SORBONNE UNIVERSITÉ

EURECOM
Sophia Antipolis

Joint Modeling and Optimization of Caching and Recommendation Systems

Dissertation

submitted to

Sorbonne Université

in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Author:

Theodoros GIANNAKAS

Scheduled for defense on the 13th March, 2020, before a committee composed of:

Reviewers		
Dr.	**Chadi BARAKAT**	Inria, France
Prof.	**György DÁN**	KTH, Sweden
Examiners		
Prof.	**Gustavo De VECIANA**	UT Austin, USA
Prof.	**Petros ELIA**	EURECOM, France
Prof.	**Karin Anna HUMMEL**	JKU, Austria
Prof.	**George IOSIFIDIS**	Trinity College, Ireland
Director of Thesis		
Prof.	**Christian BONNET**	EURECOM, France
Co-Director of Thesis		
Prof.	**Thrasyvoulos SPYROPOULOS**	EURECOM, France

SORBONNE UNIVERSITÉ

EURECOM
Sophia Antipolis

Modélisation et optimisation conjointes des systémes de mise en cache et de recommandation

Thèse

soumise à

Sorbonne Université

pour l'obtention du Grade de Docteur

Auteur:

Theodoros GIANNAKAS

Soutenance de thèse effectuée le 13 Mars 2020 devant le jury composé de:

Rapporteurs		
Dr.	**Chadi BARAKAT**	Inria, France
Prof.	**György DÁN**	KTH, Sweden
Examinateurs		
Prof.	**Gustavo De VECIANA**	UT Austin, USA
Prof.	**Petros ELIA**	EURECOM, France
Prof.	**Karin Anna HUMMEL**	JKU, Austria
Prof.	**George IOSIFIDIS**	Trinity College, Ireland
Directeur de Thèse		
Prof.	**Christian BONNET**	EURECOM, France
Co-Directeur de Thèse		
Prof.	**Thrasyvoulos SPYROPOULOS**	EURECOM, France

To my parents and Penny

Abstract

Caching content closer to the users with the help of Content Distribution Networks (CDNs) was a groundbreaking idea that shaped today's high speed communication networks. Fast forward to 2020 and the wired setup has reached very high data rates in practice, however users are becoming increasingly more active in the wireless domain. Essentially, as the users experience better service in a wired environment, their demands for the wireless service increase as well.

To this end, it has been recently proposed by the networking research community to further capitalize on the idea of caching and extend the paradigm (of the well-established CDN) to the edge of the network. Potentially, this approach could create the much needed leap and eventually become the means through which the Mobile Network Operators (MNOs) meet the users needs. Caching presents the users and the MNOs with a clear "win-win" solution, as the first party will experience a considerably better performance, while the MNOs will be able to save cost as they will need to use less and less the backhaul infrastructure.

A crucial point that we are missing in the above thought process is that in order for the caching to be *effective*, the contents we store locally *must* be able to attract a lot of requests. However, that is not to happen due to (a): the large catalogues, (b): the diverse user preferences. Despite these two difficulties, we can take advantage of the fact that the internet is becoming more entertainment oriented and that is why we propose to bind recommendation systems (RS) and caching in order to boost the network performance.

In a nutshell, the three first works presented in the thesis deal with the challenging problem of *Network Friendly Recommendations (NFR) in Long Viewing Sessions*, whereas in the last section we focus on the Recommendation Aware Caching problem.

More specifically, in the first chapter we present the current literature status of the caching and recommendation interplay, and highlight the existing gaps. In Chapter 2, we define the problem of NFR in long sessions assuming a user who requests contents in a Markovian manner. We then proceed in formulating the NFR as an optimization problem by enforcing hard constraints on the user satisfaction and show that it is nonconvex. We conclude the chapter by presenting an ADMM heuristic algorithm to validate that indeed our approach heavily outperforms existing myopic alternatives.

In the third chapter, we solidify and further strengthen the result of the previous chapter. We transform the optimization problem of NFR on long viewing sessions to a Linear Program and explicitly state the necessary conditions needed for the equivalence. We then proceed to further generalize the problem by incorporating the user preference on

the recommendations according to how they are positioned on the application/webpage screen. We prove that a similar equivalent transformation can be applied to this more general problem, and thus can be treated as an LP as well.

In Chapter 4 we cast the the NFR as a Markov Decision Problem (MDP). The contribution of this approach is essentially twofold: (a): The MDP framework offers a fertile ground for more scalable and practical algorithmic solutions than our previous approaches. Interestingly, we were able to solve our original problem much faster (maintaining ϵ-optimality) than using the state-of-the-art CPLEX solver and (b): The MDP approach on the problem allows for some far more realistic user behavior modeling. To this end we present two novel models where the users are selecting the content proportionally to how satisfactory the recommendation policy is.

While the previous chapters focus on the recommendation, in the final chapter, we take a preliminary look on the caching side of the problem. We depart from the markovian content access model, and assume a IRM access model, for simplicity, on top of a femto-caching setup. To this end, we initially formulate the problem of users in a femtocache network that can accept or reject alternative (than the one requested) contents in much better streaming quality at their mobile device and the problem of delivering. Then we formulate a slightly more aggressive use case, where the MNO (due to incentives in the user's quota) can deliver other related content (locally stored) given extreme congestion condition on the network. We then show that both of these problems are NP-Hard and then rigorously prove they have submodular structure; a property which guarantees a bounded gap from the optimal for when we apply a greedy allocation of the contents.

Acknowledgements

I would first like to thank my advisor Akis who constantly pushed me towards better results, for challenging me and for trusting me with such an interesting problem to work with. Then, my two informal advisors, Drs. Anastasios Giovanidis and Pavlos Sermpezis whose guidance definitely shaped me throughout these years. In addition, I want to thank all my friends in Athens and in Nice who tolerated me all these years and for accepting me in their lives unconditionally. More specifically, the discussions we had in Ketje over multiple pints of beers with many of you, will always be something I will remember fondly. Moreover, I was lucky enough to grow up in an environment where academic excellence was considered something admirable that should be never be enforced (or obligatory). I would thus like to thank my parents not only for their love, but more importantly for supporting each and every decision I took in life. Finally, all this would never be possible if it were not for Penny, my partner in life. I believe her unlimited love and support throughout these 10 years is the main reason I wanted to make progress in life academically and more importantly become a better person.

Contents

List of Figures

List of Tables

Acronyms and Abbreviations

The acronyms and abbreviations used throughout the manuscript are specified in the following. They are presented here in their singular form, and their plural forms are constructed by adding and *s*, e.g. CDN (Content Delivery Network) and CDNS (Content Delivery Networks). The meaning of an acronym is also indicated the first time that it is used.

CDN	Content Delivery Network.
ISP	Internet Service Provider.
CP	Content Provider.
SC	Small Cell.
QoS	Quality of Service.
QoE	Quality of Experience
MB, PB, TB	Mega Byte, Peta Byte, Tera Byte.
RS	Recommendation System.
OP	Optimization Problem.
CHR	Cache Hit Ratio.
CMR	Cache Miss Ratio.
ADMM	Alternating Direction Method of Multipliers.
LP	Linear Program.
QP	Quadratic Program.
QCQP	Quadratically Constrained Quadratic Program.
DP	Dynamic Program.
RL	Reinforcement Learning.
IRM	Independent Reference Model.
MDP	Markov Decision Problem.
IHMDPD	Infinite Horizon Markov Decision Problem with Discounts.
ViT	Value Iteration.
PiT	Policy Iteration.
Eq.	Equation.
Def.	Definition.
i.i.d.	independent and identically distributed.
w.p	with probability.
NFR	Network Friendly Recommendations.
lhs	left hand side.

rhs	right hand side.
rv	random variable.
pmf	probability mass function.
w.l.o.g	without loss of generality.

Chapter 1

Introduction

1.1 The Interplay of Caching and Recommendation

1.1.1 The Role of Caching and Why it is not Enough

The current thesis focuses on laying the ground for the eventual joint coordination of caching and recommendation policies in wireless networks. Traditionally, Content Providers (CP) such as YouTube, Netflix have been trusting their content delivery to CDN operators such as Akamai, Google or Amazon Cloudfront in order to ensure fast, reliable and secure delivery to the end users (in the wired or wireless case). A CDN in principle reduces the cost of transferring data as it has the role of the *middleman* between the users and the web servers in terms of geographical location. At the start, users could only receive the requested content from a perhaps very isolated server somewhere across the globe. However, during "rush hours" or during some important events (elections, natural disasters or big sports events) loading a webpage would frequently lead to some crashed server which could no longer respond to user requests. This situation was originally called the problem of "hot spots" and was eventually resolved after the Internet structure changed for good and the CDN paradigm was deployed. The network topology supported by the CDN is depicted in Fig.(1.1).

Fast forward to 2020, and CDNs is an established technology that had a major impact on the Internet performance. The aforementioned improvements although groundbreaking, were able to resolve many issues in the Internet *of that era*. However that was an era before the emergence of streaming content websites such as YouTube or Netflix. What is more, users are becoming "hungrier" and harder to please as they can even feel discomforted when the content is delivered in poor streaming quality, i.e., high latency, start-up delay, playout interruptions stalls, low resolution etc. CDNs have managed to resolve many of these issues in the wired setting. However, it is needless to say that today's users are requesting for heavy content through their *mobile operators* which at the moment cannot perform equally well to the home/office connections. It becomes obvious that at the dawn of the 5G era, the existing system architecture is deemed as insufficient.

Hence, new architectures for cellular networks, comprising densification of the access network (small-cells, SCs) and integration of computing and caching capabilities in base

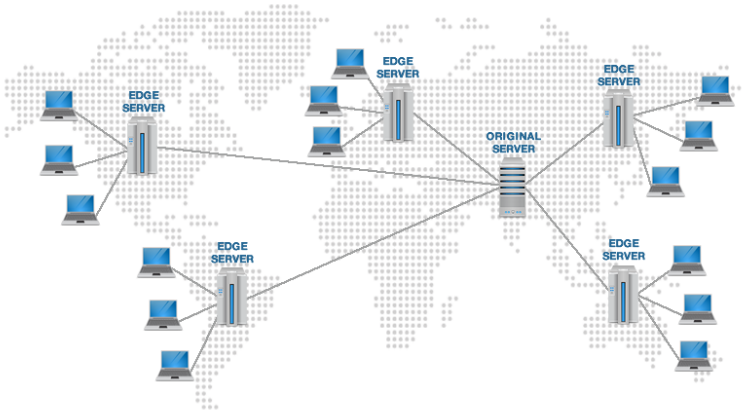

Figure 1.1 – Depiction of a CDN (Source: https://www.highcharts.com/blog/news/50-codehighchartscom-moves-to-cdn/)

stations (mobile edge caching and computing, MEC), have been proposed [1], to cope with the recent boom in traffic demand [2] and the envisioned increase in devices/traffic density in the near future ($\times$10k more traffic and $\times$ 10 to 100 more devices [3])

Thus, the core idea of the networking community is to shift the CDN paradigm to the wireless setting and cache content at the edge of the network, i.e., very close to the user. Although this idea seems very appealing, it is not a one-fits-all solution as there are several limitations to it. Fresh content such as news, music or TV series is produced on a daily basis, and user generated content (UGC) is also increasing by leaps and bounds. An important feature however of such content is that it is ephemeral, in the sense that it is demanded for some specific duration and eventually it fades over time. Therefore finding which content should be cached, translates to "predicting" its popularity for the foreseeable future.

On top of that, a practical constraint that needs to be taken into account is that caching at the edge is far more constrained by its nature than the CDN. The edge-caches are supposed to be placed in urban spots such as bus stops, buildings etc or even the mobile devices themselves and thus their physical size (and memory capacity obviously) is heavily affected by that. Impressively, it is predicted that the number of required storage points in future cellular networks will be orders of magnitude larger than in traditional CDNs [4] (e.g., 100s or 1000s of small cells (SCs) corresponding to an area covered by a single CDN server today). As a result, the storage space per local edge cache must be significantly smaller to keep costs reasonable. Note that even if we considered a small subset of the entire Internet catalogue, e.g., a typical torrent catalogue (1.5 PB) or the Netflix catalogue (3 PB), edge cache hit ratio would still be low even with a relatively skewed popularity distribution [5] and more than 1 TB of local storage [6, 7]. Therefore, a first important conclusion that we will build upon in the sequel of this work is the following:

"Due to the capacity constraints, caching content at the edge has practical limitations."

Currently, MNOs have to do peak-rate provisioning of their network, which is costly and often wasteful during non peak hours. Notice though that if most traffic could stay local, i.e., be satisfied by the local SC, then the MNO wins. That is mainly due to the fact that if caches manage to satisfy a large fraction of the content demand, the data center, core servers deeper in the network, etc will be alleviated. However, an important consequence/side-effect of keeping the traffic local is that the user will probably also win in terms of QoE, as he will be able to enjoy a much better streaming experience. It is thus evident that the networking is presented with a *clear win-win situation for both parties, MNOs and users* [8].

"Caching is a win-win situation for the ISPs and the users. It must be somehow exploited."

The above discussion suggests that caching policy alone is limited in the amount of performance gain it can bring at at an edge cache. Increasing cache capacity (to improve hit rates) or backhaul capacity (to allow for more frequent cache updates) seem like the only way to cope with this problem, but these are "hardware" solutions involving significant CAPEX/OPEX costs, when considering the very large number of small base stations envisioned in future heterogeneous and ultra-dense networks. The following question then arises

" Are there any practical 'software-based' solutions that can improve caching efficiency, at a low cost?"

1.1.2 Recommendation Driven Requests

Recommendation Systems (RSs) is a well established information filtering system [9] that nowadays is embedded in almost all entertainment platforms such as Pandora, Spotify, Netflix, YouTube as well as e-commerce websites such as Amazon or eBay. In its core, an RS aims to predict the user preferences. More specifically, the RS has at its disposal a set of items/objects (could be films, songs, goods etc.) and a set of users that have rated *some* of these items. Thus one can imagine these ratings as a table (or matrix if you will) which is *incomplete*. The RS objective is to fill in these missing ratings and its effectiveness is usually measured by how close the actual ratings will be compared to the predicted ones. Most fundamental and widely known techniques include collaborative filtering [10], matrix factorization, such as the paper that won the Netflix Prize [11] and more recently Deep Neural Networks [12].

In general, a RS *should* be able to help the user discover contents that would be of interest to him. What this suggests, is that if a RS is able to suggest interesting contents to the user, then a relationship of trust is built between the user and the RS. As today's catalogues are massive (irrespectively of the type of content), in a way recommendations

manage to shrink a catalogue of size K to an almost infinitely smaller catalogue of size N (the recommendation batch/list suggested to the user). The recommendation process, when successful, essentially can win the user several minutes (or more) when he is looking for a content or a good to purchase.

"The user requests are increasingly driven by the RS suggestions."

There is a variety of recent measurement studies that confirm the above statement such as [13], [14], [15]. What is more, in [16], we see that in the case of Netflix, a staggering 80% of its traffic comes from the recommendations, while the corresponding percentage for YouTube's related video is 50% [15]. Here we list some examples of everyday situations where users may feel happy or unhappy depending on the RS decisions.

- A user watching video lectures on YouTube *wants* to find part $i+1$ of the lecture on his recommendation list when he is currently viewing part i. (User-RS Trust $\uparrow$)
- A user who works while listening to ambient music may have left *Autoplay ON* in order to not get distracted by trying to find new pieces of music. That user will most likely get angry though, if while working all of a sudden listens to some death metal song. (User-RS Trust $\downarrow$)
- A user listening to some band, tries to remember a specific song title by that band, will feel extremely excited if he discovers that specific song at his recommendation list. (User-RS Trust $\uparrow$)
- The recommender suggesting a song or clip related to what the user is listening/watching *which the user didn't know at all*. The user eager to explore new content watches it and enjoys his new discovery. (User-RS Trust $\uparrow$)

It becomes obvious that there is a silent relationship of trust between the RS and the User. A trustworthy RS will manage to gain the user's trust and thus achieve a very high click-through rate. However, it is important that any algorithm that alters recommendations (e.g., to favor cached content) maintains this trust relation

1.1.3 Cache and Recommendation Co-Design

In an Internet that is becoming increasingly entertainment-oriented, our proposal is to connect these two seemingly unrelated entities in order to achieve the *win-win* event we mentioned above. Traditionally, the CPs such as Netflix or YouTube trusted their content delivery to the users on CDN operators (in the wired setup). However recently Netflix started operating its own CDN, known as Open Connect [17] and has started partnering with ISPs around the globe in order to use their resources and offer the highest possible streaming experience to its users. We envision a system such as Open Connect which extends to the wireless edge. Therefore we expect the CPs (such as Netflix in that case) to be responsible for this joint (caching and recommendation) system architecture. In this way we have

- No issues of privacy (which would be a problem if the MNO was involved in this)
- No issues of feasibility, i.e., no issues of HTTPS tunneling.

Of course one might ask "What is the incentive for the CP to reduce backhaul traffic?" and the clear answer to that is "it will be paying for it, as it will probably be renting it's own end-to-end network slice".

This real life example shows that multimedia giants such as Netflix already decide both *what content to cache where and what content to recommend to its subscribers*. Interestingly, measurements from [18] indicate that the user engagement (willingness to click on recommended items) is strongly correlated with the Quality of Experience (QoE) the user is receiving. However, this can be interpreted differently. It hints that in order to develop a relationship of trust between the user and the RS, the RS will need to consider also the network state when deciding in the recommendation list.

However, we have not specified what we mean by *co-design*. The cooperation of the two entities allows freedom for brainstorming and invites the research community to explore wild ideas. We distinguish the wide Caching and Recommendation Problem into three distinct categories which we specify below.

- **Cache-Aware Recommendations:** In that problem, we do not focus on what contents to place where inside the network. Our attention is on *how to increase* the request rate of the already cached contents through the recommendation mechanism. More generally, given some state of the network (routing costs, location of the content, delivery, duration etc), the goal is to come up with recommendation policies that minimize the access cost. An obvious tradeoff of this problem is that the RS objective is to balance the access cost while maintaining a trustworthy relationship with the user.
- **Recommendation-Aware Caching:** In this discipline, the recommendation aspect of the problem is considered a known and given quantity. The caching decisions should be cautiously chosen after a thorough investigation of the RS policy. As an example, if some content i is less popular than some content j, but i is of the same category with many other popular files, it perhaps should be favored over j, as it could achieve better hit rate in the long run.
- **Joint Design:** In this approach, content placement and content recommendation are decided jointly. It is the final and a very challenging variation of the problem which is based on the following observation: "recommendations shape the popularity, Caching tries to exploit popularity, but Recommendations are designed to help caching". Abstractly, the two entities form a very interesting circle that can potentially lead to significant overall performance.

The main goal of the current thesis is to deal mostly with the first problem, i.e., the Network Friendly Recommendations (NFR). One of the main challenges found in this work, is that we do not assume the Independent Reference Model (IRM) for the user requests. We depart from this assumption and we aim to model and optimize the

recommendation policies under the *sequential content consumption regime*. At the last part of this thesis, we will present an interesting use case of the Recommendation-Aware Caching Problem.

1.2 Related Work

1.2.1 Mobile Edge Caching.

Deploying small cells (SCs) over the existing macro-cell networks infrastructure, has been extensively studied and is considered a promising solution that could handle the existing and predicted massive data demands [19, 20, 21]. However, this densification of the cellular network will undoubtedly impose heavier load to the backhaul network. Taking advantage of the skewness in traffic demand, it has been suggested that caching popular content at the "edge" of the network, at SCs [22], user devices [23, 24, 25], or vehicles [26, 27] can significantly relieve the backhaul. The work in [6], focuses on learning time-varying popularities at wireless access caching and the authors propose an architecture which combines global learning and local caches with small population in order to improve the latency of accessing content. In [28], the authors developed a generic time-varying setup where a caching agent makes sequential fetch-cache decisions based on dynamic prices and user requests. They cast the problem as a Dynamic Program (DP), and also solve its online version using Reinforcement Learning (RL) techniques.

Moreover, in [29], the problem of caching is cast in the framework of online optimization. Then for the case where the request model is unknown, the authors derive dynamic minimum regret caching policies, which minimize the losses with respect to the best static policies in hindsight. Regarding the MNOs and CPs cooperation, an analytical business model was proposed in [30] where the CPs lease cache memory to MNOs in order to place their content; importantly the authors after investigating the possible policies that can be followed, they conclude that the cooperation of the two parties can be rewarding for both. This study strengthens our case, as CPs, e.g., YouTube, once able to have their own cache memory at the edge, they would be able to control both what to cache and what to recommend.

However, our work proposes a complementary approach for increasing the caching efficiency. We modify the recommendation algorithm to steer the users towards the cached content, when this is possible and satisfies the quality of user experience. This can bring further gains in cache hit ratio, on top of existing caching algorithms/architectures.

