contribution is the use of a Markov Decision Process (MDP) framework that offers significant advantages, compared to existing works, in terms of both modeling flexibility as well as computational efficiency. Specifically we show that this framework subsumes some state-of-the-art approaches, and can also optimally tackle additional, more sophisticated setups. We validate our findings with real traces that suggest up to almost 2X in cost performance, and 10X in computational speed-up compared to recent state-of-the-art works.

CCS Concepts: • **Networks** → **Network optimization**; **Network modeling**.

Additional Key Words and Phrases: Markov Decision Problem (MDP); recommendations

ACM Reference Format:
Theodoros Giannakas, Anastasios Giovanidis, and Thrasyvoulos Spyropoulos. 2022. MDP-based Network Friendly Recommendations. 1, 1 (January 2022), 30 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

1.1 Motivation

With multimedia traffic from Netflix, YouTube, Amazon, Spotify, etc. comprising the lion's share of Internet traffic [11], 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 actual monetary cost, e.g. for the CP to lease or invest in network and cloud resources, as well as network-related costs, e.g. congesting valuable resources, 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 [14], and more recently in wireless networks through femtocaching [36]. In addition to cost reduction for CPs

Authors' addresses: Theodoros Giannakas, Eurecom, Sophia-Antipolis, France, theodoros.giannakas@eurecom.fr; Anastasios Giovanidis, Sorbonne University, CNRS-LIP6, Paris, France, anastasios.giovanidis@lip6.fr; Thrasyvoulos Spyropoulos, Eurecom, Sophia-Antipolis, France, thrasyvoulos.spyropoulos@eurecom.fr.

Manuscript submitted to ACM

and NOs, caching also may lead to higher streaming rate, shorter latency, etc. [13], which results in an improved viewing/listening experience for the user. On the contrary, in platforms of video streaming services, low bitrate can result in large user abandonment rates [31]. Hence, *reducing the cost of bringing interesting content to users, whether through caching or any other means, will benefit everyone: the users, the content providers, and the network operators.*

At the same time, the role of finding such interesting content for the user is mainly the responsibility of Recommendation Systems (RS) which are an integral part of any popular content platform. For example, 80% of requests in Netflix, and more than 50% on YouTube, stem from the platform recommendations [20, 39]. RSs often make personalized recommendations to the user, suggesting items that best match her interests using techniques like collaborative filtering [34], deep neural networks [12], matrix factorization [24], etc. Nevertheless, the vast majority of popular RS focus on user satisfaction and similarity, but *do not account for the network cost of delivery*. Recently, proposals have emerged that suggest to turn the problem on its head: rather than only trying to reduce the (network) cost of bringing *every* content to the user (e.g., through caching), one could leverage the omni-present RSs or popular content platforms to also nudge the user towards such low(er) cost content [10, 23, 25]. Most of these works however, consider the problem of recommending content to the user in a myopic fashion, assuming that the user request pattern is iid. The key difference, and our main motivation, is that we focus on minimizing the network cost, assuming that the user session is of some random horizon L, and that our user has memory. Both ingredients lead to an intrinsically more difficult problem, where simply injecting cached content in the recommendation list (in order to increase the cache hit ratio) would no longer suffice, and thus, more elaborate approaches would be needed.

1.2 Related work

Ignoring network costs in content recommendation algorithms will inevitably lead to largely suboptimal performance for all parties involved. 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 [10, 17, 23, 28, 30, 35, 37]. The main objective of almost all these works can be summarized as how to recommend content that is still highly interesting to the user while at the same time its delivery cost is reduced. A simple (but not the only) example of this could be *to favor cached content* [26]. While various implementation barriers are sometimes cited [8], the increasing convergence [27] of CPs and NOs in the context of network slicing and virtualization suggests that in the very near future the 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 content X as opposed to content Y.

To date, a number of these early network-friendly RS proposals are (sometimes quite efficient) heuristics [10, 23]. A large number of these works focuses on *myopic* scenarios or algorithms, where the recommender aims to minimize the delivery cost just for the next content request [26]. In practice, however, when visiting popular applications like YouTube, Vimeo, Spotify, etc., a user [2, 3] *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 will recommend after the user watched some content in that 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 suboptimal. 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 [17, 18]. While the authors there manage to formulate the problem as a biconvex [17] and linear program [18], respectively, and provide optimal solutions, these works are characterized by 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) the user model is quite simplistic, assuming that the user click-through probability for recommended content is insensitive to the relevance of the content for the user.

1.3 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 user session, while offering content of relevance to the user. This allows the recommendation system to deal with user-sessions of arbitrary length (myopic, short-term, or long-term) and incorporate various assumptions on the user's reaction to the quality of recommendations. Our formulation can include several stricter problems that have been treated in the past literature and also solve new interesting cases.

(C.2) The MDPs are formed using the continuous item recommendation frequencies per viewed content as problem unknowns. This allows us to avoid searching for the optimal N-sized recommendation batch per viewed content. Instead, our formulation uses the least number of variables (K^2 specifically) and inequalities to describe the MDP without losing in optimality, compared to the fully detailed description. What is more, the policy iteration steps in the solution of the Bellman equations can be naturally decomposed into much smaller continuous problems, which (i) can be solved by low complexity linear, or convex programming techniques and (ii) can be solved in parallel (offering an additional potential speedup of $K\times$). Noteworthy is the fact that the complexity of the algorithmic solution becomes insensitive to the number N of recommendation slots.

(C.3) We introduce and solve two variations of the problem, where each one attempts to capture a different type of user response in relation to recommendations. The performance of the RS solution in cost and computational complexity for each model is evaluated in the simulation section, and compared with simpler heuristics.

(C.4) The benefit of the optimal MDP policy over simpler heuristic myopic policies is that it can suggest contents, which are neither low cost nor most relevant to the currently viewed object. This way, lower cost can be harvested in the future steps while guaranteeing a high quality of recommendations all the time.

The paper is structured as follows. Section 2 sets up the problem, presents the user-RS interaction and introduces the recommendation system input and multiple objectives. Feasible, optimal and sub-optimal recommendation policies are discussed. The problem is formed as an MDP in its general form in Section 3 and an algorithmic solution is described based on Bellman equations. In Section 4 we present two different variations of the general MDP; each one of which makes different assumptions over the user reaction to recommended content. Section 5 contains the evaluation of our policies in terms of cost and user satisfaction performance against heuristics and state of the art solutions. A short discussion for future directions is presented in Section 6, and our conclusions are drawn in Section 7.

2 PROBLEM SETUP

2.1 User session structure

We consider a user who starts a new session in a multimedia application, e.g. YouTube, and requests sequentially a random number of items from its library $\mathcal{K}$. Such applications are equipped with a Recommendation System (RS), responsible for helping the users discover new content. Our user starts her session by requesting content i with probability $p_0(i)$; this quantity expresses her personal preference for i, and is also the probability with which the user

requests i outside of the RS, e.g., though the application's search bar. We denote the pmf, over the set of items $\mathcal{K}$ as $\mathbf{p}_0$, populated with the $p_0(i)$'s. We further assume that $\mathbf{p}_0$ is known and fixed. [1].

The length of each session is random, and we assume that it follows a geometric distribution with mean $(1-\lambda)^{-1}$. It is further assumed here that the session length is *independent* of the RS suggestions. The user session has the following structure:

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

2.2 Recommender system inputs about user preferences

Two key questions arise now from the above discussion: (a) how does the RS decide on the batch of contents recommended to the user for item i (step 2 above), and (b) how is $\mathbf{p}_0$ attained (step 1 and 4 above). We begin the discussion with (a).

Entertainment oriented applications massively collect data related to user interaction and content ranking, allowing them to become more effective in their recommendations. The problem of recommendations, in its original form, can be cast as a matrix completion, where a matrix with columns as many as users and rows as many as items, is populated by feedback ratings of users; the goal is to predict the missing scores, i.e., estimate the user feedback for unseen content [22], [34], [38]. The RS's goal is thus to exploit similarities between users, and between contents [29] and predict user missing rankings; after discovering the said quantities, the RS usually lists the highest ranked items related to the actual viewed content.

In this work, our goal is *not* to build a RS from scratch, but use existing ones in order to build a network-friendly one for long viewing sessions, and that is able to promote low network cost content in favor of: (a) network operator (NO), and (b) user quality of experience (QoE). To this end, our algorithm is based on information that it receives from a State-of-Art RS, from which we need the predicted similarity scores *between contents*; hence, for us, those scores is our input, and we consider it as a recommendation ground truth. The output of our RS is a recommendation list that achieves the joint goals (a) and (b), rather than just promoting most related (i.e. highest ranked) content. We formalize in our paper 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 $\mathcal{K} \setminus \{i\}$. The similarity of each $j \in \mathcal{K}$ is quantified by the value $u_{ij} \in [0,1]$, forming the K-length row vector $\mathbf{u}_i$ with $u_{ii} = 0$. This information is summarized in the square non-symmetric $K \times K$ matrix U, with 0 values in the diagonal. We further denote by $\mathcal{U}_i(N)$, the set with the $N < K$ highest u_{ij} values related to content i. Note here 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 the recommendation batch w given for content i, if it includes items j with high u_{ij} values. We refer to this quantity as the user satisfaction and we simply denote it by Q_i. More specifically, Q_i is measured per viewed content i and recommendation batch w, and only depends on the entries of U, the size N of the batch, the recommendation policy

[1] The user profile, expressed by $\mathbf{p}_0$ is affected by the RS over time, however, for the time-scale we are interested in, we assume it remains fixed.

and is consequently quantified by the ratio

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

where i indicates the state $\in \mathcal{K}$ and w the batch. The denominator in (1) is the maximum batch quality Q_i^{max} so that $Q_i(w) \in [0, 1]$. Importantly, the expression states that the higher the sum u_{ij} of the recommendation batch, the happier the user is.

Finally, the RS has at its disposal statistics over the history of aggregate user requests and can estimate global content popularity $\mathbf{p}_0$. We assume that each content i can be requested independently of the RS suggestions, just through the search bar of the application, with probability $p_0(i) > 0,\ \forall i \in \mathcal{K}$. This probability distribution is the second RS input. Both the $\mathbf{p}_0$ and the matrix U is information that the RS can measure empirically over time.

2.3 Network cost model

The goal thus of a network-friendly RS is to (slightly) modify step 2 (see the list at the end of 2.1) in order to introduce network cost awareness. We argue that due to the impact of RS on user requests, the sequence of delivery costs $\{c(S_t)\}_{t=0}^{L}$ the operator experiences, will depend on the RS policy, where S_t is the random state visited at t and $c(S_t)$ is the cost of visiting state S_t. Hence, our primary objective is to come up with recommendation policies R (to be defined more formally later in the section), which promote low-cost contents and ultimately minimize the session's mean cost, while at the same time weighing in the user's natural preference for high content relevance. Mathematically, the objective alone can be written as:

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

The level of abstraction we have used in the cost modeling allows the flexibility for a variety of network and content characteristics to be integrated.