1.2.2 Caching and Recommendations

The interplay between recommendation systems and caching has been only recently considered in the literature, e.g., for peer-to-peer networks [31], CDNs [32, 33], or mobile/cellular networks [34, 35, 36, 37]. The works in [33, 35, 36, 34] consider the promotion/recommendation of contents towards maximizing the probability of hitting a local cache.

Leveraging the high influence of YouTube recommendations to users, the authors

of [33] propose a reordering method for the *related list* of videos and despite its simplicity, this method was shown to improve the efficiency of CDNs. [35] considers the joint problem of caching and recommendations, and proposes a heuristic algorithm that initially places contents in a cache (based on content relations) and then recommends contents to users (based on cached contents). At the selection of the recommendations, [35] considers a single request per user, whereas our work considers a sequential content consumption model, which is closer to user behavior in services such as YouTube, Netflix, Spotify, etc.

Similarly to [35], in [36], a single access user is considered. The caching policy in [36] is based on machine learning techniques, the users' *behavior* is estimated through the users' interaction with the recommendations and this knowledge is being exploited for the next SC cache updates.

Moreover, [34] studies the problem of recommendation-aware caching. Assuming a content provider/service that is able to offer an alternative content (at any given request), [34] proposes near-optimal approximation algorithms for content placement in mobile networks with single-cell and multi-cell (e.g., similarly to [22]) for such scenarios. In [38], the authors consider the joint problem by taking into account the user position preferences for the recommendation list; they then decompose the problem in the two main variables (that is caching and recommendations), and develop an algorithm to find the caching and recommendation policies in an alternating fashion.

In [39], the authors consider the scenario where the users are explicitly informed by the SC on whether the requested file is currently cached or not. They regard this interaction as an implicit recommendation but in the sense "high streaming quality content recommendation". The content request probabilities are assumed to be unknown, so the authors resort to Q-Learning techniques in order to learn the request distribution and after having that, they then optimize the cache policy. Another interesting dimension of the problem is introduced in [40]: the social network. Most CPs applications have currently integrated some sort of social network between the users and according to the proposed model, the content demand is shaped by popularity, recommendations *and the social diffusion.*

In addition, according to the work presented in [41], the RS should take into account that they have to serve the users under certain bandwidth constraints. The authors' objective is to gain in terms of bandwidth and user satisfaction; to this end they formulated optimization problems which prove to be hard and thus they opt for lower complexity greedy algorithms that come with performance guarantees. In [42], the authors present a first of its kind formulation of recommendations in the wireless setup as a contextual bandit problem, which they call contextual broadcast bandit. In doing so, they propose an epoch-based algorithm for its solution and show the regret bound of their algorithm. Interestingly, they conclude that the user preferences learning speed is proportional to the square of available bandwidth.

Finally, a very recent work done in [43] resembles to ours in the sense that the authors design a recommeder which aims to maximize the user engagement, i.e., keep the user in the system as much as possible, in the regime of long sessions. They start by defining the RS actions as the recommendation batches and then proceed to a decomposition of the frequency of the batches to object frequencies, which is very similar to the interpretation

we use throughout this thesis.

1.3 Contributions and Thesis Outline

In this section we present an outline of the thesis, along with the list of the main contributions divided by chapter. Throughout the thesis, we mostly focus on the NFR problem for long user sessions. To this end, Chapters 2, 3, 4 are dedicated to this topic. In these chapters, we modeled realistic user request patterns and formulated optimization problems that ultimately aim to maximize the hit rate of the local cache under specific system design constraints. Finally in Chapter 5, we depart from the NFR and focus on the problem of maximizing the SCs' total expected hit rate on a femto cache network. There, we assumed that the users are willing to accept the most relevant content in the case where their requested content is not found in the cache. More specifically and by chapter we have.

Chapter 2:

1.3.1 Limitations of Existing Works

The topic, although quite new, already had a few publications. In the existing works connecting connecting caching and recommendations, there were no prior studies that considered the sequential nature of the users request process. Essentially when users log in to an application such as YouTube or Spotify, they do not just request for one content, but rather a sequence of contents. Apart from the consideration, there were also no studies where the cost optimization was formally presented as an optimization problem under the dependent requests regime.

1.3.2 Novelties

1. We propose a model for stochastic, recommendation-driven sequential user requests, that better fits real users behavior in a number of popular applications (e.g. YouTube, Vimeo, personalized radio).

2. We then formulate the optimization problem of maximizing the cache hit rate performance, while explicitly constraining the average quality of recommendations we offer to the user.

3. We use a customized ADMM algorithm on the nonconvex problem to get a suboptimal solution.

1.3.3 Technical Summary

More specifically, we model the contents as states in a Markov Chain and assume that the user transition probabilities can be affected by the recommendation variables. The recommendation variable in our case is the probability with which a content j appears on the recommendation screen after viewing content i. Note that the contents are

underlyingly connected through a relations graph which expresses which contents are similar to what contents. According to our model, the user can either click uniformly one of the contents in his recommendation list, or do a random jump to any of the contents in the library. Each content is essentially associated with some access cost. Thus the fundamental tradeoff is caused by the potentially very similar but high access cost (or vice versa) of two contents. In particular, as we are interested in long user sessions, we approximate the very long user session by an infinite length (in terms of consumed contents) session, more specifically we want to maximize the long term percentage of time the user spends in the subset of cached contents. The optimization problem is shown to have a nonconvex objective function, while having a feasible set of solutions which is convex, thus resulting to an overall nonconvex problem. After a variable manipulation, we remove the nonconvexity from the objective and place it on the constraints set; we do so by introducing a set of new variables and the equal number of quadratic equalities (which are of course nonconvex). Importantly, what this buys us is a nice formulation which fits the widely used Alternating Direction Method of Multipliers (ADMM) framework.

In its standard form (and in ours), ADMM performs two exact minimization steps before doing a dual ascent step over the Lagrange Multipliers. We give a full algorithm where the inner minimization steps were implemented through projected gradient methods. Importantly though, we have to highlight that since the outer ADMM algorithm is performed over a general noncovex equality constraint and not on a linear one, this method comes with no theoretical performance guarantees. We perform a thorough data analysis and use three real life datasets over which the proposed algorithm is tested. Essentially, in this study our main objective is to measure how well our method fares against low-complexity but myopic alternatives. The simulation results show that our proposed method heavily outperforms a myopic approach, which by default is unable to capture the lengthy nature of the user request pattern. Finally we show some results regarding the improvement of our algorithms in terms of runtime when we used the customized first order methods instead of the CVXPY solver, which of course is a generic convex solver. The results of this chapter can be found in [44] and [45].

Chapter 3:

1.3.4 Limitations of Existing Works

Our previous chapter had two major drawbacks. For one, the algorithm we suggested was essentially a heuristic solution with no optimality guarantees. Then secondly, we assumed that our user when presented with a set of N contents, he clicks uniformly to any one of them. That is obviously an unrealistic assumption as most users essentially seem express some preferences on recommendations that appear for example in the top positions of recommendation list etc.

1.3.5 Novelties

1. Most importantly, in that chapter we present a change of variables through which we managed to transform our initial nonconvex problem to an LP. In doing so, we

can now guarantee that the performance of our solution is the globally optimal.

2. We formulated the NFR of long sessions where the user is clicking on the recommendations based on which positions they are placed on the screen. While this is a generalization of our previous model, the LP transformation works here as well.

1.3.6 Technical Summary

We associate each position i of the recommendation list to some probability v_i to be selected by the user. What is more, we assume again that user requests have the markovian property (memory of size one). To this end we formulate the objective of maximizing the hit rate in a long session where the RS has some statistics over the user preferences as an Absorbing Markov Chain.

Before attempting to solve that newly established problem, we make a small rewind and go back to the basic nonconvex optimization problem we formulated at the Chapter 2 and show the technical steps needed in order to transform it to an LP and guarantee the much desired optimality of the solution.

We then proceed in formulating the optimization problem that maximizes the cache hit rate (by assigning the appropriate cache costs to all contents) for long viewing sessions in the case where the user is clicking on the recommendation with some preferences according to their position. We now explicitly constrain not just the average quality the user is viewing, but the quality per position, i.e., we weigh positions that are more likely to be clicked as "more responsible" to suggest highly related contents compared to the less likely ones. Importantly, in that problem we can no longer optimize over some $K \times K$ matrix, but rather on an N-dimensional $K \times K$ matrix.

Like the previous Chapter, we decided to test our results over some real life datasets. However, in this work, our main focus departed from the performance gains of the proposed method compared to greedy/myopic policies, since the result of our LP problem is guaranteed to be the optimal. As a consequence, our primary goal was to investigate the gains of the *position-preference-aware* scheme with the *agnostic* one when both consider long sessions. We compared the two in terms of (1): the randomness ($H_{\mathbf{v}}$) and (2): the size of the recommendation batch (N). Essentially, when the CP has some statistics regarding the position preference of the users, placing the contents randomly could prove to be detrimental in some cases. In practice, the random placement can probably lead to some *good/useful* recommendations to "go to waste" as they could be in positions the user ignores.

Chapter 4:

1.3.7 Limitations of Existing Works

In the literature of caching and recommendation co-design (and in our previous works), none of the studies attempted to model the user clickthrough probability, i.e., his willingness to click on recommended content, as a function of how good (to be determined) the policy is. Secondly, although the LP formulation can be solved using very good

optimization toolboxes (such as CPLEX), the variable size is proportional to the square of the library size. Essentially if we deal with a catalog of 4K contents, we need to compute 1.6M variables, and even these very good solvers start becoming quite slow.

1.3.8 Novelties

1. Casting the NFR of long sessions as an MDP and the use of Dynamic Programming (DP), essentially breaks down the problem in many easier subproblems. This gave us the flexibility to explore more realistic versions of the problem where the user could evaluate if the recommender offered good sequences of contents or not.

2. The algorithmic structure of the DP solution, gave a fertile ground and revealed some "weaknesses" of the problem that we could capitalize. The DP gave us easier subproblems, where we could easily spot what tricks would cause a significant speed-up. In doing so, we were able to decrease the runtime of the same problem by a dramatic amount and reach practical problem sizes in reasonable runtimes.

1.3.9 Technical Summary

In this third and final pursue of the NFR problem, we investigated the topic under the perspective of Markov Decision Processes (MDP). To this end, we initiated our study by formulating the NFR as an Infinite Horizon MDP with discounts (IHMDPD), where in our setting the role of discount is essentially played by the average length over the user session (measured in contents). More specifically, the user session length L is modeled as a Geometric random variable (rv) of mean $\bar{L} = \frac{1}{1-\lambda}$. In Chapter 2, in order to approximate the long user session we assumed that the "very long" is "infinitely long", whereas the casting of the problem as IHMDPD relaxes that hard "infinite" assumption and allows us to solve the problem for some generic user statistics λ. We formulate a more general version of the NFR that aims in the cumulative network cost minimization in sessions of random length. We view the control variables of the problem using two interpretations; first one is the actions which is essentially the N-tuples of recommendations the RS is suggesting the user, and interestingly, the second one is the content frequency of appearance, which coincides with our formulation of the previous chapters.

Moreover, the MDP offers a framework which allows us to capture a variety of user behaviors through different stochastic models. In contrast to the convex optimization approach we took on the previous two chapters, the Bellman Equations of Optimality, essentially *breaks down* the initial problem into a series of many but much easier subproblems (Dynamic Programming approach). Therefore, this framework allowed us to experiment with different and importantly more realistic user behaviors. As opposed to the previous chapters, we removed the hard constraint from the constraint set and instead of assuming some *fixed* click-through rate on the recommendations, we modeled this quantity as a function of *how high the quality of the recommendations is*. Furthermore, the MDP formulation offers two important advantages: (1): it is not an off the shelf commercial optimization solver, thus the programmer has full knowledge of the algorithm implementation details, be it Value Iteration (ViT) or Policy Iteration (PiT), which means

that the data structures used can be customized according to the problem needs in an easier way, (2): it offers ϵ-optimality guarantees, which is in theory very important, but in practice can mean even more, as *tuning* how optimal you want your solution to be, allows more flexibility in terms of runtime, and finally (3): the Bellman Equations, reveal the structural properties of the optimal solution; understanding and using these properties could ultimately lead to extremely fast heuristic algorithms which are near-optimal.

In the results section of this chapter, we focus on two classes of results. A main contribution is the incorporation of the *quality* of the policy to the click-through rate. Essentially, when removing the recommendation quality constraint, we allow *the dataset to decide how much quality to offer for each content.* Importantly, we modeled the same user behavior as in Chapter 2 and observed that the algorithms of the MDP toolbox heavily outperform customized ADMM and our generic LP formulation solution implemented in CPLEX in terms of runtime. Results of this work can be found in the soon to be submitted work in [46].

Chapter 5:

1.3.10 Limitations of Existing Works

In the literature, the caching decisions (what content to place where) used to be only an allocation problem that mostly depended on the known (or estimated) statistics for the content probability masses. However as the user requests are heavily affected by the recommendation system suggestions (especially in the case of multimedia content), the existing designs lose the opportunity to take advantage of this fact and further improve the network performance by increasing the local caches hit rates.

1.3.11 Novelties

1. We introduce the novel concept/metric which we call Soft Cache Hits (SCH). According to that approach, the user may request a content, but could be equally happy to receive different contents that are of very similar characteristics. We propose two use cases that are quite practical and could very well be real life examples.

2. For the two use cases we model, we analyze the corresponding (discrete) optimization problem and show that it has submodular objective with matroid constraints in which case a simple greedy algorithm comes with a maximum guaranteed suboptimality gap. The latter suggests that we can significantly increase the problem size to practical ranges and compute our solution in reasonable time while being controllably suboptimal.

1.3.12 Technical Summary

We introduce the Soft Cache Hits (SCH) and consider a very different perspective from the previous chapters. Our aim is to model and optimize a recommendation-aware caching policy. To this end, we base our work on the seminal paper [22] and we further enhance

it with extra capabilities. More specifically, we consider the case where the user requests some content from the library $\mathcal{K}$ and if the content is not stored locally, a cache aware plugin notifies the user about the forthcoming poor quality of the requested video and suggests the user with a list of *alternative but related contents* that can be streamed in High Definition (HD). The user is able to accept some content from the recommendation batch or reject all of it. The latter use case is modeled in the single cache and the femtocache framework. Furthermore, we consider a more aggressive case (in a femtocache network), where the user is asking for a content and the MNO has some agreement with the user (e.g., a low cost quota) and is able to deliver him the most related content that is currently locally available without being given a permission by the user.

We model both these problems as a *caching allocation problem* under the umbrella of discrete optimization, and to this end we rigorously show that both of them are NP-Hard. For that reason, it is vital to resort to suboptimal heuristic algorithms. Thus subsequently, we prove that all the problem objectives are submodular, monotone, and have a matroid constraint, thus ensuring that a greedy $O(K^2M^2)$ placement has a guaranteed suboptimality of $(1 - \frac{1}{e})OPT$.

Chapter 2

The Long Session Problem

2.1 Introduction

The State-of-the-Art in the interplay of caching and recommendations up to that point only considered the case IRM traffic. That is users simply generated traffic (requests) based on some fixed pmf over the content library. Studying the problem under this assumption is essentially like completely ignoring the effects of the sequential content requests and the dependencies between them. In practice, users typically consume more than contents in sequence. In the case of YouTube, a user may choose a topic such as "How to?" which might be related to daily hacks, cooking, music etc and then watch (or just listen) many more videos from that category. Then it is likely the user feels a bit bored, and chooses a new topic.

We depart from this unrealistic assumption and consider a Markovian user who requests multiple items out of a finite library of size. Our objective is to fine-tune the recommender system in order to balance the tradeoff between the following two objectives in a *multiple items session*.

- Nudge the user towards low-cost content.
- Do so while maintaining the user satisfaction in high levels.

More specifically, when a user is currently viewing some content i, that content has some other files $\in \mathcal{K}$ which are related to it in some extent or totally irrelevant. In general,

1. **Global feature**, contents are either cached or uncached, this has to do with the network state, and
2. **Local feature**, conditioned on the content, contents are either related to some video i or unrelated.

According to our assumptions, if we suggest the user an unrelated content, this cannot contribute to the user satisfaction. Essentially, when we are at content i and need to come up with content suggestions, the catalogue $\mathcal{K}$ can be roughly split into four categories, i.e., contents that are

1. Cached AND Related.
2. Cached AND Unrelated.
3. Uncached AND Related.
4. Uncached AND Unrelated.

This is an extreme case which will allow us to demonstrate the necessity of policies with vision that consider the subsequent content requests and not just the next one. It becomes evident that for a *myopic approach*, the last group of contents is *totally useless* as it can neither contribute to cost reduction nor to user satisfaction. However, if one considers the case where the user has some fixed click-through on the recommended items and she is watching 4, 5 or more videos in sequence, it is quite likely that maybe some item from Group 4 can lead the user to a content where all its neighbors belong to Group 1. This myopically is a lossy event, but in the long run it is much more profitable (for the user and the network). In the next section, we will present a concrete example where recommending contents from Group 4 is indeed optimal for the multi-content case.

2.2 Problem Definition

Content Traffic. We consider a content catalogue $\mathcal{K}$ of cardinality K, corresponding to a specific application (e.g. YouTube). A user can request a content from this catalogue either by asking *directly* for the specific content (e.g., in a search bar) or by following a *recommendation* of the provider. In practice, users spend on average a long time using such applications, e.g., viewing several related videos (e.g., 40 min. at YouTube [47]), or listening to personalized radio while travelling.

Recommendation System. Recommendation systems have been a prolific area of research in the past years, and often combine content features, user preferences, and context with one or more sophisticated methods to predict user-item scores, such as collaborative filtering [10], matrix factorization [11], deep neural networks [12], etc. We will assume for simplicity that the *baseline* recommender system (RS) for applications where the user consumes multiple contents works as follows:

(i) The RS calculates a similarity score u_{ij} between every content $i, j \in \mathcal{K}$, based on some state-of-the-art method; this defines a similarity matrix $U \in \mathbb{R}^{K \times K}$. Without loss of generality, let $u_{ij} \in [0, 1]$, where we normalize values so that $u_{ij} = 0$ denotes unrelated contents and $u_{ij} \to 1$ "very related contents". W.l.o.g. we set $u_{ii} = 0, \forall i \in \mathcal{K}$ for all contents. Note also that this U might differ per user.

(ii) After a user has just watched content i, the RS recommends the N contents with the highest u_{ij} value [15, 12]. N is usually a small number (e.g. values of $3 - 5$ are typical for the default YouTube mobile app) or sometimes $N = 1$, as in the case of personalized radio (Spotify, last.fm) or "AutoPlay" feature in YouTube where the next content is simply sent to the user automatically by the recommender.

Caching Cost. We assume that fetching content i is associated with a cost $c_i \in \mathbb{R}$, which is known to the content provider. This cost might correspond to the delay experienced

by the user, the added load in the backhaul network, or even monetary cost (e.g. for an Over-The-Top content provider leasing the infrastructure). It can also be used to capture different caching topologies. For example, to simply maximize the cache hit rate, we could set $c_i = 0$ for cached content, and $c_i = 1$ for non-cached. For hierarchical caching [48, 4], the cost increases if the content is cached deeper inside the network. Since we work on the static setup, we assume that the cost of the contents remains constant regardless of recommendation policy.

Finally, as mentioned earlier, the specific wireless setup is relatively orthogonal to our approach and beyond the scope of this work. However, as a simple example, consider the well-known femto-caching setup [22]. The proposed algorithm there would first decide what will be cached at each base station. Then, x_i would have a low value for all content that the user in question can fetch from some BS in range (possibly dependent on the SINR of the BS, as well [22]), and a high value otherwise.

User Request Model. Based on the above setup, we assume the following content request model.

Definition 1 (User Request Model)**.** *After a user has consumed a content i, then*

- *(recommended request) with probability α the user picks one of the N recommended contents with equal probability $\frac{1}{N}$.*
- *(direct request) with probability $1-\alpha$ it ignores the recommender, and picks any content j from the catalogue with probability p_j, where $p_j \in [0,1]$ and $\sum_{j=1}^{K} p_j = 1$.*

p_j above represents an underlying (long-term) popularity of content j, over the entire content catalogue. For short, we denote the vector $\mathbf{p}_0 = [p_1, \ldots, p_K]^T$. Note that the above model can easily generalized to consider different probabilities to follow different recommended contents (e.g. based on their ranking on the recommended list). Note also the assumption that a is fixed: for instance, for applications where the user cannot evaluate the content quality before she actually consumes the content, this assumption is realistic, at least in the "short term". In the remainder of the paper, we assume that if the recommendation quality is above a threshold, then the user's trust in the recommender (i.e. the value of α) remains fixed. We plan to explore scenarios where α changes at every step, as a function of recommendation quality, in future work.

Recommendation Control. Our goal is to modify the user's choices through the "recommended request" part above, by appropriately selecting the N recommended items. Specifically, let an indicator variable r_{ij} denote whether content j is in the list of N recommended contents, after the user has watched content i. If $r_{ij} \in \{0,1\}$, the problem would be combinatorial. We can relax this assumption by letting $r_{ij} \in [0,1]$, and $\sum_j z_{ij} = N, \forall i$, r_{ij} can be interpreted now as a probability. For example, if $r_{13} = 0.5$, then content 3 will be recommended half the times after the user consumes content 1. Now, if we assume that the user is Markovian, i.e., she clicks on some content *only based on the item she currently views* and further assume that the user clicks uniformly among the N suggested items the transition probability between contents $i \to j$ can be written as

$$Pi \to j = \alpha \cdot \frac{r_{ij}}{N} + (1 - \alpha) \cdot p_j, \tag{2.1}$$

Putting all the transition probabilities together forms a stochastic matrix as

$$\mathbf{P} = \alpha \cdot \frac{\mathbf{R}}{N} + (1 - \alpha) \cdot \mathbf{P}_0, \tag{2.2}$$

where $\mathbf{P}_0 = \mathbf{1} \cdot \mathbf{p}_0^T$ is a rank-1 matrix ($P_0 \in \mathbb{R}^{K \times K}$), equal to the *outer product* of a vector with K unit values and the direct request vector $\mathbf{p_0}$. The above model of content requests, and the corresponding Markov Chain, is reminiscent of the well-known "PageRank model" [49], where a web surfer either visits an arbitrary webpage i (with a probability p_i) or is directed to a webpage j through a link from a webpage i (with probability p_{ij}).

Table 5.1 summarizes some important notation.

Table 2.1 – Important Notation

$\mathcal{K}$	Content catalogue (of cardinality K)
u_{ij}	Similarity score for content pair $\{i, j\}$
N	Number of recommended contents after a viewing
c_i	Cost for fetching content i
α	Prob. the user requests a recommended content
p_j	Average a priori popularity of content j
$\boldsymbol{p_0}$	A priori popularity distribution of contents, $\in \mathbb{R}^K$
r_{ij}	Prob. the RS recommends content i after viewing j
$\boldsymbol{\pi}$	Stationary distribution of contents, $\in \mathbb{R}^K$
$\mathcal{C}$	Set of cached content (of cardinality C)

Scenarios Captured by the Setup The above setup can represent the following wireless use case. The user communicates directly with some local caches where some popular content is currently stored and a central server which includes everything all the library $\mathcal{K}$. Let us focus on user 2 for a moment, and let us assume that based on past statistics he MNO knows that the user 2 during the next day he will be connected to two local caches (as he might be living for example in a place where both caches have can transmit data to him). Suppose now we have library of $K = 5$ files and the MNO has decided to cache contents $\#1, \#2$ at the cache 1, and $\#1, \#3$ to the cache 2 for the forthcoming day. From the user point of view and depending on proximity, the contents have the following costs $\mathbf{c} = [\min\{c_1^1, c_1^2\}, c_2^1, c_3^2, c_4^S, c_5^S]$; the former reads as: "content $\#1$ has the minimum cost depending on the cache delivery cost, content $\#2$ has the cost of local cache 1, $\#3$ has the cost of local cache 2, and contents $\#4, \#5$ the cost from the central server". Thus $\mathbf{c}$ encodes the *network state* for user 2. Having said that, the CP then modifies its recommendation policy for user 2 (and respectively for the other users) as

- Massively improve the streaming experience of its users

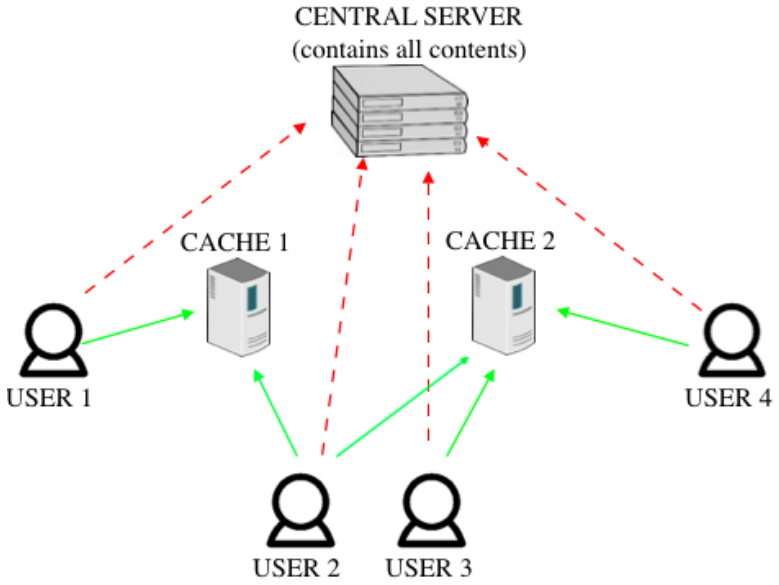

Figure 2.1 – Depiction of the wireless setup

- Might even be incentivized to this modification after agreements with MNOs.

2.3 Optimal vs Myopic: An Example

To motivate why it is important to look for policies with vision we will introduce a very simple (and temporary) problem setup which will show the obvious benefits of why it is important to solve a harder problem than the low-complexity myopic solutions.

Simple Myopic Policies. Here we list some practical and quite intuitive policies. These policies can either favor low network cost contents and/or contents that are useful for the user satisfaction *only in the next request.*

1. **Top-N (π_1):** Suggest the N files that are most related to i. In the case of ties for the values u_{ij}, recommend the lowest cost. Favors: User satisfaction.
2. **LowestCost-N (π_2):** Suggest the N least cost contents. In the case of ties for the cost c_j, recommend the highest u_{ij}. Favors: Low cost contents.
3. **δ-mixed (π_3):** Assign δ % of your budget to the least cost items and the remaining to the most related. If any of the least cost items was in the set of most similar, then simply assign the remaining budget to least cost. Favors: Both.

Remark 1. *Myopic policies that favor both low cost and user satisfaction can be expressed in terms of δ-mixed. Essentially, δ acts like a knob, for $\delta \to 1$, the policy is LowestCost-N, while for $\delta \to 0$, the policy becomes Top-N.*

$$\mathbf{U} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \end{pmatrix} \tag{2.3}$$

For simplicity, assume that the user only clicks on contents from the recommendation batch. We will infer the policies in terms of the r_{ij} values, and then compute the average quality per content i and the average long term cost they achieve. For that reason $\bar{C} = \sum_{i=1}^{4} \pi_i c_i = (1 - \pi_1) \cdot C$ (due to the cost assignment on the contents) and $\bar{Q}_i = \sum_{i=1}^{4} r_{ij} \cdot u_{ij}$, where π_i expresses the long term percentage of time the user spends in content i. Suppose the case where the RS is constrained to maintain $\bar{Q}_i \geq q$ for $i = 1, 2, 3, 4$.

Evaluate Top-N. We have $\boldsymbol{\pi}_1 = \mathbf{U}$ and hence the average cost of policy following $\boldsymbol{\pi}_1$ is $\bar{C}_{\boldsymbol{\pi}_1} = 0.75 \cdot C$, since $\pi_1 = 0.25$ and $\bar{Q}_i = 1.0 \; \forall \; i$.

Evaluate LowestCost-N. The policy is as follows

$$\boldsymbol{\pi}_2 = \begin{pmatrix} 0 & 1 - 2\epsilon & \epsilon & \epsilon \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{pmatrix} \tag{2.4}$$

In order to ensure that the user is not trapped on states #1 and #2, we split an ϵ probability to suggest any of the contents #3 and #4 when we are at content #1. However, the average quality per in the states $\#3, \#4$ is zero and in the case where the specifications of the system dictate $\bar{Q}_i \geq q$ for some $q \in [0, 1]$, this policy becomes infeasible.

It is easy to see that the above two policies are quite rigid; none of them jointly considers the cost and the user satisfaction.

Evaluate q-Mixed. That policy can combine both dimensions but does so in a short-sighted way. Thus in our example, it could either recommend cached or related items. Thus, for content #1, the only gain the RS can get is the user satisfaction as $u_{12} = 1$, while for the rest of the contents the RS can split its budget optimally in the myopic sense to the user satisfaction or the cached content as follows

$$\boldsymbol{\pi}_3 = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 1 - q & 0 & q & 0 \\ 1 - q & 0 & 0 & q \\ 1 & 0 & 0 & 0 \end{pmatrix} \tag{2.5}$$

For that policy we have an average quality of $\bar{Q} = [1, q, q, 1]^T$ and an average cost computed as

$$\bar{C}(q, C) = C \cdot (\frac{1}{q^3 + q^2 + 2} - 1)$$

Evaluate Optimal. This policy is able to lay over paths of contents that are more rewarding in the long run while preserving $\bar{Q}_i \geq q \; \forall \; i$. The policy is as follows

$$\boldsymbol{\pi}_4 = \begin{pmatrix} 0 & q & 0 & 1-q \\ 1-q & 0 & q & 0 \\ 1-q & 0 & 0 & q \\ 1 & 0 & 0 & 0 \end{pmatrix} \tag{2.6}$$

This policy achieves $\bar{Q}_i = q \; \forall \; i$ and the average cost

$$\bar{C}(q, C) = C \cdot (\frac{1}{q^2 + q + 2} - 1)$$

Observation. In terms of average cost, if we calculate the difference of the last two policies.

$$\Delta(q, C) = C \cdot (\frac{1}{q^2 + q + 2} - \frac{1}{q^3 + q^2 + 2}) \tag{2.7}$$

thus it is linear on the cost C, which means that it can grow unbounded. Right below we can see average cost of the two policies as a function of q for $C = 1$.

It is really important to notice that the optimal policy does something unintuitive which is to recommend with some probability a content which neither cached nor related. Essentially this toy example shows why myopic policies can fail under the long session regime. We refer the reader to another interesting example that can be found in .1.

2.4 Modeling and Problem Formulation

Given the above setup, our general goal in this paper is *to reduce the total cost of serving user requests by choosing matrix* $\mathbf{R}$*, while maintaining a required recommendation quality.*

Consider a user that starts a session by requesting a content $i \in \mathcal{K}$ with probability $p_0(i)$ (i.e., we assume her initial choice is not affected by the recommender), and then proceeds to request a sequence of contents according to the Markov chain $\mathbf{P}$ of Eq.(2.2). Assume that the user requests M contents in sequence. Then the associated access cost would be given by

$$\frac{1}{M} \sum_{m=1}^{M} \mathbf{p_0}^T \cdot \mathbf{P}^m \cdot \mathbf{c}, \tag{2.8}$$

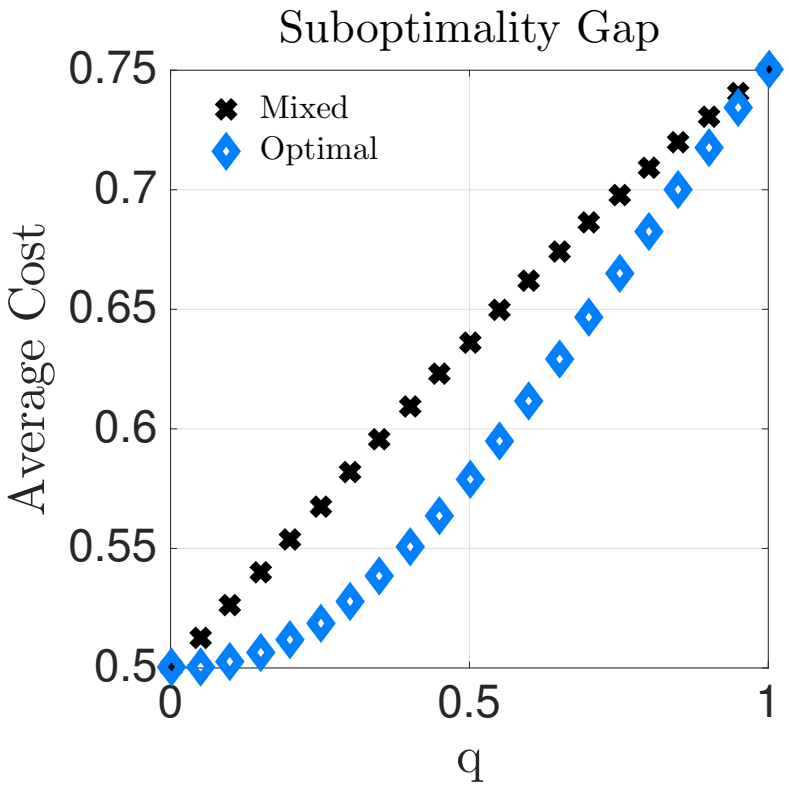

Figure 2.2 – Suboptimality gap for $C = 1$

where $\mathbf{c} = [c_1, ..., c_K]^T$ is the vector of the costs per content.

The expected cost of the request #1 is $\sum_{m=1}^{1} \mathbf{p_0}^T \cdot \mathbf{P}^m \cdot \mathbf{c} = \mathbf{p_0}^T \cdot \mathbf{P}^1 \cdot \mathbf{c}$, the expected cost of the request #2 is $\sum_{m=1}^{2} \mathbf{p_0}^T \cdot \mathbf{P}^m \cdot \mathbf{c}$ and so on until M. To find the average cost we would need to normalize with the number of requests M.

However, M is a random variable, and the various powers of transition matrix $\mathbf{P}$, which contains the control variable Y, would greatly complicate the problem. However, the above Markov chain is strongly connected and ergodic under very mild assumptions for $\mathbf{p}_0$. It thus has a stationary distribution $\boldsymbol{\pi} = [\pi_1, ..., \pi_K]^T$, which is also equal to the long-term percentage of total requests for content i. Consequently, for M large enough we can approximate the average cost *per request* with

$$\boldsymbol{\pi}^T \cdot \mathbf{c} \tag{2.9}$$

where $\boldsymbol{\pi}$ can be calculated from the following lemma.

Lemma 1. *The stationary distribution $\boldsymbol{\pi}$ is given by*

$$\boldsymbol{\pi}^T = (1-\alpha) \cdot \mathbf{p_0}^T \cdot (\mathbf{I} - \alpha \cdot \frac{\mathbf{R}}{N})^{-1} \tag{2.10}$$

where $\mathbf{I}$ the $K \times K$ identity matrix.

Proof. The stationary distribution above can be derived through the standard stationary equality [50]

$$\boldsymbol{\pi}^T = \alpha \cdot \boldsymbol{\pi}^T \cdot \frac{\mathbf{R}}{N} + (1-\alpha) \cdot \mathbf{p_0}^T, \tag{2.11}$$

by observing that matrix $(\mathbf{I} - \alpha \cdot \frac{\mathbf{R}}{N})$ has strictly positive eigenvalues (in measure). See also [51], for more details. □

We are therefore ready to formulate cache-friendly recommendations as an optimization problem.

OP 1 (Cache-Friendly Recommendations)**.**

$$\underset{\mathbf{R}}{\text{minimize}} \quad \mathbf{p_0}^T \cdot (\mathbf{I} - \alpha \cdot \frac{\mathbf{R}}{N})^{-1} \cdot \mathbf{c}, \tag{2.12a}$$

$$\text{subject to} \quad 0 \leq r_{ij} \leq 1, \quad \forall\ i \text{ and } j \ \in\ \mathcal{K}. \tag{2.12b}$$

$$\sum_{j=1}^{K} r_{ij} = N, \quad \forall i \ \in \mathcal{K} \tag{2.12c}$$

$$r_{ii} = 0, \quad \forall\ i \ \in \mathcal{K} \tag{2.12d}$$

$$\sum_{j=1}^{K} r_{ij} \cdot u_{ij} \geq q_i, \quad \forall i \ \in \mathcal{K} \tag{2.12e}$$

Objective. The objective is to minimize the expected cost to access any content, and follows directly from 2.9 and Lemma 1. Note that we have dropped the constant $(1 - \alpha)$ from 2.10, as it does not affect the optimal solution.

Control Variables. The variables r_{ij} (K^2 in total), deciding what is recommended after each content i, constitute the control variables.

Constraints. The first two constraints make sure that r_{ij} forms a stochastic transition matrix that can be translated to N recommendations per item i. Specifically, the "box" constraints of 2.12b ensures that all entries are probabilities. Together with 2.12c these ensure that exactly N contents are recommended for every i (see also Section 4.2, "Recommendation Control"). 2.12d simply ensures that the same content cannot be recommended when it was just consumed.

Quality Constraint. 2.12e ensures that that the "quality" of recommended contents for each i is above a desired threshold. Observe that, without this constraint, the optimal solution to the above problem is trivial, namely to always recommend the same N contents j with the minimum cost c_j. However, these contents will probably be unrelated (i.e. $u_{ij} \to 0$) essentially "breaking" the recommender. Hence, this constraint forces variables r_{ij} to select high u_{ij} values to ensure the recommender keeps doing its primary job, namely finding related contents. Note that, if there are at least N strongly related contents for each i (i.e., $u_{ij} = 1$), then the maximum value for q_i is 1. W.l.o.g., in the remainder we will assume the same quality constraint for all i ($q_i = q$).

The remaining quantities are constants, and inputs to the problem. While some of them might still vary over time (e.g., u_{ij}), we assume this occurs at a larger time scale, compared to our problem.

Unfortunately, the objective function (2.12a) is non-convex, unless $\mathbf{R}$ is positive semidefinite and symmetric. In that case, the problem could be cast into an SDP (Semi-Definite Program) using Schur's complement [52]. However, forcing $\mathbf{R}$ to be symmetric in our problem leads to trivial solutions, as every symmetric Markov chain has a uniform invariant measure. What is more, the inverse in the objective further complicates solving this problem, as the gradient of this expression is rather complex.

2.5 An Algorithm

Given that 1 is non-convex, there are no polynomial-time algorithms that can guarantee to converge to the optimal solution. This leaves us with two options for solving the problem: to apply (i) an exponential-time "global" optimization algorithm (e.g., Branch-and-Bound), or (ii) a heuristic algorithm for an approximate solution. The former is infeasible for all practical scenarios, due to the large problem size (K^2 control variables). Therefore, we will consider two heuristic approaches: in 2.5.1 we consider a "myopic" algorithm, essentially a greedy approach that solves a simpler objective than 1; this algorithm will be our baseline, as it resembles some recent state-of-the-art [53]); in 2.5.2, we propose a more sophisticated algorithm, inspired from ADMM type of schemes [54].

2.5.1 Myopic Algorithm

The non-convexity of 1 is due to the expression of the stationary distribution $\boldsymbol{\pi}$ that appears in the objective function. As mentioned, the stationary distribution captures the long-term behavior of a system where users sequentially consume many contents. To simplify the objective, one could consider a coarse approximation where the recommendation impact is there, but the algorithm "greedily" optimizes the access cost *only for the next content access*. In other words, it is as if a user initially requests a content i, then requests another content j (recommended or not), and then leaves the system. In this case, the objective becomes

$$(\mathbf{p_0}^T \cdot \mathbf{P}) \cdot \mathbf{c}, \tag{2.13}$$

where the first term of 2.8 is dropped (because it is independent of the control variables), and we keep only the second term. This gives rise to the following optimization problem.

OP 2 (Myopic/Single-Step Cache-Friendly Recommendations)**.**

$$\begin{aligned}
\underset{\mathbf{R}}{\text{minimize}} \quad & \mathbf{p_0}^T \cdot (\alpha \cdot \frac{\mathbf{R}}{N} + (1-\alpha) \cdot \mathbf{P}_0) \cdot \mathbf{c}, && \text{(2.14a)}\\
\text{subject to} \quad & 0 \leq r_{ij} \leq 1, \quad \forall\ i \text{ and } j\ \in\ \mathcal{K}. && \text{(2.14b)}\\
& \sum_{j=1}^{K} r_{ij} = N, \quad \forall i\ \in \mathcal{K} && \text{(2.14c)}\\
& r_{ii} = 0, \quad \forall\ i\ \in \mathcal{K} && \text{(2.14d)}\\
& \sum_{j=1}^{K} r_{ij} \cdot u_{ij} \geq q_i, \quad \forall i\ \in \mathcal{K} && \text{(2.14e)}
\end{aligned}$$

In the above problem, the constraints remain intact as the **OP 1**. However, now the objective is linear in $\mathbf{R}$. This is an Linear Program (LP) with affine and box constraints, which can be solved efficiently in polynomial time, using e.g. interior-point methods [52].

Remark. The single-step approach can be interpreted as a projection of the recent work of [53] to our framework. Specifically, the authors solve a similar "single-step" problem, jointly optimizing the caching and recommendation policy. Their problem consists of a two stage algorithm, in the case where the contents are of different size, a knapsack problem is solved for the caching decisions, or else in the case of equisized items they simply the highest popularity contents *and then* they deal with the recommendation problem. Omitting the caching decisions of [53], for the recommendations the authors solve a similar problem to **OP 2**. We should note that their take on the recommendation part refers to a "YouTube Home Page" type of recommendations unlike us, where we place our focus on the related video recommendations list.