We assume a hierarchy of M tiers. The access cost of each tier is $c_1 < \cdots < c_M$, meaning that tier-1 lies closest to the users. Note that this is a simply *a* way to model the cost, but item characteristics like size and popularity can impact item cost *inside the same tier*. In fact, costs can be easily generalised to include sizes and other aspects, in which case we would have one cost per item (rather than tier). However, to facilitate the networking aspect of the paper, we will be discussing the rest of the paper as items in the same tier have similar characteristics and thus (more or less) same cost.

We further assume it is the CP that decides which content goes to which tier and thus no privacy issues arise (e.g. due to https [16] or a need for a 3rd party (e.g. network operator) to disclose sensitive information to another entity. We consider the "prefetch" model, according to which the CP caches content statically for the timescale of interest (e.g., a day). This paradigm is fit for today's networks, since the cached content [4] is usually in the order of 100 of MBs or more, and thus dynamic replacement would end up being very costly for the backhaul.

According to the above assumptions, one could model, for example, the basic OpenConnect CDN architectures with two main tiers, the set of OpenConnect boxes placed and operated (by Netflix) in operator core networks [7], with low delivery cost c_1, and the Netflix data center which resides deeper inside the network with c_2. Similarly Google, who collaborates with ISPs and NOs at its points of presence (PoP), decides what content to prefetch in the edge nodes/servers [4]. The delivery of this content is very efficient for the network (c_1) due to proximity of the nodes, however Google can also fetch content from its data centers, with much higher cost (c_2) (as this demands backhaul

usage). Finally, looking more towards future architectures, one could imagine a 5G/5G+ wireless virtualized network, where a CP has an end-to-end slice to the BS (RAN) resources [5], and it can manage additional tiers closer to the user consisting of *more but smaller-sized caches*.

2.4 Recommendation policies

Our primary focus in this work is to come up with recommendation policies for arbitrary user sessions in terms of average length and user behavior. The policies should aim at minimizing the expected session network-cost, while at the same time guaranteeing a good level of user satisfaction. Before formally defining the optimization problem in the next section, we will present here in detail what is a policy and how it is modeled in our framework; we will present reasonable heuristics and we will argue why optimisation with look-ahead is more promising than myopic approaches. As mentioned previously, when the user visits file i, the RS can propose 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 recommendation policy, we need to associate each recommendation batch $w \in \mathcal{A}_i$ with a frequency of use $\mu_i(w)$. This gives rise to two classes of policies.

- *Deterministic:* Only one 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, we might see a different N-tuple of contents, chosen randomly.

The sum of frequencies of all the batches related to i should sum up to 1. It is easy to see that using this brute force approach, given the exploding cardinality of the action set $\mathcal{A}_i$, leads to $\binom{K-1}{N}$ variables *per item* over which we must optimize. As an example, for $K = 1000$ and $N = 3$ the RS needs to introduce 165 Billion unknown μ's.

2.4.1 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 summarise their information as follows,

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

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 *recommendation matrix* and refer to it as the *recommendation policy*. We have thus reduced the unknowns to just K^2, a considerable improvement!

We present here a very important remark: the definition of a recommendation policy R through the r_{ij} frequencies, can allow to generate recommendation batches with the optimal $\mu_i(w)$ 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 [9, 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}. To enumerate all possible batch-actions, we can pick the different N-sized combinations in [9, Fig.1] and determine the probability of a specific batch w, by its width. 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 $N > 2$.

2.4.2 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 immediate next request.

- **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 $\mathcal{U}_i(N)$ in order to maximize user satisfaction (ties broken uniformly).
- **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 budget N to the most related items and the remaining to the lowest cost items. If any of the lowest cost and most similar items coincide, then simply assign the remaining budget to lowest cost. Essentially, q acts as a knob, for $q \to 1$, the policy is R_Q, while for $q \to 0$, the policy is R_C.

2.4.3 A motivating example in favor of look-ahead policies. We will show here an example where the above heuristic myopic policies are strictly suboptimal in the long-term, and new policies with look-ahead should be proposed, which offer a better cost vs quality trade-off for long sessions. Consider as an example the following content similarity matrix for catalog $\mathcal{K} = \{1, 2, 3, 4\}$,

$$U = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix}. \tag{5}$$

Using this, we want to compare various RS policies, which recommend exactly $N = 1$ item per viewed content. To better illustrate the key message, we'll assume a simple user that always follows the recommendation, provided this has an *average* (long-term) quality of q. We will also assume that the session length goes to infinity (i.e. $\lambda \to \infty$). All content has cost $COST > 0$, except item "1" which is cached and has cost $c_1 = 0$.

We first evaluate the aforementioned q-mixed policy. Following its definition, using as input matrix U and *excluding self-recommendations*, the resulting policy is written as

$$R_{MIX} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1-q & 0 & q & 0 \\ 1-q & 0 & 0 & q \\ 1 & 0 & 0 & 0 \end{pmatrix}, \tag{6}$$

where the quality constraint is naturally included by using q budget on the related item. Here, the recommendation budget of $N = 1$ slot is split among lowest cost and most relevant, for viewed objects “2" and “3"; for viewed object “1" (which is the cached one) all the budget is given to its most related object, whereas for viewed object “4" the lowest cost and most related object coincide, so that all the budget is allocated to “1". Hence, the user starts from any initial content, and keeps moving according to (6). Note that this matrix is stochastic, and defines a simple Markov walk over the content catalog. Starting from any initial content the user will move randomly among items in the catalog, following (6).

We will also consider the Top-N ($q = 1$) and Low Cost ($q = 0$) policies for comparison. Note however that the latter is "infeasible" for this U matrix if $q > 0$ (i.e. it violates the constraint), but we still include it as a lower bound on the total cost. Our goal is to primarily compare the long-term cost achieved by all the feasible policies that do respect the desired recommendation quality level q. Finally, we have also derived the “optimal” policy for this scenario (minimizing the long term cost subject to the quality constraint), according to the MDP optimization framework introduced in Section 3. The optimal recommendations are the following:

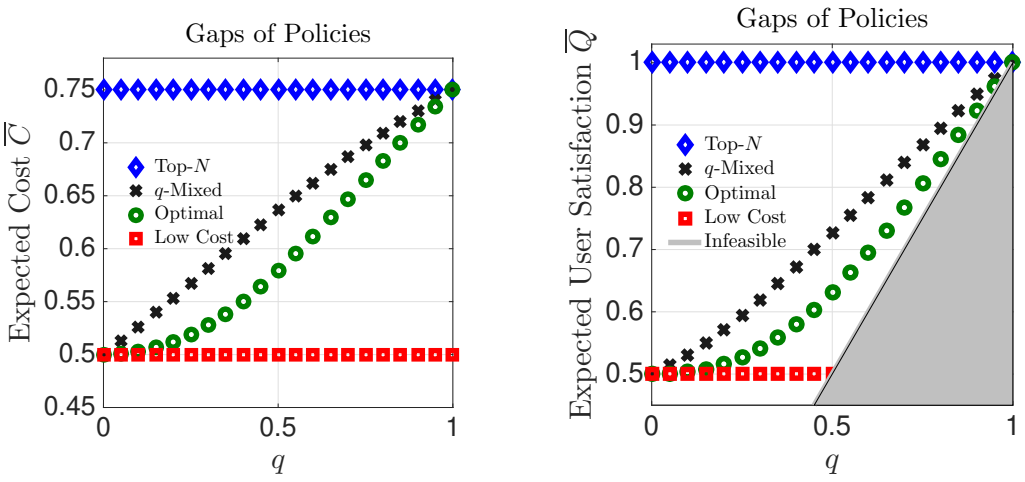
$$R_{OPT} = \begin{pmatrix} 0 & q & 0 & 1-q \\ 1-q & 0 & q & 0 \\ 1-q & 0 & 0 & q \\ 1 & 0 & 0 & 0 \end{pmatrix}. \tag{7}$$

As we can see, the optimal recommendations differ from the myopic q-Mixed policy. Specifically, the optimal policy sometimes recommends items that are neither low cost, nor most related. Specifically, while satisfying the per-viewed-content quality constraint, the $N = 1$ budget of content “1" is split among the most relevant item “2" and the unrelated *and* uncached item “4", with the rationale that after viewing “4" the low cost content “1" always follows.

In Figure 1 we plot the expected cost $\overline{C}$ and mean user satisfaction $\overline{Q}$ of all four policies, given $COST = 1$, as a function of the min RS quality q. We measure the long-term average cost by counting the sum of content costs accessed along the session using (2), and the instantaneous user satisfaction as defined in (1).

The figure shows that the optimal look-ahead policy achieves lower average cost with a slightly lower offered quality as trade-off, compared to the q-Mixed. The Top-N and Low Cost policies are inflexible. The first gives the highest quality but does not account for cost; the second gives the lowest cost but with the lowest content quality. The optimal policy minimizes the cost objective, within the quality constraint q.

Essentially, this toy example illustrates why myopic policies can be far from optimal for sessions of arbitrary length. In the next sections, we will formalize this intuition, describing and solving a more generic version of such optimal look-ahead recommendation policies. The main notation is summarized in Table 1.



(a) Expected Cost

(b) Expected User Satisfaction

Fig. 1. Toy Example: vs q - Optimal policy and Heuristics

Table 1. 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
$\mathbf{u}_i$	Vector of similarity values for all items related to i
U	Adjacency matrix, $U = [\mathbf{u}_1^T, \ldots, \mathbf{u}_K^T]$
$\mathcal{U}_i(N)$	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
R	Recommendation policy, i.e., the $\{r_{ij}\}$ values
$Q_i(w)$	User satisfaction by the recommendation batch w
q	Lower level of Q_i enforced by RS
$cost_i$	Cost of delivering from cache i
c_i	Delivery cost of content i
S_t	State/Content visited at time t

3 PROBLEM FORMULATION AND SOLUTION

We will now cast the problem of optimal sequential recommendations as a Markov Decision Problem (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.

3.1 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 complete content catalog. Following the discussion in the previous section about per-item frequencies, the action set $\mathcal{A}$ in our formulation is the set of all $K \times K$ real recommendation matrices R, whose entries $r_{ij} \in [0, 1]$ determine the frequency of suggesting item j when viewing content i. We assume that 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; we will specify the P_{ij} in the following section where we discuss different models of user behavior in detail. We assume that the RS *knows* which type of user behavior (i.e., the P_{ij} dynamics, and the α_{ij}) is dealing with, and optimizes the actions accordingly. Importantly, 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 integrated in our framework. Finally, a random item sequence $\{S_0, S_1, S_2, \dots\}$ viewed by the user, where $S_t \in \mathcal{K}$, also 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. In MDP terms, the latter corresponds to the *immediate* cost of visiting state i (requesting content i) for the NO. These costs are fixed as the caching is fixed, and could thus take arbitrary values; they are considered as a known input to our MDP[2].