2.5.2 Cache-Aware Recommendations for Sequential content access (CARS)

The above "myopic" approach does not exploit the full structure of the Markov chain P. For example, assume there are two contents A and B that are both cached and both have

high similarity with a content currently consumed, but B has slightly higher similarity. The Myopic scheme will choose to recommend B. However, assume that A is similar to many contents that happen to be cached, while B does not. This suggests that, if B is recommended, then in the next step there will be very few good options (hence the algorithm's name): the myopic algorithm will either have to recommend cached contents with low quality or high quality contents which lead to cache misses. To be able to foresee such situations and take the right decisions, we need to go back to the objective of 1.

To circumvent the problem of having the inverse of the control matrix in the objective, we formulate an *equivalent* optimization problem by introducing the stationary vector $\boldsymbol{\pi}$ as an explicit ("auxiliary") control variable.

OP 3 (Cache-Friendly Recommendations: Equivalent Problem)**.**

$$\begin{aligned}
\underset{\boldsymbol{\pi},\mathbf{R}}{\text{minimize}} \quad & \boldsymbol{\pi}^T \cdot \mathbf{c}, && \text{(2.15a)}\\
\text{subject to} \quad & 0 \leq r_{ij} \leq 1, \quad \forall\ i \text{ and } j \ \in\ \mathcal{K}. && \text{(2.15b)}\\
& \sum_{j=1}^{K} r_{ij} = 1, \quad \forall i \ \in \mathcal{K} && \text{(2.15c)}\\
& r_{ii} = 0, \quad \forall\ i \ \in \mathcal{K} && \text{(2.15d)}\\
& \sum_{j=1}^{K} r_{ij} \cdot u_{ij} \geq q_i, \quad \forall i \ \in \mathcal{K} && \text{(2.15e)}\\
& \boldsymbol{\pi}^T = \boldsymbol{\pi}^T \cdot (\alpha \cdot \frac{\mathbf{R}}{N} + (1-\alpha) \cdot \mathbf{p_0}^T) && \text{(2.15f)}\\
& \sum_{j=1}^{K} \pi(j) = 1 && \text{(2.15g)}\\
& \pi_j \geq 0, \quad \forall\ j \ \in \mathcal{K}. && \text{(2.15h)}
\end{aligned}$$

OP 3 enforces three additional constraints. Eq. 2.15g and Eq. 2.15h simply ensure that $\boldsymbol{\pi}$ is a probability distribution. However Eq. 2.15f is an important constraint that ensures that the two problems are equivalent, by forcing $\boldsymbol{\pi}$ to be a stationary distribution related to the transition matrix $\mathbf{P} = \alpha \cdot \frac{\mathbf{R}}{N} + (1-\alpha) \cdot \mathbf{P}_0$. It is easy to see that the two problems have the same set of optimal solutions.

The objective function is now linear in the control variables $\boldsymbol{\pi}$. However, constraint 2.15f is a quadratic equality constraint, and thus the problem remains non-convex. Nevertheless, observe that the problem is now *bi-convex* in the variables $\mathbf{R}$ and $\boldsymbol{\pi}$. Bi-convex problems can often be efficiently tackled with Alternating Convex Search (ACS) methods, that iteratively solve the convex sub-problems for each set of control variables. Unfortunately, such approaches fail here, as the $\mathbf{Y}$ subproblem is simply a feasibility problem ($\mathbf{R}$ does not appear in the objective), and ACS would not converge (our implementation confirms this observation). What is more, having the quadratic equality constraint as a hard constraint does not facilitate such an iterative solution.

Instead, we propose to use a Lagrangian relaxation for that constraint, moving it to the objective. To ensure the strong convexity of the new objective, we form the *Augmented Lagrangian* [54]. Let us first define the function $g(\boldsymbol{\pi}, \mathbf{R})$ as

$$g(\boldsymbol{\pi}, \mathbf{R}) = \boldsymbol{\pi}^T - \boldsymbol{\pi}^T \cdot (\alpha \cdot \frac{\mathbf{R}}{N} - (1-\alpha) \cdot \mathbf{P}_0) \tag{2.16}$$

so that the constraints of 2.15f can be written as

$$g(\boldsymbol{\pi}, \mathbf{R}) = 0 \tag{2.17}$$

The augmented Lagrangian is then given by:

$$\mathcal{L}_\rho(\boldsymbol{\pi}, \mathbf{R}) = \boldsymbol{\pi}^T \cdot \mathbf{c} + g(\boldsymbol{\pi}, \mathbf{R}) \cdot \boldsymbol{\lambda} + \frac{\rho}{2} \cdot (||g(\boldsymbol{\pi}, \mathbf{R})||_2)^2 \tag{2.18}$$

where $\boldsymbol{\lambda}$ is the column vector of length K of the Lagrangian multipliers (one multiplier per quadratic equality), ρ a positive constant scalar, and $||\cdot||_2$ the euclidean norm. This objective is still subject to the remaining constraints of **OP** 3, all of which are now affine. What is more, the problem remains bi-convex in the control variables $\mathbf{R}$ and $\boldsymbol{\pi}$. We can thus apply an ADMM-like method, where we iteratively solve the convex subproblems with respect to $\mathbf{R}$ and $\boldsymbol{\pi}$, but now with the above augmented objective, so that when $g(\boldsymbol{\pi}, \mathbf{R})$ diverges a lot from 0, the subproblem solutions in the inner loop are penalized. We also update the Lagrangian multipliers λ_i at each iteration. Our detailed algorithm is described in Algorithm 1.

Algorithm 1 CARS (Cache-Aware Recommendations for Sequential content access)

Input : $Acc_1, Acc_2, maxIter, N, \mathbf{U}, q, \mathbf{c}, \alpha, \mathbf{p}_0, \rho, \lambda_0, \mathbf{R}_0$

$i \leftarrow 1$
$COST_0 \leftarrow \infty$
$V \leftarrow True$
while V **do**
  $\boldsymbol{\pi}_i = \underset{\boldsymbol{\pi} \in \mathcal{C}}{\operatorname{argmin}} \{\mathcal{L}_\rho(\boldsymbol{\pi}, \mathbf{R}_{i-1})\}$
  $\mathbf{R}_i = \underset{\mathbf{R} \in \mathcal{D}}{\operatorname{argmin}} \{\mathcal{L}_\rho(\boldsymbol{\pi}_i, \mathbf{Y})\}$
  $\lambda \leftarrow \lambda + (\frac{\rho}{2}) \cdot c(\boldsymbol{\pi}_i, \mathbf{R}_i)$
  $COST_i \leftarrow (1-\alpha) \cdot \mathbf{p_0^T} \cdot (\mathbf{I}_{K \times K} - \alpha \cdot \frac{\mathbf{R}_i}{N})^{-1} \cdot \mathbf{c}$
  $\epsilon_1 \leftarrow (|g(\pi_i, \mathbf{R}_i)|_2)^2$
  $\epsilon_2 \leftarrow |COST_i - COST_{i-1}|$
  $V = ((\epsilon_1 > Acc_1) \wedge (\epsilon_2 > Acc_2)) \vee (i \leq maxIter)$
  $i \leftarrow i + 1$
end while
$j \leftarrow \underset{\ell=1,\ldots,i-1}{\operatorname{argmax}} \{COST_\ell\}$
return $\mathbf{R}_j$

Algorithm 1 receives as input the system parameters $N, \mathbf{U}, q, \mathbf{c}, \alpha, \mathbf{p_0}$, and the desired accuracy levels and initialization parameters $Acc_1, Acc_2, maxIter, \rho, \lambda_0, R_0$. It initializes

the objective ($COST_0$) to infinity and starts an iteration for solving the convex subproblems (lines 4–13). In the first leg of the loop (line 5), the augmented Lagrangian $\mathcal{L}_\rho(\boldsymbol{\pi}, \mathbf{R})$ is minimized over $\boldsymbol{\pi}$, considering as constant the variables $\mathbf{Y}$ (equal to their prior value). Then, considers the returned value of $\boldsymbol{\pi}$ from line 5 as constant and minimizes the Lagrangian over the variables $\mathbf{R}$. Both minimization sub-problems are convex and can be efficiently solved. The solution space of the sub-problems C_R and C_π is given by Eqs. (2.15b)–(2.15e) and Eqs.(2.15g)–(2.15h), respectively. After calculating in line 8 the long term $COST$ we get from $\mathbf{R}_i$, the status of the current iteration is computed in the (a) primal residual of the problem (line 9) and (b) the difference of returned $COST$ compared to the previous step (line 10). The algorithm exits the while loop, when the value of the primal residual and improvement in the $COST$ are smaller than the required accuracy, or when the maximum allowable iterations are reached (as described in line 11).

As a final note, the above problem can also be cast into a non-convex QCQP (quadratically constrained quadratic program). State-of-the-art heuristic methods for approximate solving generic QCQP problems [55] are unfortunately of too high computational complexity for problems of this size. It is worth mentioning that we transformed the problem to a standard QCQP formulation and we applied methods based on [55] but the algorithms were only capable of solving small instances of the problem (a few 10s of contents).

Convergence of CARS. Finally, we investigate the performance of CARS (Algorithm 1) as a function of its computational cost, i.e., the maximum number of iterations needed. Fig. 2.3 shows the achieved *actual objective* (red line, circle markers) as measured using $\mathbf{R}$ in Eq. 2.12a at each iteration, and the *virtual cost* (gray line, triangle markers) calculated from the current value of the auxiliary variable $\boldsymbol{\pi}$ as $\boldsymbol{\pi} \cdot \mathbf{c}$, in a simulation scenario (see details in Section 5.6). It can be seen that within 5 iterations, CARS converges to its maximum achieved cache hit ratio. This is particularly important for cases with large content catalogue sizes that require an online implementation of CARS. With $\mathbf{cv}$ we simply denote the hot vector of size $K \times 1$ which has 1's in indexes of cached contents and zero otherwise. We do so in order to show how the metric of interest converges as a result of the equality constraint Eq.(2.15f) approaching feasibility, that is equality.

2.6 Inner ADMM Minimizers Implementation

This short section is dedicated into how the inner minimizations of the $\boldsymbol{\pi}$ and $\mathbf{Y}$ variables carried out during the ADMM algorithm runtime. An important bottleneck of the ADMM implementation we present in the previous section is how fast are the inner minimizers solved. Essentially, if one finds himself against a convex optimization problem (as we do), he can simply use a solver such as CVX [56] and solve it optimally. However, CVX is a generic solver whose purpose is to serve as a benchmark for other solvers because of its accuracy. Obviously, due to its generality, CVX does not scale well when the problem dimensions become very large as it is not designed to solve specific problems, but rather to solve *any* convex problem that might appear. To this end, here we attempt to dissect these inner minimizers and come up with a customized solution for each one of them, that are (1): accurate and (2): fast.

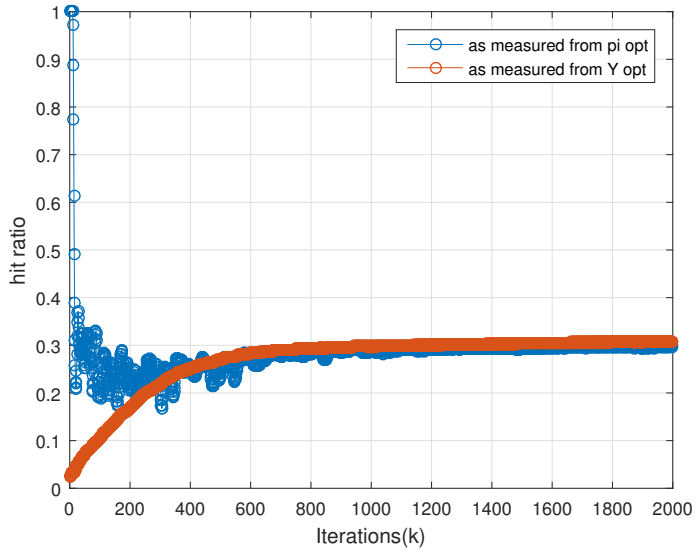

Figure 2.3 – Convergence of the CHR as measured by (1): $\boldsymbol{\pi}^T \cdot \mathbf{cv}$ and (2): $(1-\alpha) \cdot \mathbf{p}_0^T \cdot (\mathbf{I}_{K\times K} - \alpha \cdot \mathbf{R})^{-1} \cdot \mathbf{cv}$

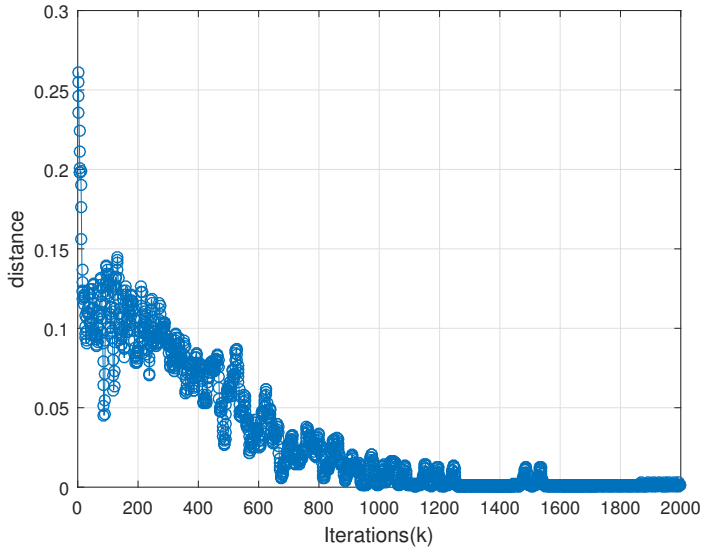

Figure 2.4 – Residual $||c(\boldsymbol{\pi}, \mathbf{R})||_2^2$.

Minimizer over $\boldsymbol{\pi}$

Observe that the objective of Line 2 consists of a smooth strongly convex part (this is due to the ADMM penalty term, a key ingredient for the convergence of the algorithm), that is

$$\mathcal{L}_\rho(\boldsymbol{\pi}) = \boldsymbol{\pi}^T \cdot \mathbf{c} + g(\boldsymbol{\pi}, \mathbf{R}) + \frac{\rho}{2}||g(\boldsymbol{\pi}, \mathbf{R})||_2^2 \tag{2.19}$$

and a non-smooth but convex part, the indicator $\mathbf{I}_{\mathcal{S}}(\mathbf{z})$. Such problems can be efficiently tackled with *proximal methods* [57], with updates

$$\boldsymbol{\pi}^{k+1} = \text{prox}_t(\boldsymbol{\pi}^k - t \cdot \nabla\mathcal{L}(\boldsymbol{\pi}^k)) \tag{2.20}$$

However, the proximal function for an indicator variable simply reduces to the projection of the variable $\boldsymbol{\pi}$ to the constraint set $\mathcal{S}$. Projected gradient methods have good convergence properties for strongly convex problems [57]. Nevertheless, the projection operation itself corresponds, in the general case, to solving a Least Squares (LS) optimization problem

$$\boldsymbol{\pi}^\star = \underset{\boldsymbol{\pi}\in\mathcal{S}}{\text{argmin}}||\boldsymbol{\pi} - \mathbf{v}||_2^2 \tag{2.21}$$

Solving this LS problem explicitly (e.g., with interior-point methods) at every iteration could end up being the bottleneck of our algorithm.

Algorithm 2 Projected Gradient Descent (over $\boldsymbol{\pi}$)

Input: $\alpha, \rho, t_1, \mathbf{c}, \mathbf{p}_0, \boldsymbol{\lambda}^{i-1}, \mathbf{Y}^{i-1}, \boldsymbol{\pi}^{i-1}$
repeat
  $\boldsymbol{\pi}^{k+1} = \boldsymbol{\pi}^k - t_1 \cdot \nabla\mathcal{L}_\rho(\boldsymbol{\pi}^k)$
  $\boldsymbol{\pi}^{k+1} = \mathcal{P}_{\mathcal{S}}(\boldsymbol{\pi}^{k+1})$
until $\mathcal{L}_\rho(\boldsymbol{\pi}^{k+1}) - \mathcal{L}_\rho(\boldsymbol{\pi}^k) \leq \epsilon$
return $\boldsymbol{\pi}^{k+1}$

Lemma 2. *The Subroutine 1 finds the optimal solution for the problem in Line 2 of ALGO-Main in $\mathcal{O}(K\log(K))$ steps.*

Proof. The constraint set $\mathcal{S}$ constitutes a *simplex*, and the projection $\boldsymbol{\pi}^* = \mathcal{P}_{\mathcal{S}}(\mathbf{v})$ can be done fast using the algorithm described in [58]. □

Minimizer over R

The minimization of Line 3 is similar to the one of Line 2 (strongly convex objective). However the constraint set now consists of the *intersection of multiple convex* sets, we call this set $\mathcal{D}$. Although we do not know how to efficiently project onto an intersection

of sets, *we do know* how to project on each of these sets individually. Each of these constraints are affine sets and halfspaces, whose projections are known operations [57]. Implementing projected gradient for this problem is harder as we are now constrained not onto a single set, but onto an intersection of sets. The constrained minimization of ($\mathbf{R}$) is a far more challenging problem due to the additional constraints (set $\mathcal{D}$). A common practice for such problems is to employ off-the-Shelf commercial interior-point method solvers. However, in large scale problems ($\geq$ 1M variables and 1M constraints), second order methods can be slow.

We could thus perform alternating projections, iteratively projecting to each constraint till convergence. Nevertheless, this method guarantees to return a point inside the intersection *but not necessarily the projection.* Instead, for our type of constraints, we need to employ "Dykstra's projection algorithm" [59], to obtain the projection to $\mathcal{D}$.

Algorithm 3 Projected Gradient Descent (over $\mathbf{Y}$)

Input: $\alpha, \rho, t_2, \mathbf{c}, \mathbf{p}_0, \boldsymbol{\lambda}^{i-1}, \mathbf{R}^{i-1}, \boldsymbol{\pi}^i$
$d \leftarrow 2 \cdot K + 2$ ▷ (The # of sets we will project onto)
repeat
  $\mathbf{R}^{k+1} = \mathbf{R}^k - t_2 \cdot \nabla \mathcal{L}_\rho(\mathbf{R}^k)$
  for j = 1,...,K **do** ▷ (Each row independently)
    $\mathbf{p} = \mathbf{w} = \mathbf{z} = 0$
    $\mathbf{x}^{temp} = 0$
    $k \leftarrow 1$
    repeat ▷ (Dykstra's Projections)
      $\mathbf{x}^1 = \mathcal{P}_{\mathcal{C}_1}(\mathbf{x}^{temp} + \mathbf{p})$
      $\mathbf{p} \leftarrow \mathbf{x}^{temp} + \mathbf{p} + \mathbf{x}^1$
      $\mathbf{x}^2 = \mathcal{P}_{\mathcal{C}_2}(\mathbf{x}^1 + \mathbf{w})$
      $\mathbf{w} \leftarrow \mathbf{x}^1 + \mathbf{w} + \mathbf{x}^2$
      $\mathbf{x}^3 = \mathcal{P}_{\mathcal{C}_3}(\mathbf{x}^2 + \mathbf{z})$
      $\mathbf{z} \leftarrow \mathbf{x}^2 + \mathbf{z} + \mathbf{x}^3$
      $\mathbf{x}(i) = 0$
      Compute dev1 and dev2
      $\mathbf{x}^{temp} = \mathbf{x}^3$
    until dev1 and dev2 $< \epsilon$
    $\mathbf{R}(i)^{k+1} = \mathbf{x}^3$
  end for
until $\mathcal{L}_\rho(\mathbf{R}^{k+1}) - \mathcal{L}_\rho(\mathbf{R}^k) \leq \epsilon$
return $\mathbf{R}^{k+1}$

Lemma 3. *The Subroutine 2 finds the optimal solution for the problem in Line 3 of ALGO-Main efficiently.*

Proof. The constraint set $\mathcal{D}$ constitutes an intersection of box, affine and halfspace constraints. Dykstra's algorithm guarantees to return the projection after the gradient step [59]. Furthermore, our constrained set consists of convex sets whose projection

Table 2.2 – Comparing Customized Projected Gradient and CVX for Inner Minimizers

	Cache Hit Rate (%)		Execution Time (s)	
Library Size	Customized	CVX	Customized	CVX
$K = 20$	0.8152	81.29	0.9224	6.1482
$K = 50$	0.8067	0.8070	5.4988	18.0016
$K = 80$	0.8059	0.8061	18.2915	181.0224

operations have *analytical formulas*, therefore can be done fast [60, 57], avoiding to solve explicitly the LS minimization problem. □

For completeness we will give here a small table of objective values and execution times we observed. These experiments were carried out using a portable MacBook Air with (1) RAM: 8 GB 1600 MHz DDR3 and (2) Processor: 1,6 GHz Dual-Core Intel Core i5.

The results presented at Table 2.2 were carried out assuming the following parameters: K varying, $C = 3$ (we cache the most popular contents according to $\mathbf{p}_0$ (zipf distribution with parameter $s = 0.2$), $N = 2$, $\alpha = 0.8$ and $q_i = 0.7$ for all contents. As an observation we can say that as expected, the customized inner minimizers helped as to execute the whole ADMM loop faster than cvx, and that is mainly due to the fact that CVX runs second order methods in the background which are inherently slower than the first order ones.