The following expression gives the transition probability of state evolution in a general way, letting room for further assumptions (user bahavior) to be integrated later on in the model

$$P_{i\to j} = P_{ij}^{rec} + P_{ij}^{rand} = \alpha_{ij} \cdot r_{ij} + (1 - \sum_{l=1}^{K} \alpha_{il} \cdot r_{il}) \cdot p_0(j). \tag{8}$$

The above expression is somewhat reminiscent of the random web surfer transitions for PageRank [32], [15], and has the following interpretation. A user finds herself in item i, and w.p. r_{ij} the item j appears on the list of i; given that j appeared on the RS list, the user clicks on item j w.p. α_{ij} (the latter represents the "user willingness" to make this transition through the recommendation link). The left summand of (8), i.e., the quantity $P^{rec}(i \to j) = \alpha_{ij} r_{ij}$, indicates the probability of the user making the $i \to j$ transition through the recommendations. Observe that in the case of probabilistic recommendations ($r_{ij} \in [0, 1]$), $P^{rec}(i \to j)$ is time varying, as it depends on the realization of the recommendations (what items were suggested). In that case, $P^{rec}(i \to j)$ represents the *expected probability* of this transition; similarly, for probabilistic recommendations, the right summand, i.e., $(1 - \sum_{l=1}^{K} \alpha_{il} r_{il})$, stands for the *expected rejection rate*. The user transits to j through the search bar when they reject recommendations *and* choose to search for j ($p_0(j)$). On the other hand, when $r_{ij} \in \{0, 1\}$ (deterministic recommendations), $P^{rec}(i \to j)$ and the rejection rate are obviously both time-invariant, as every time the user lands on some content i, they will see the exact same items on the RS list.

[2]One could instead incorporate model the quality of experience gain g_i of a user viewing item i (e.g., $g_i = 1 - c_i$, thus resulting in an MDP maximizing the long term QoE.

Finally, to see why (8) describes exactly this process, we substitute r_{ij} as in (3) as follows,

$$P_{ij} = \alpha_{ij} \sum_{w \in \mathcal{A}_i} \mu_i(w) \mathbf{1}_{\{j \in w\}} + (\underbrace{\frac{1}{N} \sum_{j=1}^{K} \sum_{w \in \mathcal{A}_i} \mu_i(w) \mathbf{1}_{\{j \in w\}}}_{\overset{(4)}{=} 1} - \sum_{l=1}^{K} \alpha_{il} \sum_{w \in \mathcal{A}_i} \mu_i(w) \mathbf{1}_{\{l \in w\}}) p_0(j) =$$

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

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 $i \to j$.

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. $P_{i,j} > 0$ in (8). It suffices to consider $p_0(j) > 0 \; \forall \; j$ and $\sum_{l=1}^{K} \alpha_{il} \cdot r_{il} < 1$.

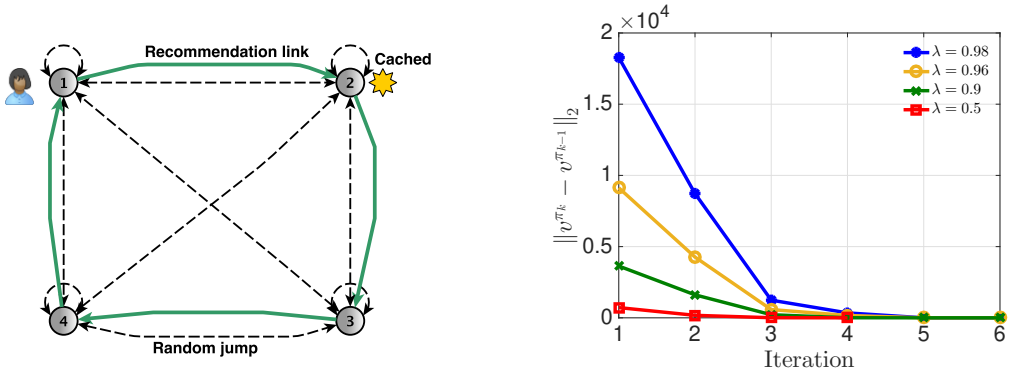
Example content transitions. To conceptually grasp the recommendation based transition probability, we provide Fig. 2(a); the recommendation variables have been decided, and the Markov chain states are represented as nodes on a graph, over which the user transits. In particular, the user can transit from item to item in two ways: either through a recommendation (green links), which is what we optimize, or through the random jump (black dashed links). Specifically, we recommend $N = 1$ item for every content, but since we do *probabilistic* recommendations, we could have $> N$ outgoing recommendations links; additionally, for every content there are exactly $K = 4$ random jumps towards *all* other items in the catalog (including self-arrows). To better connect (8) to the figure, we can think about the random surfer model [32], who transits through a link w.p. α, and then chooses one of the outgoing links at random, or does a random jump w.p. $1 - \alpha$. In that case, the intensity of the (recommendations) and the random jump links would be:

$$P_{ij}^{rec} = \alpha_{ij} r_{ij} = \alpha \frac{1}{N} r_{ij},$$

$$P_{ij}^{rand} = (1 - \sum_{l=1}^{K} \alpha_{il} r_{il}) p_0(j) = (1 - \alpha \frac{1}{N} \sum_{l=1}^{K} r_{il}) p_0(j) = (1 - \alpha) p_0(j).$$

3.2 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 starts from a given arbitrary state S_0, whose cost is not accounted for since it is outside the recommender's control. Then, for a given policy R, the rest of the visited states is random (source of randomness: user behavior) *and is influenced* by policy R (our control). As every request S_t accounts for a unique known cost c_{S_t} (the cost of fetching the item to the user), the sequence of observed costs $c(S_t|R)$, with $t = 1, \ldots, L$, is also random and a function of R. Note that the costs in consecutive states are *not* I.I.D., given the Markovian request access pattern of the user. The total cost induced by the requests of the user is $\sum_{t=1}^{L} c(S_t|R)$; our objective is to select the policy R that minimizes the *average* total cost.



(a) Transitions (b) Policy Iteration Convergence

Fig. 2. User transitions, and PI convergence

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*

$$v(s) = \mathbb{E}_s \left[\sum_{t=1}^{L} c(S_t|R) \right] = \mathbb{E}_s \left[\sum_{t=1}^{\infty} \lambda^{t-1} \cdot c(S_t|R) \right], \tag{9}$$

where the $\mathbb{E}_s$ stands for conditioning on the starting state being $S_0 = s \in \mathcal{K}$, [33, eq. 4.13].

Equality in (9) holds because the random session length L is assumed to be distributed as a geometric distribution $Geom(\lambda)$. The expectation of the total cost is found by applying the law of total expectation

$$\mathbb{E}_s \left(\sum_{t=1}^{L} c(S_t|R) \right) = \sum_{l=1}^{\infty} \mathbb{P}(L = l) \cdot \mathbb{E}_s \left(\sum_{t=1}^{l} c(S_t|R) \right). \tag{10}$$

We refer the reader to [33, 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

Problem 1 (Main Optimization Problem).

$$v^*(i) = \min_{R} \left\{ \mathbb{E} \left(\sum_{t=1}^{\infty} \lambda^{t-1} c(S_t, R) \mid S_0 = i \right) \right\} \forall\, i \in \mathcal{K}, \tag{11}$$

$$\text{subject to} \quad \sum_{j=1}^{K} r_{ij} = N, \ \forall i \in \mathcal{K}, \tag{12}$$

$$0 \leq r_{ij} \leq 1, \ \forall i, j \in \mathcal{K}, \tag{13}$$

$$r_{ii} = 0, \ \forall i \in \mathcal{K}, \tag{14}$$

$$\sum_{j=1}^{K} r_{ij} \cdot u_{ij} \geq q \cdot Q_i^{max}, \ \forall i \in \mathcal{K}, \tag{15}$$

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

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

Constraint (12) 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.

An optional hard constraint on the average user satisfaction from the recommendation batch is introduced in (15). If active, the RS is restricted to maintain an average user satisfaction $\geq q$ for every item $i \in \mathcal{K}$, regardless of how the user reacts to good/bad recommendations. However, as we will see subsequently, such an explicit constraint might not be necessary in user models where the user can immediately assess the quality u_{ij} of the recommended item, and adjust her click probability α_{ij} accordingly. We denote the feasible set of policies for viewed content i by $\mathcal{R}_i = \{r_{ij} : (12), (13), (14), (15)\}$. The complete 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.

3.3 Optimality

The optimal solution to **Main Optimization Problem**, i.e., the optimal vector $\mathbf{v}^* = [v^*(1), \ldots, v^*(K)]$ 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. Finding the optimal value vector, is equivalent to finding the optimal policy. The optimal value vector must satisfy the following K (Bellman) equations, one per state,

$$v^*(i) = c_i + \lambda \min_{\mathbf{r}_i \in \mathcal{R}_i} \left\{ \sum_{j=1}^{K} P_{ij}(\mathbf{r}_i) \cdot v^*(j) \right\} \forall i \in \mathcal{K}. \tag{16}$$

where $P_{ij}(\mathbf{r}_i)$ is defined in (8) and c_i indicates the *immediate cost* of visiting state i. We can apply well established iterative algorithms to solve these equations [33]; we choose here the well-known policy iteration, which is known to converge for infinite horizon MDP with discounts $\lambda \in (0, 1)$ [33] [Theorem 6.4.6, p. 180], at least linearly. Note however that, unlike "vanilla" MDPs with *discrete actions*, in each such iteration we are required to also solve a minimization problem in the K-sized variable $\mathbf{r}_i$ for each state i, see (16). A key contribution of our work is the use of recommendation frequencies per item, instead of discrete recommendation batches; this way the complexity of these interior minimization problems (taking place at *every* step of the policy iteration algorithm) is radically reduced. Below we describe our main policy iteration algorithm.

The minimization step of state/content i involves *only* the K recommendation variables in $\mathbf{r}_i \in \mathcal{R}_i$. 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. Before ending this discussion, we present Fig. 2(b), where the convergence of PI algorithm is depicted. More specifically in each iteration, a new policy π_k is computed and this corresponds to some new value vector v^{π_k}. On the y-axis, the values represent the $e_k = ||v^{\pi_k} - v^{\pi_{k-1}}||_2$ (which all converge to zero), and on

Algorithm 1 Policy Iteration

$n \leftarrow 0$, Initialize R_n (as R_Q for warm start)
repeat
  $\mathbf{v}^* \leftarrow$ Policy Evaluation(R_n)
  for $i \in \mathcal{K}$ **do**
    $\mathbf{r}_i \leftarrow \text{argmin}_{\mathbf{r} \in \mathcal{R}_i} \left\{ \sum_{j=1}^{K} P_{ij}(\mathbf{r}_i) \cdot v^*(j) \right\}$
  end for
until $||R_n - R_{n-1}|| < \epsilon$
return R_n

the x-axis we see the iterations of PI; we have plotted the convergence for different values of λ. It is worth observing the effect of the horizon on the convergence; to discuss this, we also need to remember that our policy is initialized as the top-N policy, which is really far from the optimal one. Therefore, the larger the horizon, the larger the first step is (in terms of value improvement), which explains why the larger mean horizon translates to bigger initial values and faster decrease.