2.7 Results

In this section, we investigate the improvements in caching performance by the proposed cache-aware recommendation algorithm on top of a preselected caching allocation. We perform simulations using real datasets of related contents (movies and songs), collected from online databases. We first briefly present the datasets (5.6.1) and the simulation setup (5.6.2), and then present simulation results in a wide range of scenarios and parameters and discuss the main findings and implications (5.6.3)

2.7.1 Datasets

We collect two datasets that contain ratings about multimedia content. We use this information to build similarity matrices U, which are later used in the selection of recommendations, e.g., to satisfy a minimum recommendation quality q (as defined in 2.4).

MovieLens. We use the $100k$ subset from the *latest Movielens* movies-rating dataset from the MovieLens website [61], containing 69162 ratings (from 0.5 to 5 stars) of 671 users for 9066 movies. To generate the matrix U of movie similarities from the raw information of user ratings, we apply a standard collaborative filtering method [62]. Specifically, we first apply an item-to-item collaborative filtering (using 10 most similar

items) to predict the missing user ratings, and then use the cosine-distance ($\in [-1, 1]$) of each pair of contents based on their common ratings

$$sim(i,j) = \frac{\sum_{n=1}^{\#users} r_n(i) \cdot r_n(j)}{\sqrt{\sum_{n=1}^{\#users} r_n^2(i)} \cdot \sqrt{\sum_{n=1}^{\#users} r_n^2(j)}}$$

where we normalized the ratings r_i, by subtracting from each rating the average rating of that item. We build the matrix U by saturating to values above 0.6 to 1, and zero otherwise, so that $u_{nk} \in \{0, 1\}$.

Last.fm. We use the subset of *The Million Song Dataset* from the Last.fm database [63], containing $10k$ song IDs. The dataset was built based on the method "getSimilar", and thus it contains a $K \times K$ matrix with the similarity scores (in [0,1]) between each pair of songs in the dataset, which we use as the matrix U. As the Last.fm dataset is quite sparse and we set the non zero values u_{ij} to one to make a binary U in that dataset as well.

To facilitate simulations, we process both datasets, by removing rows and columns of the respective U matrices with $\sum_{j \in \mathcal{K}} u_{ij} \leq N$ (where number $N = 4$ is the number of total recommendations). After the preprocessing, we ended up with a content catalogue of size $K = 1060$ and $K = 757$ for MovieLens and Last.fm traces respectively.

2.7.2 Simulation Setup

Content Demand. The users generate 40000 requests for contents in a catalogue $\mathcal{K}$; requests are either *direct* with probability $\mathbf{p_0} \sim Zipf(s)$ (s the exponent of the Zipf law) for any content, or *recommended* with probability $\frac{1}{N}$ for each of the recommended contents. We consider scenarios with exponent $s \in [0.4, 0.8]$ and $N = 4$. Unless otherwise stated, we set the default value $\alpha = 0.8$, similarly to the statistics in [16].

Caching Policy. We consider a popularity based caching policy, where the C most popular (w.r.t. $\mathbf{p_0}$) contents are locally cached in the base station. This policy is optimal in a single cache network, when no recommendation system is employed.

Recommendation policy. We simulate scenarios under the following three recommendation policies:

- *No Recommendation*: This is is also a baseline scenario, where users request contents only based on $\mathbf{p_0}$ (or, equivalently $a = 0$).
- *Myopic policy*: Cache-aware recommendations using the algorithm of 2.5.1, which optimizes recommendations assuming single-step content requests. This policy relates to the previous works of [34, 53].
- *Proposed Policy - CARS*: Cache-Aware Recommendations using *CARS*, which optimizes recommendations for sequential content consumption.

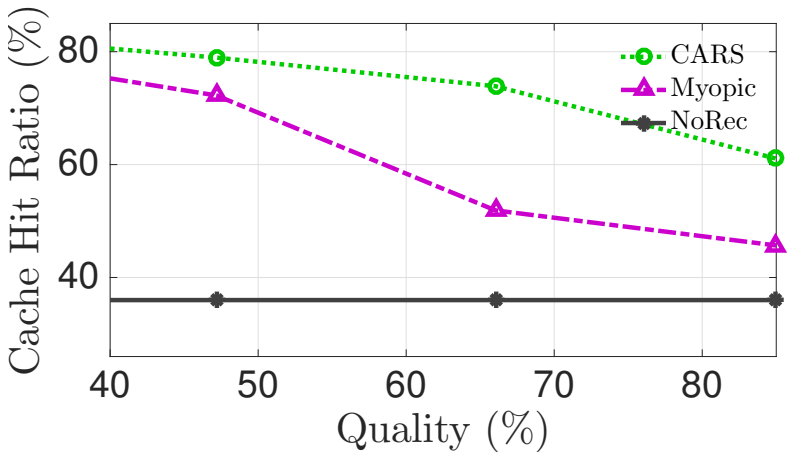

Figure 2.5 – Cache Hit Ratio vs Quality $N = 4,\ C/K = 5\%$. (MovieLens, $s = 0.7$)

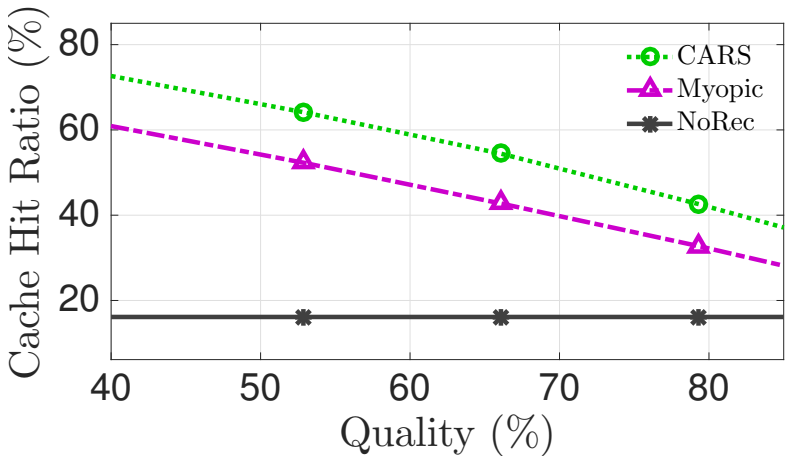

Figure 2.6 – Cache Hit Ratio vs Quality $N = 4,\ C/K = 5\%$. (Last.fm, $s = 0.4$)

2.7.3 Results

We compare the three recommendation policies in scenarios with varying q (minimum quality of recommendations - see 2.4), cache size C, probability to request a recommended content a, and N recommended contents. For simplicity, we assume costs $c = 0$ for cached contents and $c = 1$ for non-cached. Hence, the cost becomes equivalent to the cache hit ratio ($CHR = (1 - \alpha) \cdot \mathbf{p_0}^T \cdot (\mathbf{I} - \alpha \cdot \mathbf{Y})^{-1} \cdot (1 - \mathbf{c})$), which we use as metric to measure the achieved performance in our simulations.

Impact of Quality of Recommendations. Recommending cached contents becomes trivial, if no quality in recommendations is required. However, the primary goal of a content provider is to satisfy its users, which translates to high quality recommendations. In the following, we present results that show that the proposed *CARS can always achieve a good trade-off between cache hit ratio and quality of recommendation, significantly outperforming baseline approaches.*

In Figures 2.5 and 2.6 we present the achieved cache hit ratio (y-axis) of the four recommendation policies for the MovieLens and Last.fm, datasets, respectively, in scenarios where the recommender quality is imposed to be above a predefined threshold q (x-axis). The first observation is that *Myopic* and *CARS* achieve their goal to increase the CHR compared to the baseline case of *NoRec*. The absolute gains for both policies increases for lower values of q, because for lower q there is more flexibility in recommendations. For high values of q, close to 100%, less recommendations that "show the cache" are allowed, and this leads to lower gains. However, even when the quality reaches almost 100%, the gains of *CARS* remain significant. In fact, the relative performance of *CARS* over the *Myopic* increases with q, which indicates that non-*Myopic* policies are more efficient when high recommendation quality is required.

Moreover, comparing Figures 2.5 and 2.6 reveals that the achievable gains depend also on the similarity matrix U. While in Fig. 2.6 both cache-aware recommendation policies follow a similar trend (for varying q), in Fig. 2.5 for the larger dataset of MovieLens, the performance of *CARS* decreases much less compared to *Myopic* with q.

Impact of Caching Capacity. In Figures 2.7 and 2.8 we investigate the performance of the recommendation policies with respect to the cache size, for a fixed value of the recommender quality q. The proposed algorithm, outperforms significantly the other two policies. For example, in Fig. 2.8, for $C/K = 8\%$ it achieves a 25% improvement over the *Myopic* algorithm. Even in the case of the MovieLens dataset (Fig. 2.7), where the *Myopic* algorithm can only marginally improve the cache hit ratio, *CARS* still achieves significant gains. In total, in all scenarios we considered, *the relative caching gains from the proposed cache-aware recommendation policy (over the no-recommendation case) are consistent and even increase with the caching size.*

Impact of Sequential Content Consumption. *CARS* takes into account the fact that users consume more than one content sequentially, and optimizes recommendations based on this. On the contrary the *Myopic* algorithm (similarly to previous works [34], [53]) considers single content requests. Therefore, Algorithm 1 is expected to perform better as the average number of consecutive requests by a user increases. The simulation results in Fig. 2.9 validate this argument. We simulate a scenario of a small catalogue

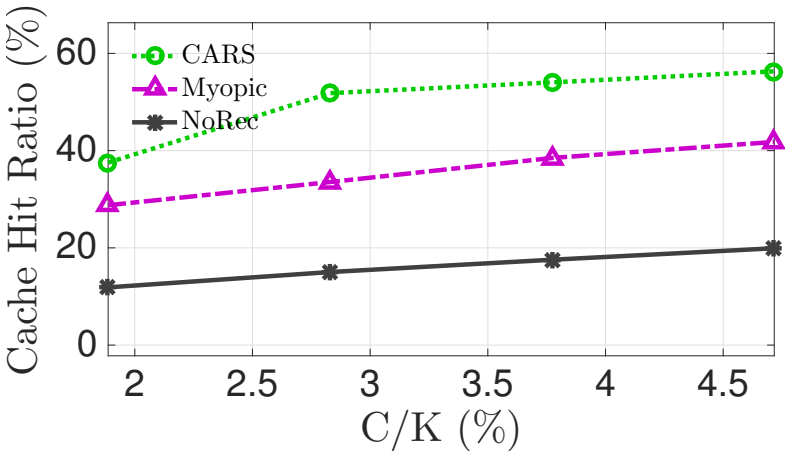

Figure 2.7 – Cache Hit Ratio vs Relative Cache size, $Q = 80\%$, $N = 4$. (MovieLens, $s = 0.5$)

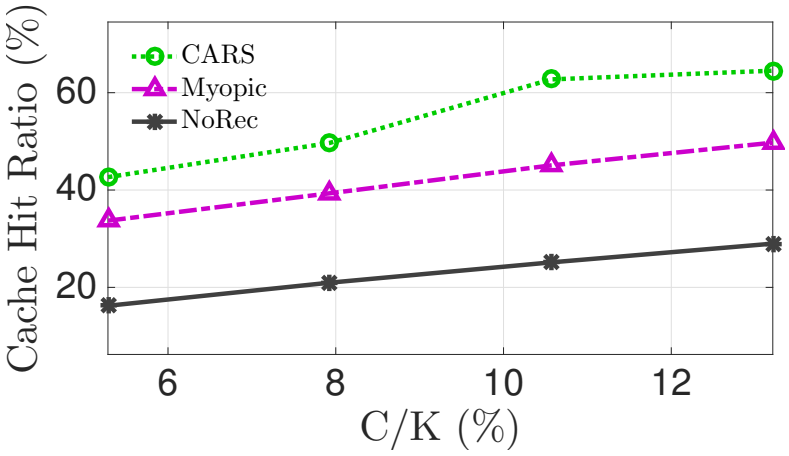

Figure 2.8 – Cache Hit Ratio vs Relative Cache size, $Q = 80\%$, $N = 4$. (Last.fm, $s = 0.4$)

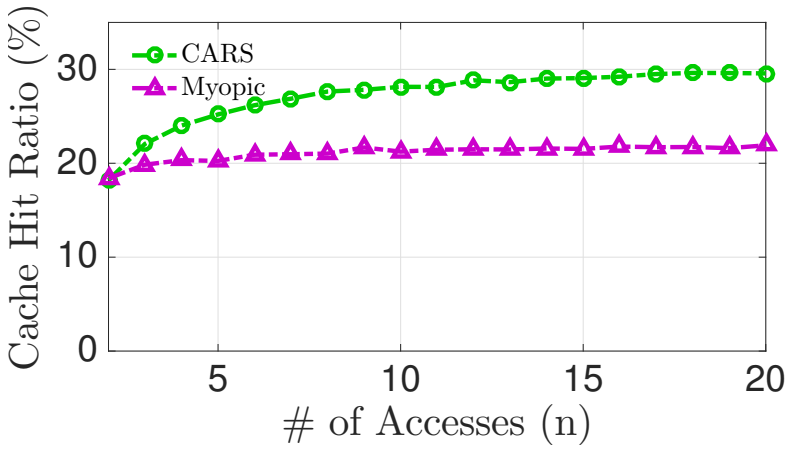

Figure 2.9 – CHR vs # of Accesses, for $N = 3$, (synthetic scenario), $q = 85\%$, $s = 0.2$, $C/K = 4\%$

$K = 100, C = 4, N = 3, s = 0.6, q = 90\%$ and a $\mathbf{U}$ matrix with an $\overline{R} = 4$ related contents per content on average, where we vary the number of consecutive requests by each user. It can be seen that the *Myopic* algorithm increases the cache hit ratio when the users do a few consecutive requests (e.g., 3 or 4); after this point the cache hit ratio remains constant. However, under *CARS*, not only the increase in the cache hit ratio is higher, but it increases as the number of consecutive requests increase. This is a promising message for real content services (such as YouTube, Netflix, Spotify, etc.) where users tend to consume sequentially many contents.

Impact of Probability α. The probability α represents the *frequency* that a user follows a recommendation rather than requesting for an arbitrary content (restart probability, e.g., through the search bar in YouTube). The value of a indicates the influence of the recommendation system to users; in the cases of YouTube and Netflix it is approximately 0.5 and 0.8 respectively [15], [16]. In Fig. 2.10 we present the performance of the two cache-aware recommendation policies for varying values of α. The higher the value of α, the more frequently a user follows a recommendation, and thus the higher the gains from the cache-aware recommendation policies. However, while the gain from the *Myopic* algorithm increases linearly with α, the gains from the proposed *CARS* increase superlinearly. This is due to the fact that Algorithm 1 takes into account the effect of probability α when selecting the recommendations (e.g., see the objective function of 3).

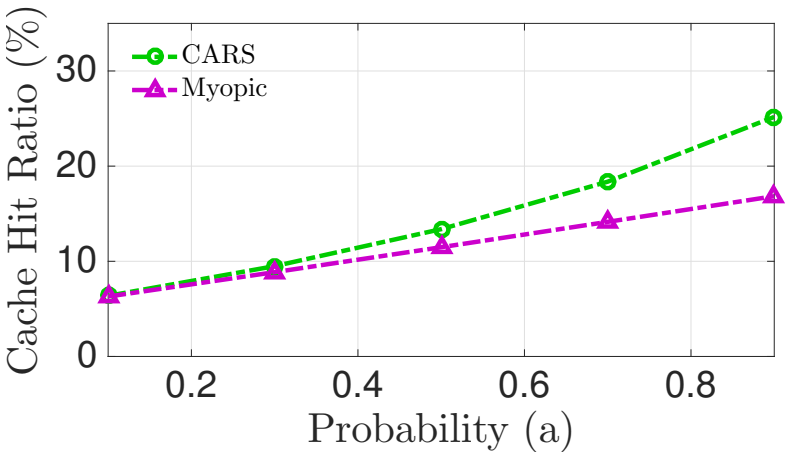

Figure 2.10 – CHR vs Probability α, for $N = 3$, (synthetic scenario), $q = 90\%$, $s = 0.6$, $C/K = 2.5\%$

Chapter 3

LP Transformation and the Non Uniform Click-through Case

3.1 Introduction

In the previous Chapter we approximated the very long session with an infinite one and showed that significant gains are possible. However we have left two key questions unanswered.

Firstly, in our analysis we formulated a nonconvex, in fact biconvex, problem and we applied a variant of ADMM upon it. Although that attempt seems reasonable due to the problem structure, i.e., a set of K challenging biconvex equality constraints, and performs very well in practice, there are some serious pitfalls in this approach. The ADMM algorithm and its convergence for nonconvex problems remains an open question [64, 65] and importantly when it comes to optimality guarantees even less is known. Moreover, algorithms such as the ADMM we designed in Chapter 2, need a fine tuning in order to work properly. Especially in our case where the inner minimizers are implemented using a gradient descent method, extra care is needed as there are even more parameters to tune such as step sizes etc. In this Chapter, we bypass all the aforementioned problematic aspects of our solution, as we perform a transformation on the problem we presented previously and establish the conditions under which the two problems are absolutely equivalent.

Secondly, it has been shown that the users have the tendency to click on recommended contents (or products in the case of e-commerce) according to the position they find them, e.g., contents higher up in the recommendation list [32, 15]. However, several of the aforementioned studies tend to ignore this aspect [44, 66, 31] in their analysis, assuming that an equally good recommendation will be clicked equally frequently, regardless of the position in the application GUI that it appears. The work in [35], while taking into account the ranking of the recommendations in the modeling and their proposed algorithm, in the simulation section they assume that the boosting of the items is equal. So an interesting question arising then is: *Does the performance of network-friendly recommendation schemes improve, deteriorate, or remains unaffected by such position*

preference?

3.2 Problem Setup

3.2.1 Recommendation-driven Content Consumption.

We consider a user that consumes one or more contents during a session, drawn from a catalogue $\mathcal{K}$ of cardinality K. It is reported that YouTube users spend on average around 40 minutes at the service, viewing several related videos [47]. After each viewing, a user is offered some recommended items that she might follow or not, according to the model below.

Definition 2 (Recommendation-Driven Requests)**.** *After a user consumes a content, N contents are recommended to her (these might differ between users).*

- with probability $1-\alpha$ *($\alpha \in [0,1]$) she ignores the recommendations, and picks a content j (e.g., through a search bar) with probability $p_j \in (0,1)$,* $\mathbf{p}_0 = [p_1, p_2, ..., p_K]^T$.
- with probability α *she follows one of the N recommendations.*
- *each of the N recommended contents is placed in one of N possible slots/positions in the application GUI; if she* does *follow recommendation, the* conditional *probability to pick the item in position i is v_i, where $\sum_i v_i = 1$.*

We assume the probabilities $p_0(j)$ capture long-term user behavior (beyond one session), and possibly the impact of the baseline recommender. W.l.o.g. we also assume $\mathbf{p}_0$ governs the first content accessed, when a user starts a session. This model captures a number of everyday scenarios (e.g., watching clips on YouTube, personalized radio, etc).

The last point in the definition is a key differentiator of this work, compared to some previous ones on the topic [44], [35], [31]. A variety of recent studies [32, 15] has shown that the web-users have the tendency to click on contents (or products in the case of e-commerce) according to the position they find them. For example, in the PC interface of YouTube, they show a preference for the contents that are higher in the list of the recommended items. Hence, the probability of picking content in position 1 (v_1), might be quite higher than the probability to pick the content in position N (v_N)[1]. In contrast, [44, 35, 31] explicitly or implicitly assume that $v_i = \frac{1}{N}, \forall i$.

Remark - Position Entropy: A key goal of this paper is to understand the additional impact of position preference on the achievable gains of network-friendly recommendations. A natural way to capture position preference is with the *entropy* of the probability mass function $\mathbf{v} = [v_1, v_2, ..., v_N]$, namely

$$H_{\mathbf{v}} = H(v_1, .., v_N) = -\sum_{n=1}^{N} v_n \cdot \log(v_n). \tag{3.1}$$

[1] In fact, a Zipf-like relation has been observed [15].

The original case of no position preference, corresponds to a uniformly distributed $\mathbf{v}$, which is well known to have maximum entropy. Any position preference will lead to lower entropy, with the extreme case of a "1-hot vector" (i.e., only one $v_i = 1$) having zero entropy.

Content Retrieval Cost. We assume that fetching content i is associated with a generic cost $c_i \in \mathbb{R}$, $\mathbf{c} = [c_1, c_2, ..., c_K]^T$, which is known to the content provider, and might depend on access latency, congestion overhead, or even monetary cost.
Maximizing cache hits: Can be captured by setting $c_i = 1$ for all cached content and to $c_i = 0$, for non-cached content.
Hierachical caching: Can be captured by letting c_i take values out of n possible ones, corresponding to n cache layers: higher values correspond to layers farther from the user [48, 4].

3.2.2 Baseline Recommendations

For simplicity, we assume that the baseline RS works as follows:

Definition 3 (Baseline Recommendations and Matrix $\mathbf{U}$)**.** *(i) For every pair of contents $i, j \in \mathcal{K}$ a score $u_{ij} \in [0, 1]$ is calculated, using a state-of-the-art method. Note that these scores can be personalized, and differ between users.*[2]
(ii) After a user has just consumed content i, the RS recommends contents according to these u_{ij} values (e.g., the N contents j with the highest u_{ij} value [15, 12].[3]

3.2.3 Network-friendly Recommendations.

Our goal is to depart from the baseline recommendations (Def. 3) that are based only on $\mathbf{U}$, and let them consider the access costs $\mathbf{c}$ as well. We define recommendation decisions as follows.

Definition 4 (Control Variables $\mathbf{R}^1, .., \mathbf{R}^N$)**.** *Let $r_{ij}^n \in [0, 1]$ denote the probability that content j is recommended after a user watches content i in the position n of the list. For the n-th position in the recommendation list, these probabilities define a matrix $K \times K$ recommendation matrix, which we call $\mathbf{R}^n$.*

Defining recommendations as probabilities provides us more flexibility, as it allows to not always show a user the same contents (after consuming some content i). For example, assume $K = 4$ total files, a user just watched item 1, and $N = 2$ items must be recommended. Let the first row of the matrix $\mathbf{R}^1$ be $\mathbf{r}_1^1 = [0, 1, 0, 0]$ and that of $\mathbf{R}^2$ be $\mathbf{r}_1^2 = [0, 0, 0.5, 0.5]$. In practice, this means that in position 1 the user will always see content 2 being recommended (after consuming content 1), and the recommendation

[2] u_{ij} could correspond to the *cosine similarity* between content i and j, in a collaborative filterting system [10], or simply take values either 1 (for a small number of related files) and 0 (for unrelated ones). These scores might also depend on user preferences and past history of that user, as is often the case when users are logged into the app.

[3] N depends on the scenario. E.g., in YouTube $N = 2, .., 5$ in its mobile app, and $N = 20$ in its website version.

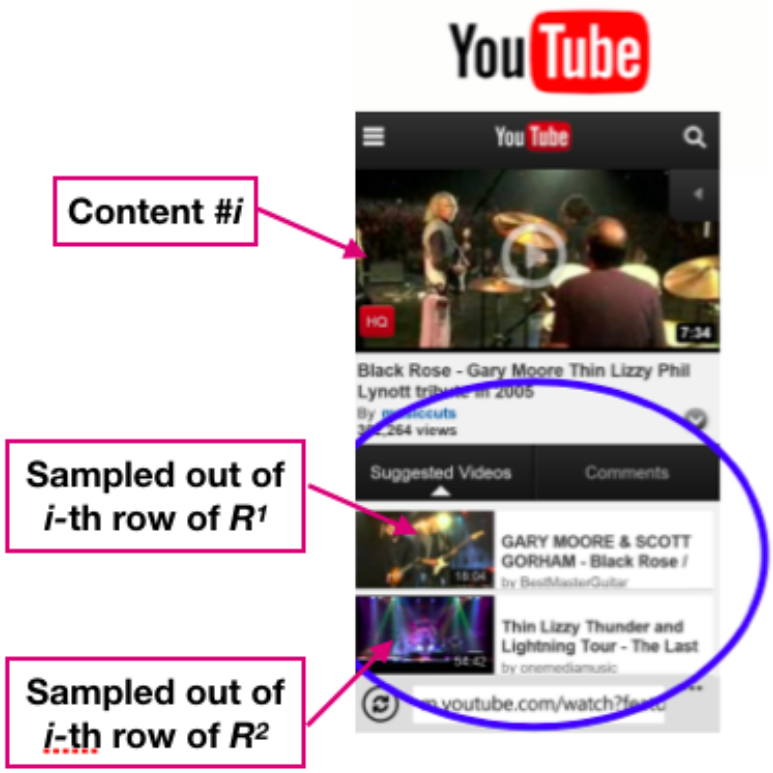

Figure 3.1 – Example of a Related Items List along with the respective click-through probabilities per position.

for position 2 will half the time be for content 3 and half for content 4. In Fig.3.1, a related video list of some video i is depicted along with the click-through probabilities per position.

Our objective is to choose what to recommend in which position, i.e., choose $\mathbf{R}^1, ..\mathbf{R}^N$, to minimize the average content access cost. However, we still need to ensure that the user remains generally happy with the quality of recommendations and does not abandon the streaming session.

Recommendation Quality Constraint.

Let $r_{ij}^{n(B)}$ denote the baseline recommendations of Def .3. We can define the recommendation quality of this baseline recommender for content i, q_i^{max} as follows

$$q_i^{max} = \sum_{j=1}^{K} \sum_{n=1}^{N} v_n \cdot r_{ij}^{(n)(B)} \cdot u_{ij}. \tag{3.2}$$

This quantity will act as another figure of merit for other (network-friendly) RS.

Definition 5 (Quality of Network-Friendly Recommendations)**.** *Any other (network-friendly) RS that differs from the baseline recommendations r_{ij}^B can be assessed in terms*

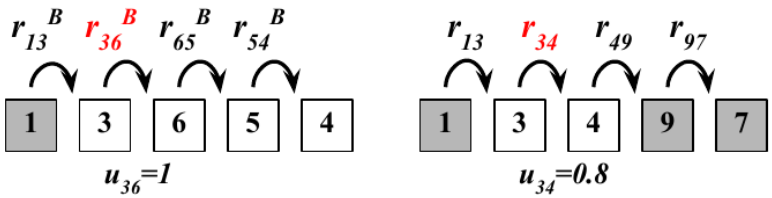

Figure 3.2 – Comparison of baseline (left) and network-friendly (right) recommenders. Gray and white boxes denote cached and non-cached contents, respectively. Recommending after content 3 a slightly less similar content (i.e., content 4 instead of 6), leads to lower access cost in the long term.

of its recommendation quality $q \in [0, 1]$ *with the constraint:*

$$\sum_{j=1}^{K} \sum_{n=1}^{N} v_n \cdot r_{ij}^n \cdot u_{ij} \geq q \cdot q_i^{max}, \forall i \in \mathcal{K}. \tag{3.3}$$

where q_i^{max} *is the quantity defined in Eq.(3.2).*

This equation weighs each recommendation with: (a) its quality u_{ij}, and (b) the importance of the position n it appears at, v_n. Note however that this constraint is not a restrictive choice. One could conceive a more "aggresive" recommender that removes the weight v_n from the left-hand side. In fact, our framework can handle any quality constraint(s) that are convex in r_{ij}^n.

Based on the above discussion, a network-friendly recommendation could favor at each step contents j (i.e., give high r_{ij}^n values) that have low access cost c_j but also are interesting to the user (i.e., have high u_{ij} value). However, as we show in later sections, such a greedy approach is suboptimal, as the impact of r_{ij}^n goes beyond the content j accessed next, affecting the entire *sample path* of subsequent contents in that session. The example in Fig. 3.2 depicts such a scenario: after content 3, instead of recommending content 6 (related value $u_{36} = 1$) content 4 is recommended ($u_{34} = 0.8$), because 4 is more related to cached contents (9 and 7) that can be recommended later (whereas 6 is related to the non-cached contents 5 and 4).[4]

Remark on Recommendation Personalization. As hinted at earlier, content utilities u_{ij} and recommendations r_{ij}^n can be user-specific (e.g. u_{ij}^u for user u), since different users might have different access patterns that can be leveraged. Nevertheless, to avoid notation clutter we do not use superscript u in the remainder of the paper, and will assume that these quantities and the respective optimization algorithm is done per user.

Remark on Recommendation Quality. Cache-friendly recommendations might also improve user QoE, in addition to network cost, a "win-win" situation. Today's RS, measure their performance (QoR) without taking into account where the recommended content is stored. Assuming two contents equally interesting to the user where the one *is*

[4]The reason is that many contents j will have high enough relevance u_{ij} to the original content i, and are thus interchangeable [15]

Table 3.1 – Important Notation

α	Prob. the user follows recommendations
r_{ij}^n	Prob. to recommend j after i at position n
q_i^{max}	Maximum baseline quality of content i
q	Percentage of original quality
$\mathbf{p}_0$	Baseline popularity of contents
u_{ij}	Similarity scores content pairs $\{i, j\}$, included in $\mathbf{U}$
v_n	Click prob. of recommendation at the position n
c_i	Access cost for content i
$\mathcal{K}$	Content catalogue (of cardinality K)
N	Number of recommendations

stored locally while the other *is not*; it is obvious that the cached one could be streamed in much better quality (e.g., HD, so higher QoS), thus leading to $q > 1$. Hence, more sophisticated QoE (= QoR + QoS) metrics *could* combine these effects: e.g., a content's effective utility $\hat{u}_{ij} = f(u_{ij}, c_j)$ that increases if j is highly related to i but also if it is locally cached (i.e., c_j is low). Such a metric could be immediately integrated into our framework, simply be replacing u with $\hat{u}$.

Table 5.1 summarizes important notation. Vectors and matrices are denoted with bold symbols.

3.3 Problem Formulation

Having defined the content access model, our first step towards "optimizing" the (network-friendly) recommendations, is to better understand what we are trying to optimize. To this end, in this section we derive the expected content access cost for a typical user session, as a function of recommendation variables r_{ij}^n. This will serve as the *objective* of our problem.

Definition 6. *Let $S = \{i_1, i_2, \ldots, i_s\}, i_n \in \mathcal{K}$ be a sequence of contents accessed by a user according to Def. 2 during a viewing session. Then S is a discrete-time Markov process with transition matrix*

$$\mathbf{P} = \alpha \cdot \sum_{n=1}^{N} v_n \cdot \mathbf{R}^n + (1 - \alpha) \cdot \mathbf{1} \cdot \mathbf{p}_0^T, \tag{3.4}$$

where $\mathbf{1} = [1, 1, ..., 1]^T$ is a column vector of all 1s.

When the user has just consumed content i, then she might next consume content j if all the following occur: she decides to follow a recommendation (probability α according to Def. 2), j appears in the position n (probability r_{ij}^n), and she picks the content at the n-th position (probability v_n). These probabilities are by definition independent, hence the probability of these three events is their product, $\alpha \cdot v_n \cdot r_{ij}^n$. Note that the user might

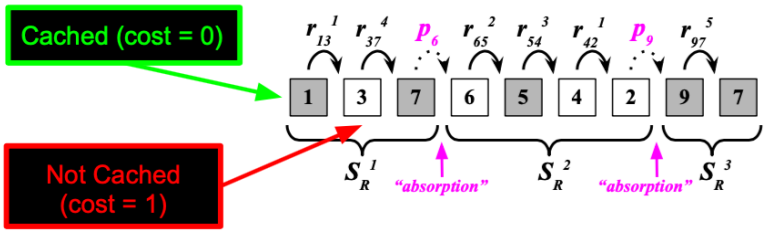

Figure 3.3 – Example of a multi-content session. The instant where the user requests for some content from the seatch/text bar signifies the absorption of the Markov Chain. Moreover the color of the contents expresses their network cost.

consume j, if she finds it in positions other than n (for example in position m) and will then click it with v_m. Moreover, the user might also consume j after i, if she ignores the recommendations (with probability $1-\alpha$ according to Def. 2) and picks content j from the entire catalog (with probability p_j). Putting all these together gives the transition probability from i to j, $Pr\{i \to j\} = \alpha \cdot \sum_{n=1}^{N} v_n \cdot r_{ij}^n + (1-\alpha) \cdot p_j$, which written in matrix notation gives 3.4.

Lemma 4 (Content Access as Renewal-Reward)**.** *A content access sequence* $S = \{S_R^1, S_R^2, \ldots\}$ *defines a renewal process, with subsequences* S_R*, where the user follows recommended content, each ending with a jump outside of the recommender. The cost* c_i *incurred at each state is the reward.*

It is easy to see that whenever a user makes a jump outside of the recommendations (w.p. $1-\alpha$), the process renews to state $\mathbf{p}_0$. An example can be found in Fig. 3.3.

To derive the mean access cost, we employ Lemma 4 and the framework of Absorbing Markov Chains (AMC) [67]: a user is in *transient* states while she is following recommendations; and she gets *absorbed* as soon as a jump outside of recommendations occurs, as shown in Fig. 3.3. Hence, during a content access sequence, recommendations affect the user's choices (and related costs) only during the transient states.

Lemma 5 (Recommendation-Driven Cost)**.** *The content access cost* $C(S_R)$ *during a renewal cycle* S_R *is given by*

$$E[C(S_R)] = \mathbf{p}_0^T \cdot \mathbf{G} \cdot \mathbf{c}, \tag{3.5}$$

and the expected length of such a cycle is

$$|S_R| = \mathbf{p}_0^T \cdot \mathbf{G} \cdot \mathbf{1} = \frac{1}{1-\alpha}, \tag{3.6}$$

where $\mathbf{G} = \left(\mathbf{I} - \alpha \cdot \sum_{n=1}^{N} \cdot \mathbf{R}^n\right)^{-1}$ *is the* Fundamental Matrix *of an AMC with* K *transient states and* 1 *absorbing state, corresponding to a jump outside recommendations.*

Proof. Let a user start a sub-sequence by retrieving content i. The expected number of retrievals of content j (or, number of times visiting state j) until the end of the

sub-sequence is given by g_{ij}, where g_{ij} is the (i-row,j-column) element of the *fundamental matrix* $\mathbf{G}$ of the AMC [67].
The fundamental matrix is defined as follows

$$\mathbf{G} = \sum_{n=0}^{\infty} \mathbf{Q}^n = (\mathbf{I} - \mathbf{Q})^{-1} \tag{3.7}$$

where $\mathbf{Q}$ the matrix with the transition probabilities q_{ij} between the transient states of the AMC ($i, j \in \mathcal{K}$). Following the same arguments as in Def. 6, we get that $q_{ij} = \alpha \cdot \sum_{n=1}^{N} v_n \cdot r_{ij}^n$, or, in a matrix format $\mathbf{Q} = \alpha \cdot \sum_{n=1}^{N} v_n \cdot \mathbf{R}^n$. Substituting this into 3.7 gives the expression for $\mathbf{G}$ that appears in Lemma 5. Now, the cost of retrieving a content j is c_j. Since each content j is retrieved on average g_{ij} times during a sub-sequence that starts from i, the total cost is given by

$$E[C(S_R) \mid i] = \sum_{j \in \mathcal{K}} g_{ij} \cdot c_j \tag{3.8}$$

The probability that a sub-sequence starts at content i is equal for all sub-sessions and is given by p_i. Thus, taking the expectation over all the possible initial states i, gives

$$E[C(S_R)] = \sum_{i \in \mathcal{K}} E[C(S_R) \mid i] \cdot p_0(i) = \sum_{i \in \mathcal{K}} \sum_{j \in \mathcal{K}} g_{ij} \cdot c_j \cdot p_0(i) \tag{3.9}$$

Expressing the above summation as the product of the vectors $\mathbf{p}_0$ and $\mathbf{c}$, and the matrix $\mathbf{G}$, gives 3.5.

Similarly, if g_{ij} is the amount of time spent on state j before absorption, starting from state i, then $\sum_j g_{ij}$ must be equal to the total time spent at *any state* before absorption. Weighing this with the probability $p_0(i)$ of starting at each state i, gives the expected time to absorption, which is the expected duration of a sub-sequence $E[|S_R|] = \sum_i p_0(i) \cdot \sum_j g_{ij}$. Writing this in matrix notation, gives the first part of Eq.(3.6).

However, observe that the probability of absorption at any state i is equal to $1 - \alpha$, independent of i. Hence, the number of steps till absorption is a *geometric* random variable with parameter $1-\alpha$, and thus the mean time (i.e., number of steps) to absorption is $\frac{1}{1-\alpha}$. □

The following Theorem, which gives the expected retrieval cost for a user session, follows immediately from Lemmas 4, 5, and the Renewal-Reward theorem [50]

Theorem 1. *The expected retrieval cost per content, for a user session S, given a recommendation matrix* $\mathbf{R}$ *is*

$$E[C(S) \mid \mathbf{R}^1, \ldots, \mathbf{R}^N] = \frac{\mathbf{p}_0^T \cdot \left(\mathbf{I} - \alpha \cdot \sum_{n=1}^{N} v_n \cdot \mathbf{R}^n\right)^{-1} \cdot \mathbf{c}}{\frac{1}{1-\alpha}} \tag{3.10}$$

Note: It is important to stress again that $\mathbf{R}^n$ denotes the $K \times K$ recommendation matrix of the n-th position of the position of the website/application screen, it simply serves as a superscript and *is not* an exponent.

3.3.1 Optimization Methodology

In this part, we use the results of the previous section to formulate the problem of minimizing the expected access cost until absorption under a set of modeling constraints.

OP 4 (Nonconvex formulation)**.**

$$\underset{\mathbf{R}^1,\ldots,\mathbf{R}^N}{\text{minimize}} \quad \mathbf{p}_0^T \cdot (\mathbf{I} - \alpha \cdot \sum_{n=1}^{N} v_n \cdot \mathbf{R}^n)^{-1} \cdot \mathbf{c} \tag{3.11a}$$

$$\text{subject to} \quad \sum_{j=1}^{K}\sum_{n=1}^{N} v_n \cdot r_{ij}^n \cdot u_{ij} \geq q \cdot q_i^{max},\ \forall\ i \in \mathcal{K} \tag{3.11b}$$

$$\sum_{j=1}^{K} r_{ij}^n = 1,\ \forall\ i \in \mathcal{K}\ \textit{and}\ n = 1, ..., N \tag{3.11c}$$

$$\sum_{n=1}^{N} r_{ij}^n \leq 1,\ \forall\ \{i,j\} \in \mathcal{K} \tag{3.11d}$$

$$0 \leq r_{ij}^n \leq 1\ \ (i \neq j), \quad r_{ii}^n = 0\ \forall\ i,\ n. \tag{3.11e}$$

The constraint in Eq.(3.11b), is responsible for keeping the quality of the recommendations above a pre-specified (and given) threshold. The pair of constraints in Eqs.(3.11c,3.11e), defines a probability simplex for every row of all the $\mathbf{R}^n$ matrices. Note that we also prohibit self-recommendations ($r_{ii}^n = 0\ \forall\ i$ and n) (see Eq.(3.11e)). Importantly, Eq.(3.11d) is necessary in the position-aware setup, to ensure that the same content will not be recommended in two different positions. As an example assume that $\mathbf{r}_1^1 = [0, 1, 0, 0]$ and $\mathbf{r}_1^2 = [0, 0.2, 0.3, 0.5]$, in that case we clearly see that content 2 would always be shown in position 1 (after watching content 1), but 20% of those times it would be shown in position 2 as well. Hence, Eq.(3.11d) ensures that such decision vectors would be *infeasible*. Therefore, constraint Eq.(3.11d), makes sure that each recommendation appears at most once even in the probabilistic setup. Evidently, our feasible space consists of either linear (equalities or inequalities) or box constraints with respect to the decision variables r_{ij}^n. However, the objective is non-convex in general.