Dynamic caching. An alternative to static caching is dynamic algorithms that could be predecided by the NO, e.g., LRU, LFU etc. In our MDP, the caching costs are time invariant, and do not depend on any source of randomness (user behavior); however, if the caching placement is subject to the user randomness, then the caching costs *are random as well*. This further complicates the MDP for the following reason: the caching costs c_i (cost of state) in (16) are a priori unknown. To bypass this, one can include the cache state in the system's state, making it a tuple (last accessed item, $c_1, \ldots, c_K$) (in principle we would need the cost of every item in the catalog). Now, for a state for which the "last accessed item" was i, we would have many different caching configurations, which could uniquely determine the cost of item i. This approach does not scale well, however, if one conditions on the caching protocol, it is likely that structural properties of the problem may simplify the problem.

3.4 User session modelling

In this work, we have modelled the length of the session, as an integer counting the requests made by the user. Modellingwise, there are two immediate options for this integer: finite and (a) deterministic, or (b) random. Opting for (a), would lead to a deterministic user session, which is of course an unrealistic assumption. However, our MDP can be used to solve the deterministic horizon case, but that would result in a "shortest-paths" like formulation, which makes the solution computationally harder. In that case, the recommendation policy *is not time-invariant*; put simply, the recommendation policy at item i in step n and m (with $n \neq m$) would be different. We opted for (b) (random horizon) and more specifically $L \sim G(1-\lambda)$, as it is a natural alternative for our problem, it has nice solution properties, and results in a more tractable algorithm. Interestingly, this choice buys us a flexibility on the range of problems we can tackle, as we can now solve the recommendation problem for an arbitrary average session length $\overline{L}$, by controlling λ (the discount factor). Note that existing works in the literature analyze *infinitely long* sessions, like in [17, 18], which is of course unrealistic, while the MDP framework introduced in the current work, uses λ as a tuning parameter. Let us observe some special cases.

Case: $\lambda \rightarrow 0$**.** The objective function in (11), becomes $v(s) = \mathbb{E}_s[0^0 c(S_1) + 0^1 c(S_2) + \ldots] = \mathbb{E}_s[c(S_1)]$ (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 $\lambda \rightarrow 0$, 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

$$\min_{\mathbf{r}_s \in \mathcal{R}_s} \left\{ \sum_{j=1}^{K} P_{s,j}(\mathbf{r}_i) \cdot c_j \right\} \forall\ s\ \in\ \mathcal{K}. \tag{17}$$

Setting $\lambda = 0$ in Main Optimization Problem, 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 [33, 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, in terms of length, the session is somewhere in the middle ($0 < \lambda < 1$), and in terms of randomness, it could be non geometric. However, the RS could measure empirical averages of the user sessions, fit a geometric distribution (i.e., and determine a λ that is representative of the user session), and solve the problem with our MDP. Obviously, when the user session has statistically very different characteristics, our solution could be considered as an approximation; we defer researching this problem for a future work.

4 TWO MODELS OF USER BEHAVIOR

We will consider two potential user models, that differ in their reactions towards high/low quality recommendations, and investigate how a network-friendly RS should optimally adapt to each. We stress that these models are still somewhat simplistic, and the question of realistically modeling all sorts of users (as well as how to distinguish them, e.g., via model-free approaches), is well beyond the scope of this paper. Our main goal is to showcase the flexibility of the MDP framework in incorporating various modeling assumptions, as well as the potential impact of such user differences on the optimal performance.

4.1 The flexible user

Assumptions on the user. The first model captures a somewhat "flexible" or more "curious" user. Described informally: a flexible user remains equally curious about any recommended item, provided that the long-term quality of recommendations remains reasonably good. We attempt to capture such a user with the following assumptions:

$$\textbf{flexible user}: \ \alpha_{ij} = \alpha/N \text{ and } \mathbb{E}[Q_i] \geq q\ \forall\ i.$$

The average RS quality *must* exceed a minimum threshold q -see (15)- and given it does, the user's *expected* clickthrough 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. The user transition probabilities, i.e. (8), now become

$$P\{i \to j\} \quad = \quad \frac{\alpha}{N} \cdot r_{ij} + (1-\alpha) \cdot p_0(j), \tag{18}$$

where notice that $\sum_{j=1}^{K} \frac{\alpha}{N} \cdot r_{ij} = \alpha$. The average rejection rate, with which the user ignores the recommendations is the probability $1-\alpha \in [0,1]$, and consequently the click-through probability is just α. Note that this type of user is reminiscent of the PageRank web surfer, where the user clicks any hyperlink with a fixed probability, or jumps to a random page [32]. Using (18), the Bellman equations take the form

$$v^*(i) = c_i + \bar{v} + \lambda \frac{\alpha}{N} \min_{\mathbf{r}_i \in \mathcal{R}_i} \left\{ \sum_{j=1}^{K} r_{ij} \cdot v^*(j) \right\} \forall i \in \mathcal{K}, \tag{19}$$

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

Problem 2 (OP-flexible user).

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

$$\text{subject to} \quad \mathcal{R}_i = \{r_{ij} : (12), (13), (14), (15)\},$$

where $q \in [0,1]$ is the tuning quality parameter of the constraint (15).

Lemma 3. *In the case of the "flexible" user, the greedy improvement step of the policy iteration reduces to solving the OP-flexible user which is an 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 policy iteration can be solved using standard software (e.g. CPLEX). Note here, that solving the MDP for "flexible" user, returns a randomised policy in general, due to the constraint (15). 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 (15) 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 (15) reads

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

$$(u_{ij} - u_{im}) \cdot x \geq 0. \tag{20}$$

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 $v^*(i)$.

Table 2. Summary of Models

	flexible	picky
User selects item from recommendations	uniform	proportional to u_{ij}
User satisfaction model	hard constraint	in item selection

Case (a): If $1 \leq i \leq N$ then

$$v^*(i) = c_i + \bar{v} + \lambda \left(\sum_{j=1: j \neq i}^{N} v^*(j) + v^*(N+1) \right), \tag{21}$$

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

$$v^*(i) = c_i + \bar{v} + \lambda \sum_{j=1}^{N} v^*(j). \tag{22}$$

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 (21))

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

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 (21) and (22) and we result in the exact same expression for their difference as above. Finally, for Pair-case (III): $N < i < j$, we use (22)

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

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. □

While for extreme q parameters the respective myopic policies Low Cost, Top-N are optimal, as explained earlier with the example of Section 2.4.3, the q-Mixed policy is often *suboptimal* for any intermediate q values. This will become evident in the validation section.

4.2 The picky user

Assumptions on the user. In the second model, we assume the user does not enforce a hard, fixed threshold q on the average recommendation quality, but her choice of a recommended item j is immediately affected by the perceived relevance of j to the currently watched item i

$$\textbf{picky user} : \alpha_{ij} = \frac{u_{ij}}{Q_i^{max}} \text{ and } q = 0.$$

Hence, the higher the u_{ij} of the suggested content j, the more probable it is for the user to click on j over the other $N-1$ suggestions. Therefore, this model represents some user who is very selective on what she chooses to watch and does not interact with the RS system unless she sees interesting suggestions.

MDP solution. Substituting $\alpha_{ij} = \frac{u_{ij}}{Q_i^{max}}$ we get

$$P\{i \rightarrow j\} = \frac{u_{ij}}{Q_i^{max}} \cdot r_{ij} + (1 - \sum_{m=1}^{K} \frac{u_{im} \cdot r_{im}}{Q_i^{max}}) \cdot p_0(j). \tag{23}$$

The rejection rate at item i (here is different per item i) is $1-\sum_{m=1}^{K} \frac{u_{im} \cdot r_{im}}{Q_i^{max}}$ and its compliment expresses the clickthrough rate onto recommended items. Hence, the closer the policy $\mathbf{r}_i$ is to Top-N (closer to offering Q_i^{max}), the more likely the user is to click on one of the recommended items. To derive the optimal policy of the "picky" user, we need to determine the optimization problem that arises during the execution of policy iteration (Step 4). Substituting the expression for P_{ij} in (23), the Bellman Equations gives rise to the following

$$\begin{aligned}
\underset{\mathbf{r}_i \in \mathcal{R}_i}{\arg\min} \sum_{j=1}^{K} \left(\frac{u_{ij}}{Q_i^{max}} r_{ij} + (1 - \sum_{m=1}^{K} \frac{u_{im}}{Q_i^{max}} r_{im}) p_0(j) \right) v_j = \\
\underset{\mathbf{r}_i \in \mathcal{R}_i}{\arg\min} \left\{ \sum_{j=1}^{K} r_{ij} u_{ij} v_j - \sum_{j=1}^{K} \sum_{m=1}^{K} \left(r_{im} u_{im} \right) p_0(j) v_j \right\} = \\
\underset{\mathbf{r}_i \in \mathcal{R}_i}{\arg\min} \left\{ \mathbf{r}_i^T \cdot \mathbf{u}_i \odot \mathbf{v} - \mathbf{r}_i^T \cdot \mathbf{u}_i \cdot \mathbf{p}_0^T \cdot \mathbf{v} \right\},
\end{aligned} \tag{24}$$

where $\mathbf{u}_i \odot \mathbf{v}$ denotes element-wise multiplication. In this model, the constraint (15) is removed from $\mathcal{R}_i$ or rendered inactive by setting $q = 0$. Thus, the optimization problem to be solved in the loop of policy iteration is the following

Problem 3 (OP-picky user).

$$\begin{aligned}
&\min_{\mathbf{r}_i \in \mathcal{R}_i} \left\{ \left(\mathbf{u}_i \odot \mathbf{v} - \mathbf{u}_i \cdot \mathbf{p}_0^T \cdot \mathbf{v} \right)^T \cdot \mathbf{r}_i \right\}, \\
&\text{subject to} \quad \mathcal{R}_i = \{ r_{ij} : (12), (13), (14), (15) \},
\end{aligned}$$

where $q = 0$ for the constraint (15).

Lemma 4. *OP-picky user is an LP which can be solved optimally in $O(K \log(K))$ operations.*

Proof. The calculations reveal a problem with linear objective with unknowns the $\{r_{ij}\}$, where all constraints are linear. More importantly though, observe that all weights in (12) are equal to one. Hence, the optimal solution is to assign the maximum possible budget, i.e. $r_{ij} = 1$, to the j associated with the lowest weight in the objective. Generalizing this observation, we assign $r_{ij} = 1$ to the N lowest weights of the constant vector $(\mathbf{u}_i \odot \mathbf{v} - \mathbf{u}_i \cdot \mathbf{p}_0^T \cdot \mathbf{v})$. Thus OP-picky user can be reduced into a sorting problem, which is known to need $O(K \log(K))$ steps. □

Corollary 1. *The optimal policy of OP-picky user is deterministic.*

5 VALIDATION

In this section, we evaluate the performance of the proposed MDP-based recommendation policies, and compare them with other myopic approaches. We arrive at useful conclusions and gain a better understanding over the structure of the optimal policies.