Lemma 6. *The problem described in **OP 4** is nonconvex.*

Proof. The problem **OP 4** comprises $N \cdot K^2$ variables r_{ij}^n, and a set of $K^2 \cdot (N+2) + K$ linear (equality and inequality) constraints, thus the feasible solution space is convex. However, assume w.l.o.g that $\mathbf{p}_0 = \mathbf{c} = \mathbf{w}$, $N = 1$, and $v_1 = 1$; the objective now becomes $f(\mathbf{R}) = \mathbf{w}^T \cdot (\mathbf{I} - \alpha \cdot \mathbf{R})^{-1} \cdot \mathbf{w}$. Unless $\mathbf{R}$ is symmetric positive semi-definite (PSD), $f(\mathbf{R})$ is non-convex [52]. Forcing $\mathbf{R}$ to be symmetric would *require additional* constraints that lead to suboptimal solutions of this problem [68]. Therefore, our objective as is, is nonconvex and there are no exact methods that can solve it in polynomial time. □

3.3.2 The Journey to Optimality

In the previous subsection we showed that **OP 4** is nonconvex. However, notice that **OP 4** is informally a *linear combination of* **OP 5** (weighted by $\mathbf{v}$) and thus the nonconvexity

result makes absolute sense. Our ultimate goal is to solve **OP 4**, however for the sake of clarity of presentation, we will use a simpler version of it by assuming $v_i = \frac{1}{N}$. Then if the position *does not matter*, i.e., the user selects contents randomly, it is no longer necessary to have N recommendation matrices, and thus $\mathbf{R}^1 = \cdots = \mathbf{R}^N = \mathbf{R}$. We remind the reader that essentially if we set $\mathbf{Y} = \frac{1}{N} \cdot \mathbf{R}$ we end up with our initial problem of the previous chapter,**OP 5**. Along these lines the problem reduces to

OP 5 (Cache-Friendly Recommendations)**.**

$$\underset{\mathbf{R}}{\text{minimize}} \quad \frac{\mathbf{p}_0^T \cdot \left(\mathbf{I} - \frac{\alpha}{N} \cdot \mathbf{R}\right)^{-1} \cdot \mathbf{c}}{\frac{1}{1-\alpha}}, \tag{3.12a}$$

$$\text{subject to} \quad \sum_{j=1}^{K} r_{ij} \cdot u_{ij} \geq q \cdot q_i^{max}, \quad \forall i \ \in \mathcal{K}, \tag{3.12b}$$

$$\sum_{j=1}^{K} r_{ij} = N, \quad \forall i \ \in \mathcal{K} \tag{3.12c}$$

$$0 \leq r_{ij} \leq 1 \quad (i \neq j), \quad r_{ii} = 0. \tag{3.12d}$$

We proceed by introducing K auxiliary variables $\mathbf{z}^T = \mathbf{p}_0^T \cdot (\mathbf{I} - \frac{\alpha}{N} \cdot \mathbf{R})^{-1}$, which leads to the following equivalent problem.[5]

Intermediate Step (Equivalent formulation)**.**

$$\underset{\mathbf{R}\in\mathcal{D},\ \mathbf{z}\in\mathcal{S}}{\text{minimize}} \quad \mathbf{z}^T \cdot \mathbf{c}, \tag{3.13}$$

$$\text{subject to} \quad \mathbf{z}^T - \frac{\alpha}{N} \cdot \mathbf{z}^T \cdot \mathbf{R} = \mathbf{p}_0^T \tag{3.14}$$

where $\mathcal{D}$ is the convex set formed by the intersection of Eqs.(3.12b)-(3.12d), and

$$\mathcal{S} = \{\mathbf{z} \in \mathbb{R}^K : \mathbf{z} \geq 0, \sum_i z_i = \frac{1}{1-\alpha}\}.$$

The constraints $\mathcal{S}$ for $\mathbf{z}$ follow from Eq.(3.6) (Lemma 5).We have omitted the constant $(1-\alpha)$ in Eq.(5).

The new objective is now convex (in fact linear) in the new variables $(\mathbf{z}, \mathbf{F})$. However, as the set of constraints Eq.(3.14) are all *quadratic equalities*, the problem remains nonconvex. The above formulation falls under the umbrella of non-convex QCQP (Quadratically Constrained Quadratic Program), where it is common to perform a convex relaxation of the quadratic constraints, and then solve an approximate convex problem (e.g., SDP or Spectral relaxation, see [55] for more details). The problem can also be seen as *bi-convex* in variables $\mathbf{R}$ and $\mathbf{z}$, respectively. Alternating Direction Method

[5]Two problems are equivalent if the solution of the one, can be uniquely obtained through the solution of the other [52]; introducing auxiliary variables preserves the property.

of Multipliers (ADMM) can be applied to such problems, iteratively solving convex subproblems [54, 44]. Nevertheless, none of these methods provides any optimality guarantees, and even convergence for non-convex ADMM is an open research topic [69].

Therefore, we introduce a set of new variables f_{ij}, defined as $f_{ij} = z_i \cdot r_{ij}$. Since the j-th element of the vector $\mathbf{z}^T \cdot \mathbf{R}$ can be written as $\sum_i z_i \cdot r_{ij}$, we can write now $\mathbf{z}^T \cdot \mathbf{R} = \mathbf{1}^T \cdot \mathbf{F}$, and the new variables are $\mathbf{z}$ ($K \times 1$ vector) and $\mathbf{F}$ ($K \times K$ matrix).

Intermediate Step (LP formulation)**.**

$$\underset{\mathbf{z}\in\mathcal{S},\ \mathbf{F}}{\text{minimize}} \quad \mathbf{z}^T \cdot \mathbf{c}, \tag{3.15a}$$

$$\text{subject to} \quad \sum_{j=1}^{K} f_{ij} \cdot u_{ij} - z_i \cdot q \cdot q_i^{max} \geq 0, \quad \forall\ i\ \in \mathcal{K}, \tag{3.15b}$$

$$\sum_{j=1}^{K} f_{ij} - z_i = 0, \quad \forall\ i\ \in \mathcal{K} \tag{3.15c}$$

$$f_{ij} - z_i \leq 0 \ \ \forall\ i,j \in \mathcal{K} \tag{3.15d}$$

$$f_{ij} \geq 0\ (i \neq j), \quad f_{ii} = 0 \tag{3.15e}$$

$$z_j - \frac{\alpha}{N} \sum_i f_{ij} = p_0(j), \quad \forall j \in \mathcal{K} \tag{3.15f}$$

Lemma 7. *The change of variables $f_{ij} = z_i \cdot r_{ij}$, is a bijection (one-to-one mapping) between (z_i, r_{ij}) and (z_i, f_{ij}) iff we have $p_i > 0, \forall i$.*

Proof. This follows immediately, as we can readily obtain $r_{ij} = \frac{f_{ij}}{z_i}$ from $\{z_i, r_{ij}\}$. Note that, since $z_j = \sum_i f_{ij} + p_j$, and $p_i > 0, \forall i$ (see Def. 2), this forces $\mathbf{z} > \mathbf{0}$ and thus r_{ij} are always uniquely defined. □

Remark 2. *The condition we need to establish in our case, that is $p_i > 0, \forall\ i$ essentially translates to "all considered contents inside the library have a nonzero probability to be requested by the user".*

Combining Lemma 9, along with the definition of problem in Intermediate Step 1, yields the formulation of Intermediate Step 2. The Intermediate Step 2 problem corresponds to a Linear Program (LP), as it consists of $2K^2 + 4K + 1$ linear constraints. LPs can be solved efficiently with well-established methods like simplex or interior-point, implemented by popular solvers (e.g., CPLEX, GUROBI, etc.).

Lemma 8. *Similarly to **OP 5**, **OP 4** is also convex as it can be cast into an LP.*

Proof. Can be found in Appendix .2. □

This transformation leads to the following problem.

OP 6 (LP formulation)**.**

$$\underset{\mathbf{z},\ \mathbf{F}^1,\ldots,\mathbf{F}^N}{\textit{minimize}} \quad \mathbf{c}^T \cdot \mathbf{z}, \tag{3.16a}$$

$$\textit{subject to} \quad \sum_{j=1}^{K}\sum_{n=1}^{N} v_n \cdot f_{ij}^n \cdot u_{ij} - z_i \cdot q \cdot q_i^{max} \geq 0,\ \forall\ i\ \in \mathcal{K} \tag{3.16b}$$

$$\sum_{j=1}^{K} f_{ij}^n - z_i = 0,\ \forall\ i\ \in \mathcal{K}\ \textit{and}\ n = 1, .., N \tag{3.16c}$$

$$\sum_{n=1}^{N} f_{ij}^n - z_i \leq 0,\ \forall\ \{i,\ j\} \in \mathcal{K} \tag{3.16d}$$

$$f_{ij}^n \geq 0\ (i \neq j),\ \ f_{ii}^n = 0, \forall\ i, j\ \in\ \mathcal{K} \tag{3.16e}$$

$$z_j - \alpha \cdot \sum_{n=1}^{N} v_n \cdot \sum_{i}^{K} f_{ij}^n = p_0(j),\ \forall j \in \mathcal{K} \tag{3.16f}$$

Lemma 9. *The change of variables $f_{ij}^n = z_i \cdot r_{ij}^n$, is a bijection (one-to-one mapping) between (z_i, r_{ij}^n) and (z_i, f_{ij}^n).*

Proof. This follows immediately, as we can readily obtain $r_{ij}^n = \frac{f_{ij}^n}{z_i}$ from $\{z_i, r_{ij}^n\}$. Note that, since $z_j = \sum_i f_{ij}^n + p_0(j)$, and $p_0(i)\ \in\ (0, 1)\ \forall\ i$, i.e. nonzero (see Def. 2), this forces $\mathbf{z} > \mathbf{0}$ and thus r_{ij}^n are always uniquely defined. □

Corollary. ***OP 4*** *can be solved efficiently as an LP.*

Proof. Equivalency due to Lemma 9. □

We have therefore transformed the nonconvex **OP 4** to a convex (LP) one **OP 6**, and can now solve it optimally.

3.3.3 A Myopic Approach

A natural way to tackle the **OP 4** is to try minimizing the cost of content retrieval in a single-content session (i.e., only one transition in the Markov chain). This is equivalent to minimizing the scalar quantity

$$\mathbf{p}_0^T \cdot (\alpha \cdot \sum_{n=1}^{N} v_n \cdot \mathbf{R}^n + (1 - \alpha) \cdot \mathbf{1}^T \cdot \mathbf{p}_0) \cdot \mathbf{c} \tag{3.17}$$

Ignoring the terms that do not depend on the control variables $\mathbf{R}^n$, yields the following.

OP 7 (Greedy Aware Recommendations)**.**

$$\underset{\mathbf{R}^1, .., \mathbf{R}^N}{\text{minimize}} \quad \mathbf{p}_0^T \cdot \Big(\sum_{n=1}^{N} v_n \cdot \mathbf{R}^n \Big) \cdot \mathbf{c}, \tag{3.18}$$

$$\text{subject to} \quad \textit{Eqs.}(3.11b, 3.11c, 3.11d, 3.11e) \tag{3.19}$$

Unlike the multi-step problem, this is already an LP, and can be solved directly without the earlier transformation steps. This solution of **OP 7** will serve in the upcoming results section as a baseline approach, to solving the hard basis problem **OP 4**. Interestingly, we consider as the baseline approach the solution of **OP 7** (we will call *Greedy* from now), resembles the policies proposed in [35, 33]. Although the algorithm of [35] targets a different context, i.e., the *joint* caching and single access content recommendation, the Greedy algorithm could be interpreted as applying the recommendation part of [35] for each user, along with a continuous relaxation of the control (recommendation) variables. In doing so, the recommendation problem is simply an LP of the type of Eq.(3.18), when the recommendations are allowed to be probabilistic. Due to this relaxation, the greedy algorithm is an upper bound for [35], looking at the recommendation problem *only*.

3.4 Results

3.4.1 Warm Up

In this section we evaluate the performance of the proposed algorithm and provide insights regarding the behavior of the network-friendly recommendations schemes. For a realistic evaluation, we use three collected datasets from video/audio services. Before diving into the details, we need to state the following

- *Performance metric*: *Cache Hit Rate (CHR)*, as computed by the objective of Eq.(3.10), here we will minimize the cache miss.
- *Relative Gain*: Computed as $\frac{CHR_{(\text{proposed})}-CHR_{(\text{baseline})}}{CHR_{(\text{baseline})}} \cdot 100\%$.
- $\mathbf{p}_0$: Drawn from Zipf [5] of parameter s.
- $\mathbf{v}$: Drawn from Zipf [15] of parameter β.
- α: Will vary from 0.7 to 0.8.
- $\mathbf{c}$: $c_i = 0$ for the C (cache capacity) most popular contents according to $\mathbf{p}_0$, and 1 to the rest.
- *Solving **OP 6, OP 7**:* carried out using IBM ILOG CPLEX in Python. We note that since CPLEX is designed to receive LPs in the standard form, we had to vectorize our matrices in order to bring the problem in the format $\underset{\mathbf{x}\geq 0, \mathbf{A}\cdot\mathbf{x}\leq\mathbf{b}}{\min} \{\mathbf{c}^T \cdot \mathbf{x}\}$ with linear and bound constraints over the variables. Regarding **OP 7**, it is easy to see that the problem's objective Eq.(3.18) decomposes into K independent minimization problems, of size NK each, as the *variables per content i are not coupled*.

Finally note that for the simulations in all Figures, we will quote the cache-hit rate *without* recommendations for reference, (i.e. storing the most popular contents that fit in the cache C, based on $\mathbf{p}_0$) and, which we denote as MPH (Most Popular Hit - No

Recommendations). This information along with the simulation parameters are included in Table 3.2.

3.4.2 Schemes we compare with

We refer to the proposed solution of **OP 6** as *Optimal*.

- *Greedy Aware:* We consider as baseline algorithm for network-friendly recommendations Moreover and as stated in the end of 3.3.1, an important baseline for us will be (**OP 7** [35]), which is a position-aware scheme, and takes into account known statistics about the user click rate with respect to the position that the recommendations appear, but does not take into account that requests are sequential.
- *CARS:* algorithm [44], a position-*unaware* scheme for sequential content requests proposed in, will serve as our second baseline. The CARS algorithm optimizes (with no guarantees) the recommendations for a user performing multiple sequential requests, but assumes that the user selects *uniformly* one of the recommendations regardless of the position they appear. Note that the objective of Eq.(4.11) assumes knowledge of $\mathbf{v}$, while the algorithm of [44],

Note on *CARS*. In our framework, this translates to solving **OP 4** for uniform $\mathbf{v}$. The algorithm will then return N *identical stochastic recommendation matrices*. Importantly, whichever $\mathbf{v}$ we choose, the parenthesis of the Eq.(4.11) will be $(\mathbf{I}-\alpha\cdot(v_1\cdot\mathbf{R}+..+v_N\cdot\mathbf{R})) = (\mathbf{I}-\alpha\cdot\mathbf{R})$. This explains why the hit rate of *CARS* in the plots, remains constant regardless of the click distribution $\mathbf{v}$. this force the algorithm to return the same recommendation matrix N times; this will then be normalized by the click probability $\frac{1}{N}$. As it is oblivious to $\mathbf{v}$, the performance of *CARS* remains constant regardless of the click distribution.

3.4.3 Datasets

Here is a list of the used datasets and a brief explanation on how we collected them.

YouTube FR. ($K = 1054$) We used the crawling methodology of [70] and collected a dataset from YouTube in France. We considered 11 of the most popular videos on a given day, and did a breadth-first-search (up to depth 2) on the lists of related videos (max 50 per video) offered by the YouTube API. We built the matrix $\mathbf{U} \in \{0, 1\}$ from the collected video relations.

last.fm. ($K = 757$) We considered a dataset from the last.fm database [71]. We applied the "getSimilar" method to the content IDs' to fill the entries of the matrix $\mathbf{U}$ with similarity scores in [0,1]. We then set scores above 0.1 to $u_{ij} = 1$ to obtain a dense $\mathbf{U}$ matrix.

MovieLens. ($K = 1066$) We consider the Movielens movies-rating dataset [72], containing 69162 ratings (0 to 5 stars) of 671 users for 9066 movies. We apply an item-to-item collaborative filtering (using 10 most similar items) to extract the missing user ratings, and then use the cosine distance ($\in [-1, 1]$) of each pair of contents based on their common ratings. We set $u_{ij} = 1$ for contents with cosine distance larger than 0.6.

Table 3.2 – Parameters of the simulation

	q %	$zipf(s)$	α	N	MPH %
MovieLens	80	0.8	0.7	2	23.26
YouTube FR	95	0.6	0.8	2	12.17
last.fm	80	0.6	0.7	3	11.74

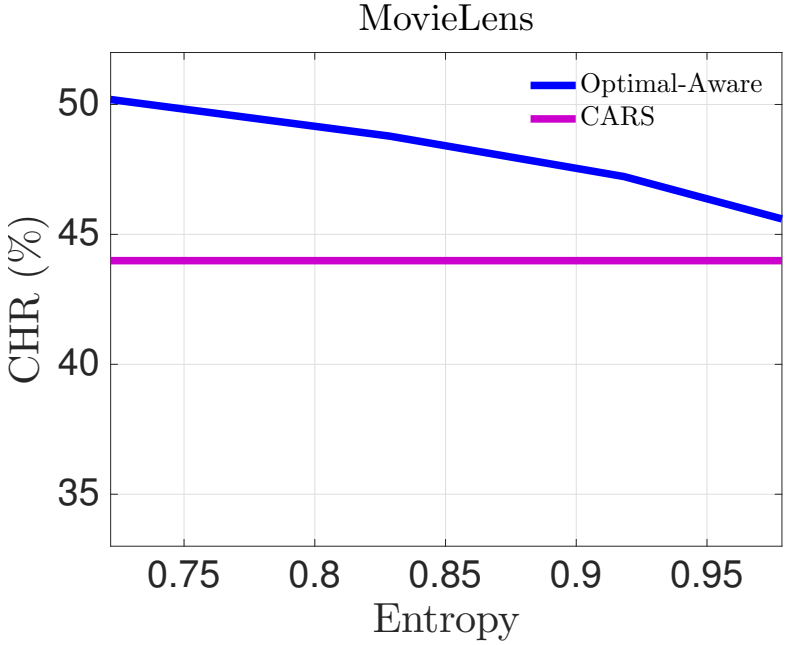

Figure 3.4 – Absolute Cache Hit Rate Performance vs $H_{\mathbf{v}}$ ($C/K \approx 1.00\%$) - MovieLens.

3.4.4 Results

***Optimal* vs *CARS*.** We initially focus on answering a basic question: *Is the non-uniformity of users' preferences to some positions helpful or harmful for a network friendly recommender?* In Figs. 3.4, 3.5, 3.6 (see Table 3.2 for simulation parameters), we assume behaviors of increasing entropy; starting from users that show preference on the higher positions of the list (low entropy), to users that select uniformly recommendations (maximum entropy). In our simulations, we have used a zipf distribution [15] over the N positions and by decreasing its exponent, the entropy on the x-axis is increased. As an example, in Fig. 3.4, lowest $H_{\mathbf{v}}$ corresponds to a vector of probabilities $\mathbf{v} = [0.8, 0.2]$ (recall that $N = 2$), while the highest one on the same plot to $\mathbf{v} = [0.58, 0.42]$.

Observation 1. Our first observation is that the lower the entropy, the higher *the optimal result*. In the extreme case where the $H_{\mathbf{v}} \to 0$ (virtually this would mean $N = 1$, the user clicks deterministically), the optimal hit rate becomes maximum. This can be validated in Fig. 3.11, where for increasing entropy the the hit rate decreases and its max is attained for $N = 1$.

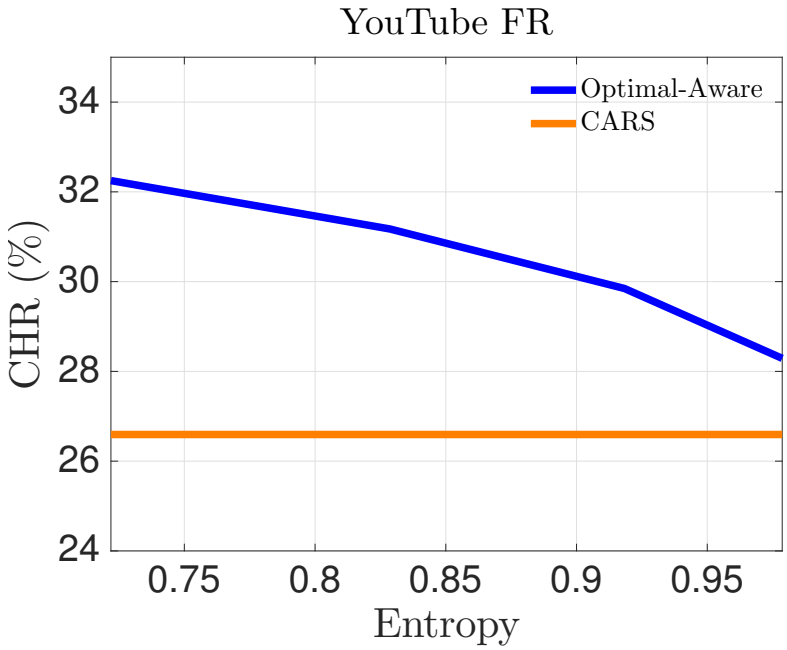

Figure 3.5 – Absolute Cache Hit Rate Performance vs $H_{\mathbf{v}}$ ($C/K \approx 1.00\%$) - Youtube France.

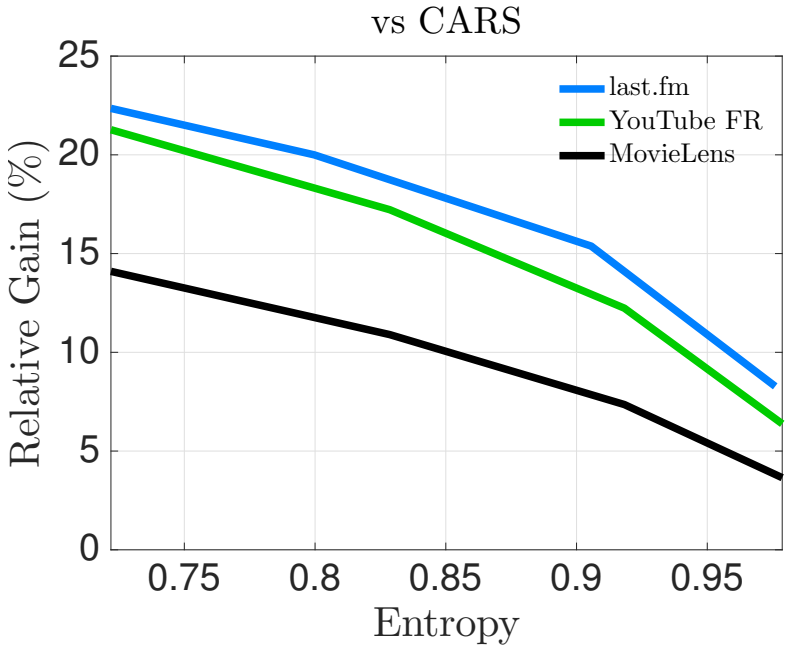

Figure 3.6 – Relative Cache Hit Rate Performance vs $H_{\mathbf{v}}$ ($C/K \approx 1.00\%$) - All Datasets.