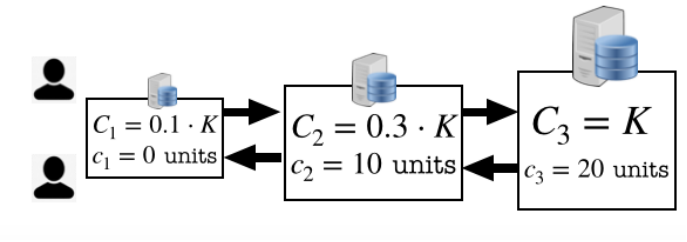


Fig. 3. Caches Representation

5.1 Simulation setup

Placement policy. In our simulations we assume, as it is common, that caches *further away* from the user fit more content, namely $C_1 = \lfloor 0.1 \cdot K \rfloor$, $C_2 = \lfloor 0.3 \cdot K \rfloor$, and $C_3 = K$, but also serve content at a higher cost, namely c_1 for level 1, $c_2 > c_1$ for level, etc. More specifically, $c_1 = 0$, $c_2 = 10$ and $c_3 = 20$. Our setup resembles the hierarchical setup of [19]. Notably, for the level 1 cache (the cheapest of all), we assume a capacity similar to the seminal work of [36], i.e., 10% of the catalog.

Moreover, we consider a uniform personal preference distribution $\mathbf{p}_0$ over the K contents i.e., $\mathbf{p}_0 \sim unif(1, \ldots, K)$. Since the the goal of this work is to study the positive effects of the RS's *network friendliness*, and not the pure caching gains from the possible skewness personal preferences of $\mathbf{p}_0$. Thus, since we have removed the skewness effect, we can choose which contents to place in which tier randomly. Moreover, in this way we isolate the effect of *different network-friendly recommendation policies*, and see which ones are able to better capitalize on the content graph and find more interesting tradeoffs *in the long run*. Moreover, since the caching policy is essentially random, the performance of our algorithm will be unaffected by the preference distribution $\mathbf{p}_0$. To sum up, the uniform popularity and the random caching policy set the stage for the RS and help us better understand the *true gains* that come only from the bias it creates towards the lower cost content.

Simulation and metrics. When we refer to the "flexible" or the "picky" user, we implement a user who behaves accordingly. The RS *knows* which user case is doing the requests and the recommendation policy induced is the optimal one computed by the MDP. In the case of the "flexible" user, the optimal RS policy is probabilistic, and in order to implement it and suggest N contents, we follow the method of [9], whereas the optimal policy for the "picky" user is deterministic. 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, and characterizes the duration of the session). Therefore,

$$C_L = \frac{1}{L} \sum_{t=1}^{L} c(S_t) \text{ and } Q_L = \frac{1}{L} \sum_{t=1}^{L} Q_{S_t}(w). \tag{25}$$

where C_L is defined in (2) and is the average cost of the session, Q_L the corresponding average user satisfaction, and $Q_{S_t}(w)$ is defined in (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.

Recommendation policies. For the two user models, the optimal policy is different; irrespective of the underlying user model we will refer to this policy as MDP optimal throughout in this section. Moreover, we will compare the MDP optimal policy with the policies described in Section 2.4 referred to as myopic.

5.2 Traces

To test the policies discussed throughout the previous sections, we need some content libraries and most importantly the respective content relation graph U (Section 2.2). To this end, we use three datasets to construct different U, two real ones and one synthetic.

MovieLens. We consider the Movielens movie rating dataset [21], 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 [1]. From this, we choose the largest component of the library and build a graph of 2098 nodes (contents) if there is a link from $i \rightarrow j$. As the values of the dataset were $u_{ij} \in \{0, 1\}$, whenever an edge was found, we assigned it a random weight $u_{ij} \sim unif(0.5, 1)$.

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

Statistics. 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 $i \rightarrow j$ 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.
- $\deg_i^-(C_j)$: out-degree of node i directed *only* to nodes in the set C_j (the set of cache level $j = 1, 2, 3$).
- $D_C(i)$: we find the shortest paths of every node $i \in C$ to all the other nodes which are also $\in C$. This makes up for $\frac{M \cdot (M-1)}{2}$ shortest path distances over which we average.

We compute the last two quantities for both C_1 and C_2, but not for C_3, because this corresponds to the whole catalog, and will not give additional information.

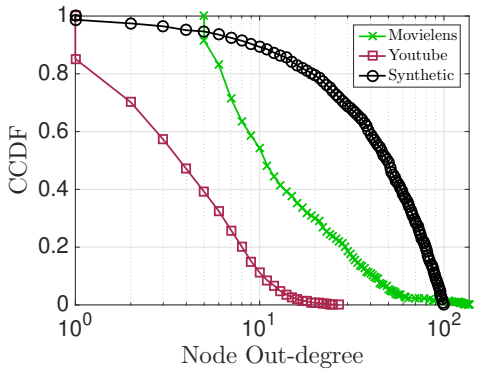


Fig. 4. Traces: Node Out-degree CCDF

	MovieLens	YouTube	Synthetic
Nodes	1060	2098	2000
Total Edges	20162	11288	99367
mean $\deg^-$	19.02	5.38	49.68
$std(\deg^-)$	19.61	4.16	28.84
mean $\deg^-(C_1)$	2.23	0.55	4.96
$std(\deg^-(C_1))$	25.78	0.66	12.65
mean $\deg^-(C_2)$	6.02	1.67	15.02
$std(\deg^-(C_2))$	155.78	2.72	85.42
mean D_{C_1}	1.27	1.95	1.09
mean D_{C_2}	1.10	1.32	1.03

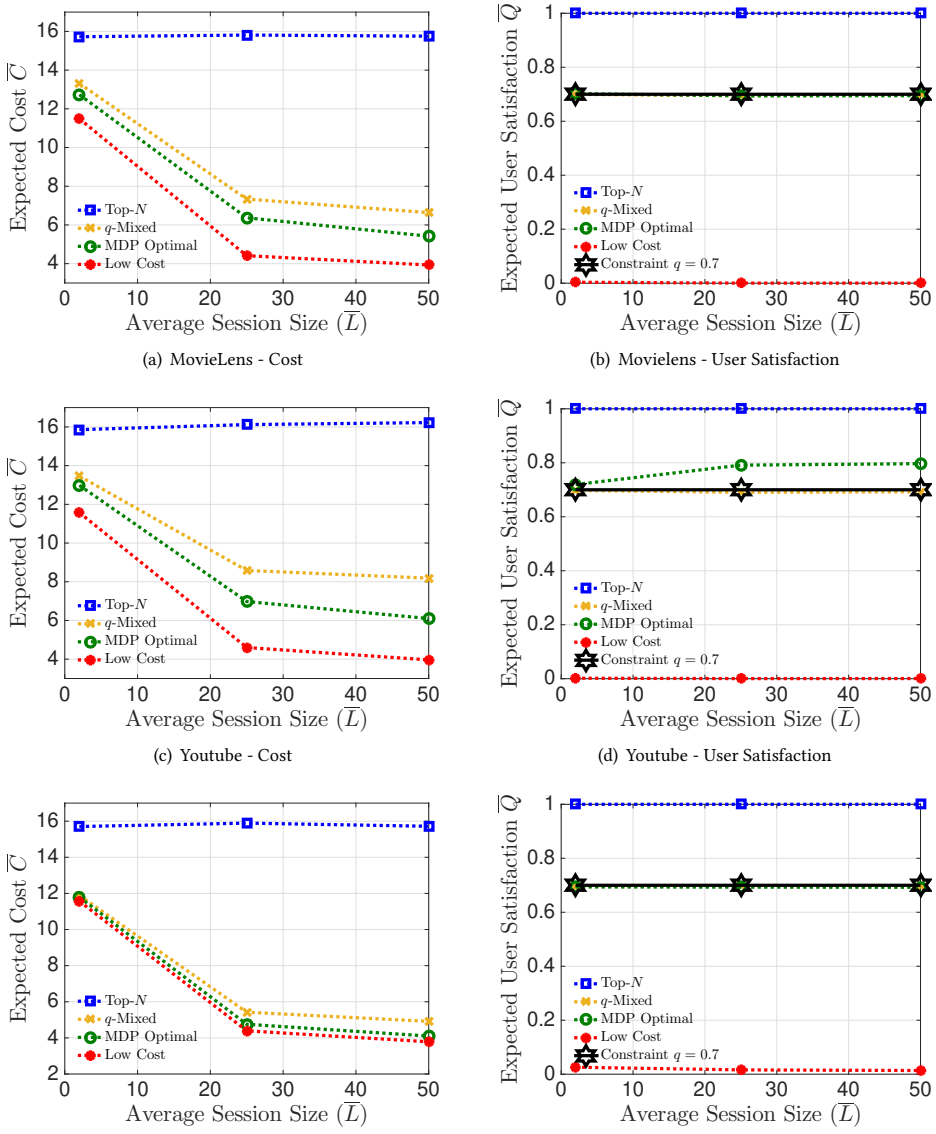
Table 3. Statistics - Datasets

On the x-axis we see the $\deg_i^-$ in logarithmic scale, and on the y-axis its ccdf. We can conclude for the two real traces, that the $\deg_i^-$ tends to be quite high only for a small fraction of the nodes. This metric will be useful to interpret the optimal policy's actions later.

Execution of the Policy Iteration 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 "flexible" user, i.e., OP-flexible user that arise were solved through CPLEX. It is important to note that the results of the MDP (both models discussed in the previous section) have been carried out using policy iteration with termination criterion being that policies between two consecutive iterations *are identical*.

5.3 Results: the flexible user

Effect of mean session size ($\overline{L}$). We first compare the average cost 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. 5, we vary the parameter λ (Section 2) to simulate random sessions with increasing mean size $\overline{L} = \{2, 25, 50\}$; we remind the reader that $\overline{L} = (1-\lambda)^{-1}$. We compare the performance of the optimal policies for the "flexible" user against the three myopic policies discussed in Section 2.4. For this simulation, we set $q = 70\%$ (for all datasets) in order to ensure high user satisfaction $\overline{Q}$ for the cost-oriented policies. This hard constraint of $\overline{Q} \geq 70\%$ is depicted in Figs. 5(b), 5(d) with a dashed grey line. For the q-Mixed policy, we set $q = 70\%$, hence it becomes 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 contrast, the Low Cost returns the best possible cost but is (as expected) *infeasible*. The policy 0.7-Mixed succeeds in lowering the cost while offering user satisfaction at the feasibility boundary. It is important to note that a very dense content graph translates to better $\overline{C}$ performance. Compared to the real datasets, the Synthetic U is more dense: each node/content has ≈ 50 neighbors, and ≈ 5 of them point towards C_1; this accounts for a $\overline{C}$ performance (see Fig. 5(e)) that is close to the Low-Cost policy, which simply suggests the most lowest cost items (the ones of C_1) by completely disregarding user satisfaction. We should however, also observe that the myopic 0.7-mixed policy is suboptimal, but is close to the MDP which is equipped with horizon and a vision of the request path. This relates to the fact that a very dense content graph translates to an



(a) MovieLens - Cost (b) Movielens - User Satisfaction

(c) Youtube - Cost (d) Youtube - User Satisfaction

(e) Synthetic - Cost (f) Synthetic - User Satisfaction

Fig. 5. Flexible user: Metrics vs mean session size $\overline{L}$ ($\alpha = 0.8$, $q = 0.7$, $N = 2$)

"easier" problem, where even myopic approaches work well. The relation matrices/graphs are in practice however -see our real datasets- much sparser.

Obs. #1: The MDP-optimal policy keeps the user satisfaction feasible while achieving the minimum $\overline{C}$ which is lower than all myopic policies. The relative gain between MDP and the q-Mixed is increasing with $\overline{L} : 1 \longrightarrow 50$, where at $\overline{L} = 50$ it reaches a plateau.

Obs. #2: The values of $\overline{C}$ we observe in the two real datasets (Movielens, Youtube) (see Fig. 5) are quite close, but are not the same. From Table 5.2, a random content of Movielens has 4 times the $\deg^-(C_1)$ Youtube has. For that reason, one might expect that Movielens would have lower cost than Youtube.

However, this is not the case, as we should also pay attention to the variances of the $\deg^-(C_{1,2})$ which are much larger in the Movielens dataset, in contrast to the Youtube case. Finally, it is important to remember that even if the dataset has many outgoing edges (and therefore the RS more items to consider)—we recommend 2 items, and in the Movielens dataset a randomly picked item has more than 2 (see Table 5.2) edges towards the cache level 1—the "flexible" user is hard to steer. The latter observation relates to the fact that the random walk resets (user goes to search bar) irrespective of the user satisfaction we are offering through the recommendations, and that the user chooses from the recommendations uniformly at random. The above argument, as we will see later, does not hold for the "picky" user, whose cost performance in Movielens is improved.

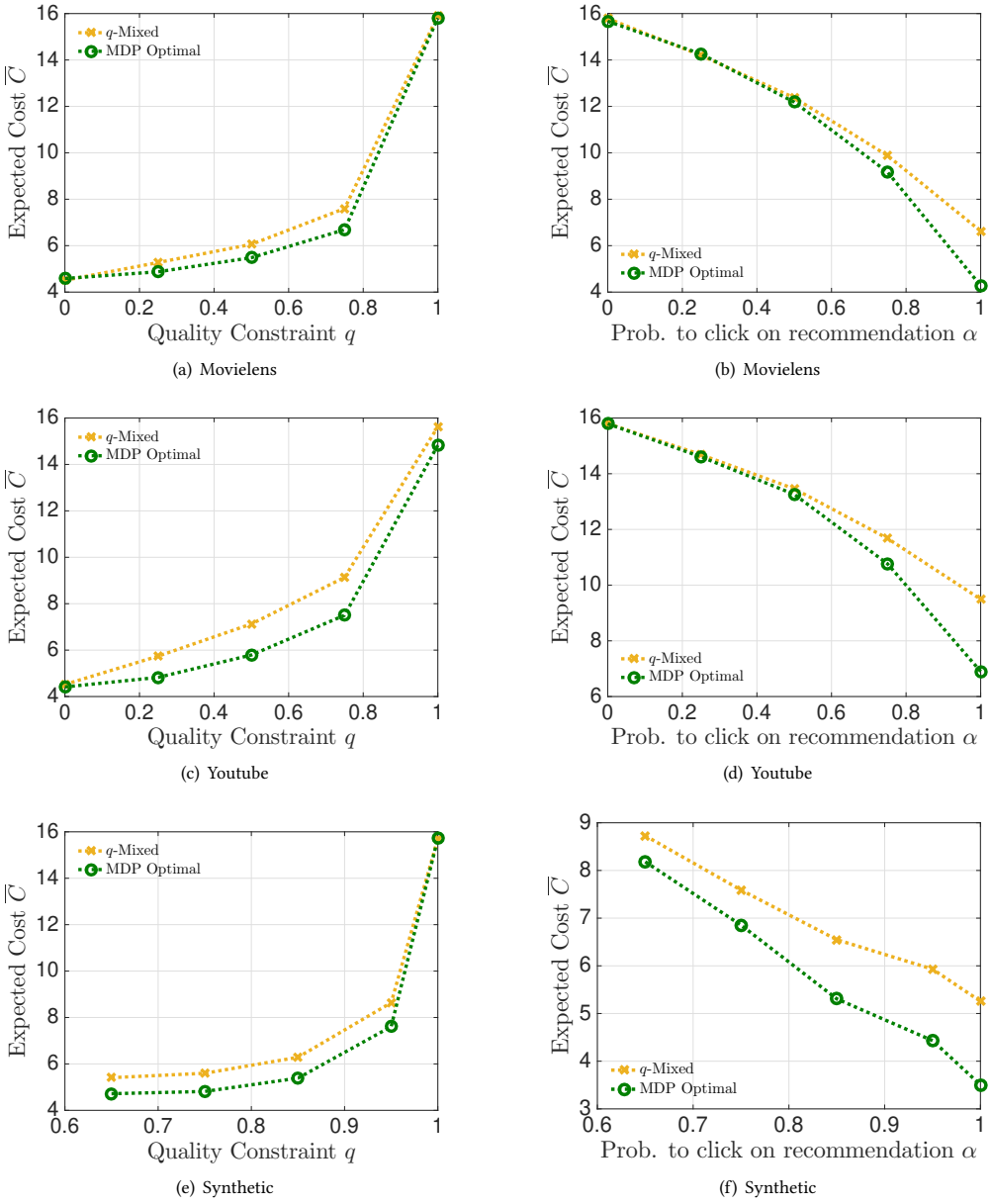
Obs. #3: In Youtube and Fig. 5(d), we can see that for $\overline{L} \geq 25$, $\overline{Q} > q$, and our RS offers *better user satisfaction than expected*. This can be answered by doing the following thought: Essentially, when the session becomes larger, the look-ahead vision of the MDP manages to drive the user in a neighborhood of "good" (low cache cost and related) items. Then, the policy is able to exploit the combination of caches and U, and achieve low cost by offering contents of high relevance to the user.

Note that the mean D_C is not that important for the "flexible" user. Essentially, low mean D_C suggests that the set C is very well connected through U but the important aspect of the "flexible" user is that she always rejects the recommendations with *fixed* $1-\alpha$, no matter how good the recommendations are. Therefore, if the user finds herself in the very convenient (for the network) neighborhood C, she can easily "escape" the RS well targeted actions because with fixed $1-\alpha$ she will always ignore our recommendations.

Effect of q and α. In the "flexible" user, we have two key input user parameters. For each dataset, in Figs. 6(a), 6(c), 6(e), we pick some α and tighten the quality constraint by increasing q. In the same fashion, in Figs. 6(b), 6(d), 6(f) we pick some q for every dataset and increase the value of α. In the real datasets, we vary q, α from $0 \rightarrow 1$, but on the Synthetic dataset, we focused on higher values of q and α (which are more applicable in practice), and increase the granularity of values. 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. 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, the MDP-optimal policy improves performance compared to the q-Mixed. Notably, the costs observed in Figs. 6(e), 6(f), the values are much lower compared to the datasets due to the fact that U (the content graph) is much denser. Finally, an interesting feature of Fig. 6(e), is that for many values of high $65\% < q < 85\%$ remains constant, which again relates to how denser is the Synthetic dataset compared to the real ones. We can also observe that the MDP-optimal policy is able to better exploit the increase of α (the willingness of the user to click on recommended items) 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 [2].

Effect of recommendation batch size (N). In Fig. 7(a), 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. Closely related to the **Obs. # 2**, we can see that irrespective of $\overline{L}$, the cost is becoming worse with the increase of N even for the Movielens graph which has a much larger $\deg^-(C_1) = 2.23$ and $\deg^-(C_2) = 6.02$.

Obs. #4: Network-friendly recommendations *is not* an easy task, but *many* network-friendly recommendations is even harder. To better grasp this, it helps to consider a myopic RS. To satisfy both parties (network and the user), when the



(a) Movielens (b) Movielens

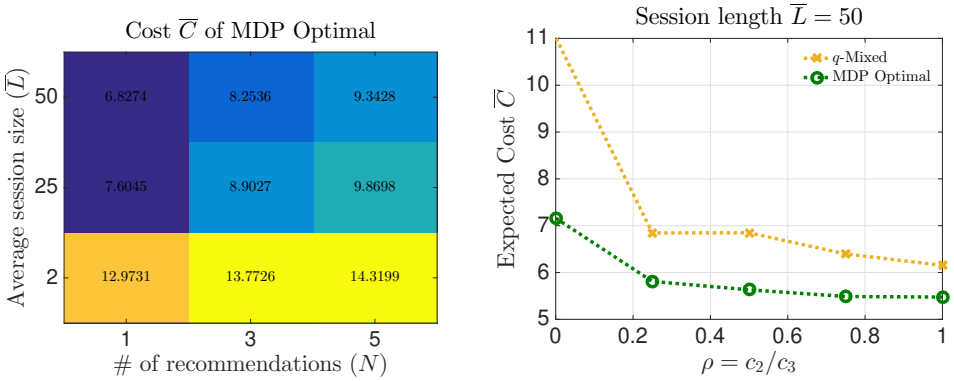
(c) Youtube (d) Youtube

(e) Synthetic (f) Synthetic

Fig. 6. Flexible user: Expected Cost per request, MDP-Optimal vs q-Mixed evaluated on $\overline{L} = 25$ requests ($N = 2$)

user is at content i, the RS must have many cached *and* related contents to recommend, which in principle are always less than cached *or* related.

Effect of caching costs. The last parameter we discuss is the effect of the relative delivery remote (c_2, c_3) costs. To this end, in Fig. 7(b), we fix $c_1 = 0$ and $c_2 + c_3 = 30$ units. We set c_2 as a fraction of c_3, i.e., $c_2 = \rho \cdot c_3$, and we vary



(a) Num. of Recommendations (N) and mean Session Size $\overline{L}$ ($\alpha = 0.75$, $q = 0.75$) - Synthetic

(b) ρ ($\overline{L} = 25$ $\alpha = 0.8$, $q = 0.8$, $N = 2$) - Movielens

Fig. 7. Flexible user: Expected Cost per request vs

$\rho = \{0.25, 0.5, 0.75, 1.0\}$. For $\rho = 1$, we have $c_1 = 0, c_2 = c_3 = 15$, so this captures the scenario of one cache of size $C_1 = 0.1K$ with $c_1 = 0$, and all remote items cost 15 units. When $\rho = 0$, means $c_1 = c_2 = 0, c_3 = 15$, and we have a bigger cache of size $C_1 + C_2 = 0.3K$, with zero cost, and a remote one with delivery of 30 units. We observe that the more we approach $\rho = 0$, which means bigger cache *and more* costly remote access (from inner servers with cost c_3), the more the MDP tries *not to* recommend items from there, essentially using its vision to bypass future requests from the costly servers.

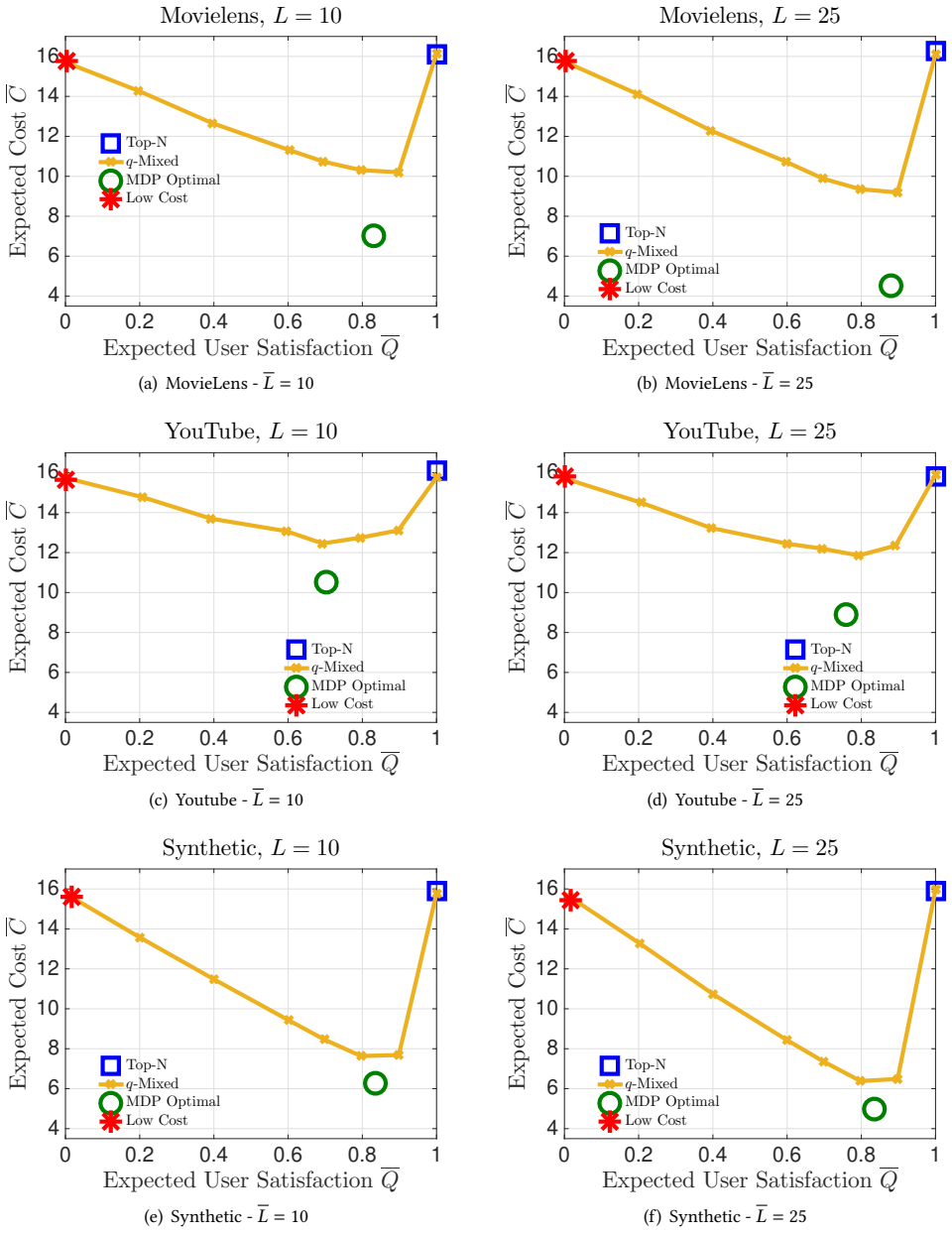
5.4 Results: the picky user

Effect of mean session size $\overline{L}$. We have so far established that network-friendly recommendation can achieve significantly better cost-quality tradeoffs than simple myopic policies for a "flexible" user, and more importantly, that such optimal tradeoffs can be efficiently calculated with our MDP framework. In this section, we investigate to whether similar conclusions hold for the "tougher" "picky" user.

We again first vary $\overline{L} = \{10, 25\}$ and observe the two performance metrics, $\overline{C}$ and $\overline{Q}$, see (25). We slightly change the way of presentation and do the following: each plot in Fig. 8 corresponds to one dataset and one value of $\overline{L}$ (both are indicated in the caption) and for each of these plots, we report the metric pair $(\overline{C}, \overline{Q})$ of the Low Cost, the Top-N and the MDP-Optimal policies *with a single point*. For fair comparison of the myopic q-Mixed policy, we list the metric pairs (i.e. $\overline{C}$ and $\overline{Q}$) for a variety of q values, i.e., $\{0.0, 0.2, 0.4, 0.6, 0.7, 0.8, 0.9, 1.0\}$.

The main observation is that the MDP-optimal policy, achieves by far the best $\overline{C}$ of the four policies, while having quite high $\overline{Q}$ values. In the "tougher" case of the "picky" user, none of the three myopic policies is able to lower the mean cost of the request $\overline{C}$ significantly. The Top-N policy achieves $\overline{Q} = 1$, but its content selection is extremely limited (Top-N must achieve Q_i^{max} for all i) and thus leads to a very poor cost. The Low Cost policy, biased only towards the cached content is heavily ignored by the "picky" user, which results to a jointly bad $(\overline{C}, \overline{Q})$ metric pair. The q-Mixed policy achieves a better tradeoff $(\overline{C}, \overline{Q})$ than the other myiopic policies, but remains heavily suboptimal.

Obs. #5: Isolating only the performance of the MDP-optimal policy, in all three datasets we can see that the higher the $\overline{L}$, the better the $\overline{C}$, which hints us a similar behavior to Fig. 5 and additionally, the $\overline{Q}$ is also improved. Similarly to



(a) MovieLens - $\overline{L} = 10$

(b) MovieLens - $\overline{L} = 25$

(c) Youtube - $\overline{L} = 10$

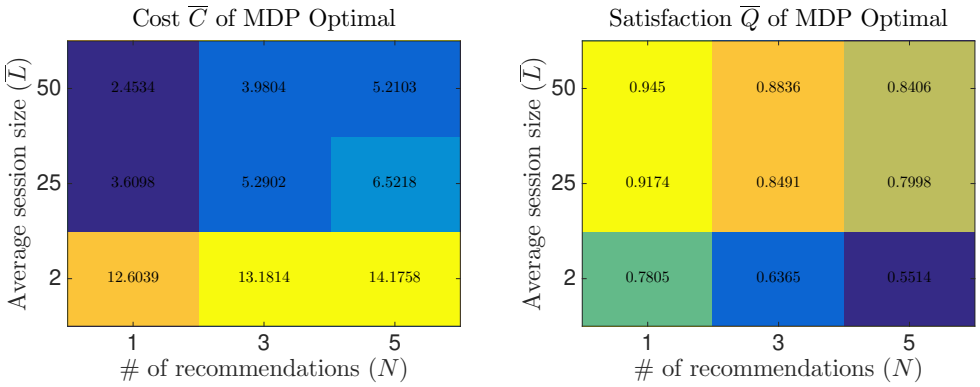
(d) Youtube - $\overline{L} = 25$

(e) Synthetic - $\overline{L} = 10$

(f) Synthetic - $\overline{L} = 25$

Fig. 8. Picky user: $\overline{C}$ - $\overline{Q}$ points of operation ($N = 2$)

earlier, in the Synthetic dataset, the myopic q-mixed can find values closer to the optimal compared to the real datasets,



(a) Values of Expected Cost per request

(b) Values of Expected user Satisfaction per request

Fig. 9. Picky user: Expected Cost and user Satisfaction per request vs: Num. of Recommendations (N) and mean Session Size $\overline{L}$ - Movielens

due to how dense it is. However, as a general observation throught all datasets: the more requests the user is doing, the more the MDP improves the cost and the user satisfaction simultaneously!

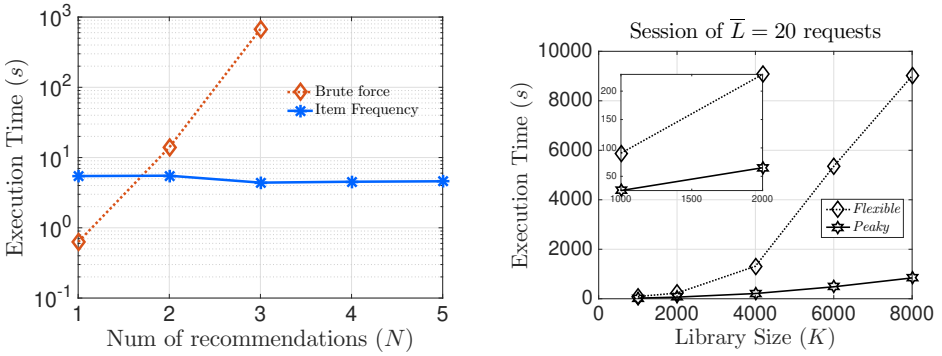
Effect of recommendation batch size (N). We investigate the joint effect of $\overline{L}$ and N in Fig. 9. An interesting observation that has to do with relating the two models, is that the "picky" user has much lower access costs in the regime $N = 1$ than the "flexible". Regardless of the session size, the Movielens dataset has very large D_{C_1}, and essentially, since the user does follow the relevant items, and does not behave randomly (i.e. click w.p α), the RS manages to keep the user recommend items from within the set C_1 quite easily.

5.5 Execution time

Up to this point, we investigated tradeoffs between different policies and different datasets. Yet a very 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. 10(a), we choose a catalog of $K = 150$, and fix some parameters for the "flexible" user, and solve the MDP using *our item-frequency* 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 2.4, we see that the increase of N is devastating for the batch-frequency MDP, whereas our item-frequency approach is insensitive to it.

Scaling up. In Fig. 10(b), we investigate the the execution times of the MDP algorithms for the "flexible" and the "picky" user. To this end, we select $\overline{L} = 20$, increase the content library size, run the algorithm, and report the time it took until completion. These results serve as evidence that this method can be used to tune recommendations of practical size and not just toy scenarios of some hundred contents. As stated in [36], even the library size of $K = 1000$ can be considered practical since it could refer to the 1000 most popular files of Netflix for example. The authors in [10] perform simulation with sizes $K = 1000$ and $K = 10000$, which is the same order as our experiments. The MDP for the "flexible" user takes



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

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

Fig. 10. MDP: Execution Times

Table 4. MDP(0.99) vs OPT (under [18])

	(Average Cost (units), Execution Time (s))	
Dataset/Method	MDP(0.99)	OPT
Movielens	(5.69, 86.4)	(5.54, 1065.7)
Youtube	(6.16, 148.4)	(6.10, 5164.6)

about 9000 seconds (or 2.5h) for a library of 8K and the corresponding instance for the "picky" user instance needed about 800 seconds. These runtimes could be significantly decreased in a powerful server with multiple cores as the policy iteration we have implemented runs on as many cores as the algorithm finds available. Evidently, the runtimes of the "flexible" user are an order of magnitude higher than the ones for the "picky" user case. This is reasonable as the minimizers of the former are the K-sized LPs (see OP-flexible user) whereas the ones of the latter are simply sorting operations (see OP-picky user), and reasonably they scale better with the increase of K. This hints that the algorithm for the family of users that resemble the "picky" user, could be applicable in much larger scenarios.

Optimality vs execution time tradeoff. Here we compare our algorithm with the solution in [18] where the authors optimize the per request average cost of an *infinite size* session. Such a user can be captured by the "flexible" user; their framework easily reduces to ours by setting $\lambda \rightarrow 1$, in practice we set $\lambda = 0.99$, thus using a slightly smaller horizon of planing, and assuming $\alpha_{ij} = \frac{\alpha}{N}$, i.e., uniform click towards recommended content. In [18], the authors formulate the average cost minimization as an LP of size K^2 and the optimal solution is found using the State-of-the-Art LP solver CPLEX. Their solution is constrained to obey stationarity which is (a): a very demanding set of constraints and (b): unrealistic as the size of a user session *is never* infinite in practice. In Table 4, 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 [18]. In that table, we will refer to our proposed policy $MDP(0.99)$ (due to the selected λ) and to the one of [18] as *OPT*. We fix a set of parameters $\{\alpha = 0.9, q = 0.9, N = 2\}_{movielens}$ and $\{\alpha = 0.7, q = 0.7, N = 2\}_{youtube}$ and measure for every dataset the cost and the runtime of the existing approaches, i.e., [18] and MDP(0.99). The results are shown in Table 4 and justify that the MDP approach is far superior from [18] in terms of execution time while resulting in almost the same value of average cost. The MDP cost per viewed item can be further reduced to coincide with the solution from

[18] by further increasing λ. The most impressive point of this experiment is the dramatic decrease in run time that the MDP achieves while not sacrificing much in terms of performance.

6 DISCUSSION

Applicability. Network-friendly recommendations may be viewed as a "hard-to-deploy" technology, mainly due to security and privacy issues raised by the use of *https* protocol, whose consequence is that different entities decide the caching (NOs) and the recommendation policies (CPs). However, recent developments in networking show otherwise. Netflix already operates its own CDN, called OpenConnect [7] since 2012, and recently, launched a new feature according to which, recommended content will be prefetched offline to subscribers phone devices [6]. Moreover, Google cooperates with NOs and has presence in the edge of the network through Google Global Cache [4], where it statically places (prefetches) popular multimedia files (> 100 MBs) from Google Play and Youtube. Consequently, security issues are thus bypassed, as one party controls both what to prefetch and what to recommend.

Learning. We employed the model-based optimization path of MDP and did so in a problem where the human factor is heavily involved. However, the MDP sets the stage for learning based techniques, e.g., q-learning, sarsa etc., where the RS is trained over massive user request datasets; an approach which requires minimal assumptions on the user and could thus generalize better than model-specific methods. Along these lines, we believe that the use of item frequencies (instead of batches) could still be of great importance in reducing the size of the action set .

Controlling the cache. An interesting path is to jointly consider the optimal (static) caching and recommendation problem for arbitrary mean size random sessions. A reasonable first effort could be: (a) solve our MDP for the recommendation variable, and then check the steady-state distribution of the contents induced by the recommendations, and (b) cache the top items based on the ranking (the steady state pmf), and then repeat until the caching does not change (convergence). This approach, however lacks theoretical guarantees, but we believe that its simplicity is appealing, especially for studying its theoretical properties. Another possible path of research, is to consider caching as an explicit variable in the MDP and optimize jointly for the long run optimal expected cost.

7 CONCLUSIONS

We have developed a promising MDP framework for optimal look-ahead network friendly recommendations that are able to exploit the structure of the content graph and discover non-obvious content recommendations. The framework is very flexible and can incorporate several possible user behaviours related to preferences and click-through. More importantly, by using item-frequency recommendations in the Bellman equations we have proposed algorithms that scale well both with the size of content library, as well as with the batch size. The complexity remains low because the related optimization problems have less unknowns, they are linear or at worse convex, and allow for parallelisation. Hence such algorithms are promising for application in real-world problems.

REFERENCES

[1] 2007. http://netsg.cs.sfu.ca/youtubedata/.
[2] 2015. Google spells out how YouTube is coming after TV. {http://www.businessinsider.fr/us/google-q2-earnings-call-youtube-vs-tv-2015-7/}.
[3] 2015. The average mobile YouTube session is now 40 minutes, Google says. https://www.cio.com/article/2949473.
[4] 2020. Google Peering. https://peering.google.com/#.
[5] 2020. *Multi-access Edge Computing (MEC); Device application interface.* Technical Report.
[6] 2021. Netflix launches 'Downloads for You,' a new feature that automatically downloads content you'll like. https://techcrunch.com/2021/02/22/netflix-launches-downloads-for-you-a-new-feature-that-automatically-downloads-content-youll-like.

[7] 2021. Netflix Open Connect. https://openconnect.netflix.com/.

[8] Abdulrahman Al-Dailami, Chang Ruan, Zhihong Bao, and Tao Zhang. 2019. QoS3: Secure Caching in HTTPS Based on Fine-Grained Trust Delegation. *Security and Communication Networks* (2019).

[9] Bartlomiej Blaszczyszyn and Anastasios Giovanidis. 2015. Optimal geographic caching in cellular networks. In *Proc. IEEE ICC*.

[10] Livia Elena Chatzieleftheriou, Merkouris Karaliopoulos, and Iordanis Koutsopoulos. 2019. Jointly optimizing content caching and recommendations in small cell networks. *IEEE Trans. on Mobile Computing* 18, 1 (2019), 125–138.

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

[12] Paul Covington, Jay Adams, and Emre Sargin. 2016. Deep neural networks for youtube recommendations. In *Proc. ACM RecSys*. 191–198.

[13] Trinh Viet Doan, Ljubica Pajevic, Vaibhav Bajpai, and Jorg Ott. 2018. Tracing the path to YouTube: A quantification of path lengths and latencies toward content caches. *IEEE Communications Magazine* 57, 1 (2018), 80–86.

[14] David A Farber, Richard E Greer, Andrew D Swart, and James A Balter. 2003. Internet content delivery network. US Patent 6,654,807.

[15] O. Fercoq, M. Akian, M. Bouhtou, and S. Gaubert. 2013. Ergodic Control and Polyhedral Approaches to PageRank Optimization. *IEEE Trans. on Automatic Control* 58, 1 (Jan 2013), 134–148.

[16] Benjamin Frank, Ingmar Poese, Yin Lin, Georgios Smaragdakis, Anja Feldmann, Bruce Maggs, Jannis Rake, Steve Uhlig, and Rick Weber. 2013. Pushing CDN-ISP collaboration to the limit. *ACM SIGCOMM Computer Communication Review* 43, 3 (2013), 34–44.

[17] Theodoros Giannakas, Pavlos Sermpezis, and Thrasyvoulos Spyropoulos. 2018. Show me the Cache: Optimizing Cache-Friendly Recommendations for Sequential Content Access. In *Proc. IEEE WoWMoM*.

[18] Theodoros Giannakas, Thrasyvoulos Spyropoulos, and Pavlos Sermpezis. 2019. The Order of Things: Position-Aware Network-friendly Recommendations in Long Viewing Sessions. In *Proc. IEEE/IFIP WiOpt*.

[19] Lazaros Gkatzikis, Vasilis Sourlas, Carlo Fischione, Iordanis Koutsopoulos, and György Dán. 2015. Clustered content replication for hierarchical content delivery networks. In *Proc. IEEE ICC*.

[20] Carlos A Gomez-Uribe and Neil Hunt. 2016. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems* 6, 4 (2016), 13.

[21] F Maxwell Harper and Joseph A Konstan. 2016. The movielens datasets: History and context. https://grouplens.org/datasets/movielens/. *ACM Transactions on Interactive Intelligent Systems (TiiS)* 5, 4 (2016), 19.

[22] Jonathan L Herlocker, Joseph A Konstan, Al Borchers, and John Riedl. 1999. An algorithmic framework for performing collaborative filtering. In *Proc. ACM SIGIR*. 230–237.

[23] Savvas Kastanakis, Pavlos Sermpezis, Vasileios Kotronis, and Xenofontas Dimitropoulos. 2018. CABaRet: Leveraging Recommendation Systems for Mobile Edge Caching. In *Proc. ACM SIGCOMM Workshops*.

[24] Yehuda Koren, Robert Bell, and Chris Volinsky. 2009. Matrix factorization techniques for recommender systems. *Computer* 42, 8 (2009).

[25] Dilip Kumar Krishnappa, Michael Zink, and Carsten Griwodz. 2013. What should you cache?: a global analysis on youtube related video caching. In *Proc. ACM NOSSDAV Workshop*. 31–36.

[26] Dilip Kumar Krishnappa, Michael Zink, Carsten Griwodz, and Pål Halvorsen. 2015. Cache-centric video recommendation: an approach to improve the efficiency of youtube caches. *ACM Transactions on Multimedia Computing, Communications, and Applications* 11, 4 (2015), 1–20.

[27] Jonatan Krolikowski, Anastasios Giovanidis, and Marco Di Renzo. 2018. Optimal cache leasing from a mobile network operator to a content provider. In *Proc. IEEE INFOCOM*. 2744–2752.

[28] Dong Liu and Chenyang Yang. 2018. A Learning-based Approach to Joint Content Caching and Recommendation at Base Stations. *arXiv preprint arXiv:1802.01414* (2018).

[29] Yuanhua Lv, Taesup Moon, Pranam Kolari, Zhaohui Zheng, Xuanhui Wang, and Yi Chang. 2011. Learning to Model Relatedness for News Recommendation. In *Proc. WWW*. 57–66.

[30] Diogo Munaro, Carla Delgado, and Daniel S Menasché. 2015. Content recommendation and service costs in swarming systems. In *Proc. IEEE ICC*.

[31] Hyunwoo Nam, Kyung-Hwa Kim, and Henning Schulzrinne. 2016. QoE matters more than QoS: Why people stop watching cat videos. In *Proc. IEEE INFOCOM*.

[32] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank citation ranking: Bringing order to the web.* Technical Report. Stanford InfoLab.

[33] Martin L Puterman. 2014. *Markov Decision Processes.: Discrete Stochastic Dynamic Programming*. John Wiley & Sons.

[34] Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. 2001. Item-based collaborative filtering recommendation algorithms. In *Proc. WWW*.

[35] Pavlos Sermpezis, Theodoros Giannakas, Thrasyvoulos Spyropoulos, and Luigi Vigneri. 2018. Soft Cache Hits: Improving Performance through Recommendation and Delivery of Related Content. *IEEE JSAC* (2018).

[36] Karthikeyan Shanmugam, Negin Golrezaei, Alexandros G. Dimakis, Andreas F. Molisch, and Giuseppe Caire. 2012. FemtoCaching: Wireless video content delivery through distributed caching helpers. In *Proc. IEEE INFOCOM*.

[37] Linqi Song and Christina Fragouli. 2018. Making recommendations bandwidth aware. *IEEE Trans. on Inform. Theory* 64, 11 (2018), 7031–7050.

[38] Larissa Spinelli and Mark Crovella. 2017. Closed-Loop Opinion Formation. In *ACM WebSci*. 73–82.

[39] Renjie Zhou, Samamon Khemmarat, and Lixin Gao. 2010. The Impact of YouTube Recommendation System on Video Views. In *Proc. ACM IMC*.