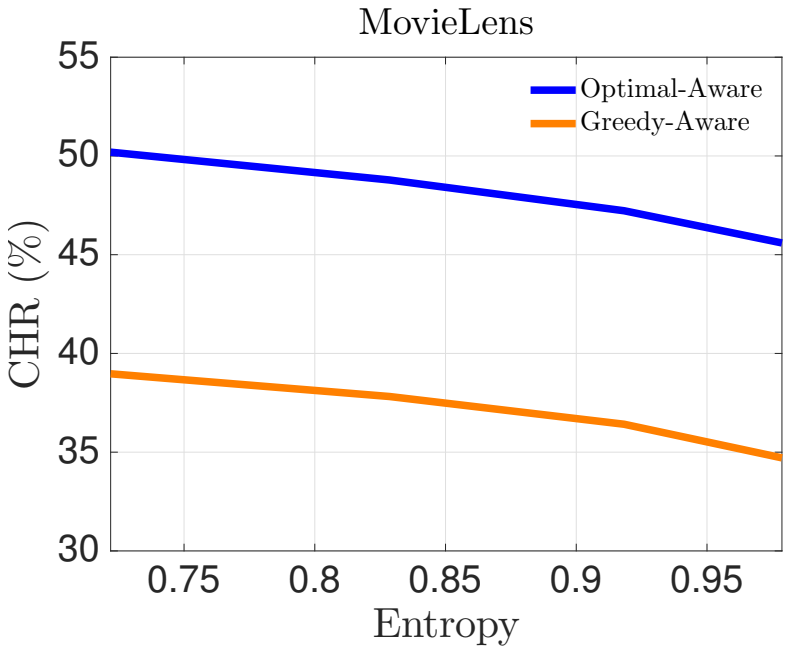

Figure 3.7 – Absolute Cache Hit Rate Performance vs $H_{\mathbf{v}}$ ($C/K \approx 1.00\%$) - MovieLens.

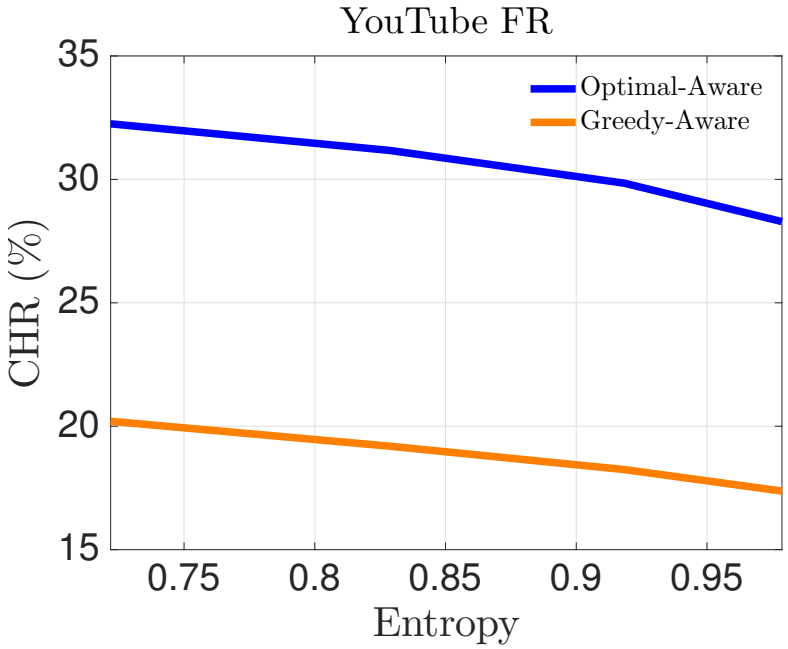

Figure 3.8 – Absolute Cache Hit Rate Performance $H_{\mathbf{v}}$ ($C/K \approx 1.00\%$) - YouTube France.

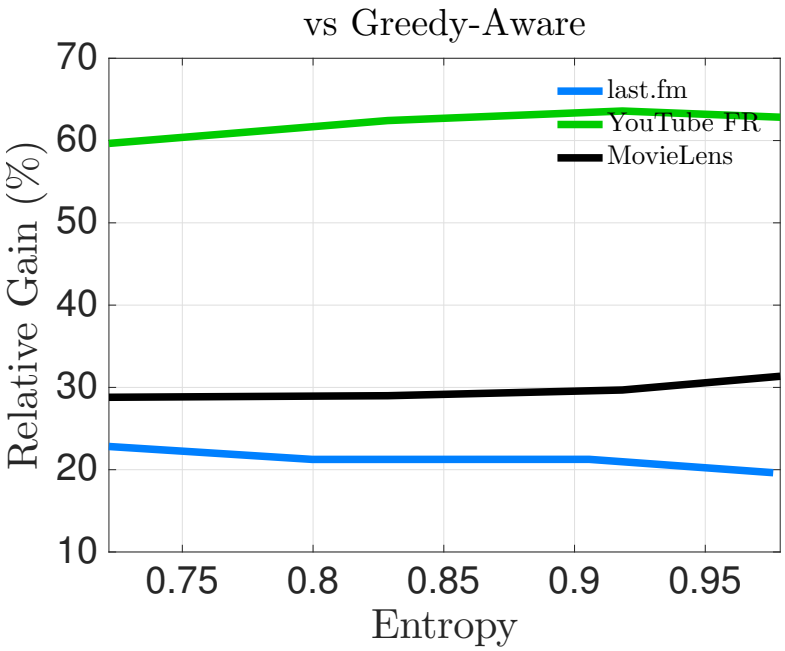

Figure 3.9 – Relative Cache Hit Rate Performance vs $H_{\mathbf{v}}$ ($C/K \approx 1.00\%$) - All Datasets.

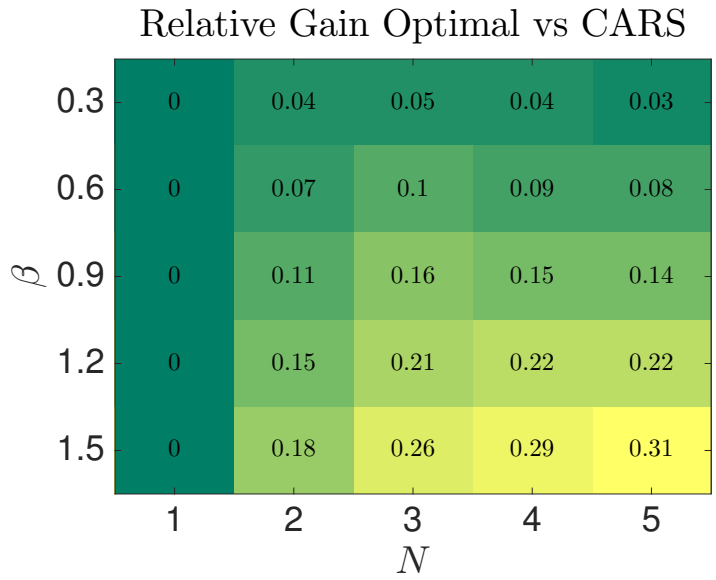

Figure 3.10 – Relative Gain vs (N, β), for $q = 80\%$, $K = 400$, $C/K \approx 1.00\%$, $\alpha = 0.7$

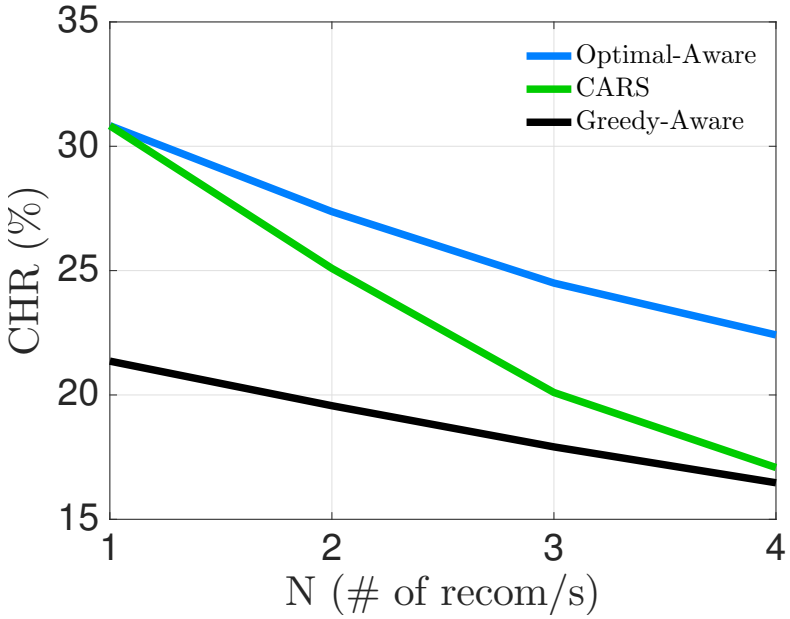

Figure 3.11 – Cache Hit Rate vs N $(C/K \approx 1.00\%,\ \alpha = 0.7)$

***Optimal* vs *Greedy*.** The second question we study is: *How would a simpler greedy/myopic, yet position-aware, algorithm fare against our proposed method?* Fundamentally, the *Greedy* algorithm solves a less constrained problem than **OP 4**, and is therefore a more lightweight option in terms of execution time. However, the merits of using the proposed optimal method are noticeable in Figs. 3.7, 3.8, 3.9 (parameters in Table 3.2). In all three datasets, we see an impressive improvement, between $20 - 60\%$.

Observation 2. The constant relative gain of the two *aware* algorithms hints that both, as the entropy increases, seem to do the right placement in the positions. However, as *Greedy* decides with a small horizon, it cannot build the correct *long* paths that lead to higher gains in the following requests (clicks) of the user.

Lastly, we investigate the sensitivity of the three methods against the number of recommendations (N).In Fig. 3.11, we present the *CHR* curves of all three schemes for increasing N, where we keep constant the distribution $\mathbf{v} \sim zipf(0.9)$. As expected, for $N = 1$ (e.g., YouTube autoplay scenario) *CARS* and the proposed scheme coincide, as there is no flexibility in having *only one* recommendation. However, as N increases, *CARS* and *Greedy* decay at a much faster pace than the proposed scheme, which is more resilient to the increase of N. This leads to the following observation.

Observation 3. For large N, *CARS* may offer the "correct" recommendations (cached or related or both), but it cannot place them in the right positions, as there are now too many available spots. In contrast, our algorithm *Optimal* recommends the "correct" contents, and places the recommendations in the "correct" positions. Fig. 3.10, strengthens even more the Observation 3; its key conclusion is that with high enough enough β (i.e. low $H_{\mathbf{v}}$) and more than 2 or 3 recommendations, while *CARS* aims to solve the multiple

access problem, its *position preference unawareness* leads to suboptimal recommendation placement, and thus severe drop of its *CHR* performance compared to the *Optimal*.

Chapter 4

The Random Session Case

4.1 Introduction

In this Chapter we dive even deeper to the NFR problem. In the previous two chapters, we formulated the problem borrowing tools from convex optimization and were able to

- Present a heuristic ADMM solution which performed quite well in practice.
- Transform the long session NFR problem to an LP with hard constraints on the user satisfaction.
- Incorporate the position preferences of the users using a basic stochastic model and solve this optimally as well through the LP.

As we have stressed in the beginning of this manuscript, the solution under investigation in the current thesis is essentially a software solution and as one it must be able to run in *reasonable computational time* with good performance guarantees. Although LP solvers such as ILOG CPLEX are extremely efficient and perform very well, they can still suffer when the number of variables and constraints becomes very large. We remind the reader that in the previous two chapters, the long session was approximated by an asymptotically infinite length session. In the MDP formulation we resolve this issue as we now optimize the cost for some session of average length $\bar{L}$ (measured in contents). In terms of computational efficiency, the MDP has two clear advantages, (a): by assuming some average session length, it avoids unecessary computations and (b): through the DP approach, it breaks down the problem in easier subproblems and thus we managed to

- Achieve improved runtimes while having ϵ-optimality guarantees.

More importantly, a major weakness of our previous works is that we considered users whose clickthrough probability is fixed and independent on the quality of the recommendation policy. However, using MDP as our main workhorse allows for some very interesting modeling extensions along this dimension. Thus, a major result of this work/chapter is that we could finally

- Explore the tradeoffs of the long session NFR under users who can be reactive to the quality they receive from the recommender. We did so by maintaining optimality guarantees of our policies.

4.2 Problem Setup

4.2.1 User Session and Interaction with the RS

A user launches some multimedia application (such as YouTube) and requests *sequentially* a random number of contents from its library $\mathcal{K}$. Importantly, such applications are now equipped with a RS which is responsible for helping the users discover new content. The user starts off by choosing a content out of $\mathbf{p}_0$, which expresses the pmf of his personal preferences over the library, and once he clicks on it, the RS proposes him N new contents which appear on the side of the screen (YouTube). This exact suggestion process happens every time he clicks on some content. Depending on his behavior and how he assesses the offered contents, the user might find the recommendation batch interesting and click on one of its contents or else use the text bar to look for a new one. He does so until eventually he quits the application which signifies the end of the session.

Session Length. The length of each session is equal to some random integer X which we model as a Geometric r.v parameterized by λ. This r.v is assumed to be independent from the RS actions. During the consumption of content i, the user has three possible actions. He might take one of the following actions

- **Recommendations:** Click on a content from the recommendation batch $\mathcal{N}_i$, that is related to content i.
- **Search/Text Bar:** Ignore the recommendations and choose some content out of his personal preferences, i.e., from $\mathbf{p}_0$.
- **Exit:** Quit the session with prob. $1 - \lambda$.

4.2.2 Recommender Knowledge about the User

Entertainment oriented applications are becoming increasingly effective due to the massive amount of data that is collected by interacting with users. Contents are requested, rated/liked and accepted or rejected as part of a recommendation batch; all these feedback measurements can only be an added to RS and its objectives.

According to the RS literature [73], user ratings can be used to infer the level of similarity between contents, e.g. item-item similarity. To formalize the notion of "related/similar" content for our framework, we use the following definition.

Definition 7 (Content Relations)**.** *We consider a fixed library of files $\mathcal{K} = \{1, 2, \ldots, K\}$. For every content i there is a set of similar contents $\mathcal{S}_i$, along with their similarity values $u_{ij} \in [0, 1]$, which populates the set $\mathcal{U}_i$. The values of $\mathcal{U}_i$ are not normalized per content.*

We can essentially imagine this structure as a graph of nodes/contents where the content i has outgoing edges for the contents in $\mathcal{S}_i$ with corresponding weights $\mathcal{U}_i$. What is more, the RS has at its disposal the aggregate measurements of user requests. This is another valuable piece of data which gives the system a global view of content popularity among all the users.

Definition 8 (User Preferences)**.** *We assume that the normalized popularity (requests) of content i represents the probability of the user to request i independently of the RS actions, e.g. through the text bar for YouTube. We encode these measurements in the normalized vector $\mathbf{p}_0(i) > 0\ \forall i \in \mathcal{K}$. Thus $\mathbf{p}_0$ represents the randomized choice rule (pmf) of the user over the library $\mathcal{K}$.*

Due to the sequential nature of user's request, we assume that the user satisfaction is measured per content. In other words, when the content ends, the recommendation list of content i collectively amasses some content relevance.

Definition 9 (User Satisfaction)**.** *For some set of recommended items $\mathcal{N}_i$ that is related to i, the user satisfaction at content i is perceived by the RS as*

$$q_i = \frac{\sum\limits_{l \in \mathcal{N}_i} u_{il}}{\sum\limits_{l \in \mathcal{U}_i(N)} u_{il}} \tag{4.1}$$

We call q_i^{max} the denominator of Eq.(4.1) (it is per content). Therefore, $q_i \in [q_i^{min}/q_i^{max}, 1]$ where q_i^{min} is the sum of the N lowest u_{ij} entries. The above quantities solely depend on the entries of $\mathcal{U}_i$ and N.

4.2.3 Cost-Aware Recommender over Network

We assume a generic network setup, where the average user interacts with one content provider by requesting contents in sequence.

Definition 10 (Content Cost)**.** *From the network's point of view, each content $i \in \mathcal{K}$ has a nonnegative network cost c_i, $\mathbf{c} = [c_1, \ldots, c_K]^T$ associated to its delivery to the user.*

The cost of delivering some content might depend on several factors such as its size (in MB), its routing expenses, its location on the network etc. Thus when a user has a session of M requests, his session *incurs* some cost on the network. Due to the impact of RS on user request, this sequence of costs $\{c(t)\}_{t=0}^{M}$ will also depend on the RS policy, where $c(t)$ takes values in $\mathbf{c}$. Thus, our main objective is to come up with recommendation policies $\boldsymbol{\pi}$ which promote low-cost contents and ultimately minimize the session's cost. With the letter $\boldsymbol{\pi}$ we denote the policy of the RS, more specifically $\boldsymbol{\pi} = [\mathbf{r}_1^T; \ldots; \mathbf{r}_K^T]$. in other words $\boldsymbol{\pi}$ is the $K \times K$ matrix that has the policies of all contents/states in a concatenated way ($\mathbf{r}_1$ is the first row, and so on).

$$\underset{\boldsymbol{\pi}}{\text{minimize}} \quad \left\{ \sum_{t=1}^{M} c(t) \right\} \tag{4.2}$$

Cost Examples. Allowing c_i to be a generic number, allows us to capture different scenarios such as

1. Caching: set $c_i = 0/1$ to cached/uncached contents respectively $\rightarrow$ maximizes cache hit.

2. CDN: set $c_i \in \mathbb{R}$ can capture the CDN case $\rightarrow$ minimizes delivery cost.

4.2.4 Policies

Our primary focus on this work is to come up with recommendation policies for long sessions. However, as pointed out in the previous two subsections, while our main objective is to minimize the cost of the user's session, the user satisfaction remains fundamentally an important dimension of the RS. Before going any further, we must first precisely define what we mean by policy. When the user visits file i, the RS can propose *any* N-tuple of unique contents (not including i). The set of all N-tuples w form the feasible set of actions related to content i, that is $\mathcal{A}_i$. Thus for every content i, the RS can select an action from a combinatorial space, i.e., the set $\mathcal{A}_i$ which consists of $\binom{K-1}{N}$ different possible recommendation batches of size N. As an example, for $K = 1000$ and $N = 3$ the RS has more than 165M batches at its disposal for each content.

Classes of Policies. In principle, there are two classes of policies, more specifically we have

- **Deterministic:** Only one N-tuple of contents can appear. For every i, there is one action a for which $\mu(a, i) = 1$.

- **Randomized:** At least two actions have $\mu(a, i) > 0$. This means that at every appearance of i, we might see a different N-tuple of contents.

Definition 11 (Batches Frequencies)**.** *Each recommendation batch $\omega \in \mathcal{A}_i$ is associated with some frequency of appearance $\mu_i(w)$. The sum of frequencies of all the batches related to i should sum up to 1.*

Therefore, for some content i, if all w have $\mu_i(w) = 0$ except for one which has $\mu_i(w) > 0$, the policy is deterministic and the RS always "plays" the same action. Whereas if at least two $\mu_i(w) > 0$ are nonzero, then the RS will propose the batches associated to the $\mu_i(w) > 0$ as frequently as dictated by $\mu_i(w)$. More importantly, Def. 11 prepares the ground for a different interpretation we can give to the RS actions.

Definition 12 (Object Frequencies)**.** *For every content/object j, we define its frequency of appearance in the recommendation batches related to i as*

$$r_{ij} = \sum_{w \in \mathcal{A}_i} \mu_i(w) \cdot \mathbf{1}_{\{j \in w\}} = \sum_{w \in \mathcal{A}_i : j \in w} \mu_i(w) \tag{4.3}$$

Where the above reads as "for all possible actions, sum how many time j is found in the batch w of the action set $\mathcal{A}_i$". Therefore, r_{ij} represents the probability of object j to appear in a recommendation batch of i. Naturally, it follows that for the vector $\mathbf{r}_i$ we have

$$\sum_{j=1}^{K} r_{ij} = \sum_{j=1}^{K} \sum_{w \in \mathcal{A}_i : j \in w} \mu_i(w) = \sum_{w \in \mathcal{A}_i : j \in w} \sum_{j=1}^{K} \mu_i(w) = N \ \forall i \in \mathcal{K}. \tag{4.4}$$

In the deterministic case, for every content i, there are exactly N $r_{ij} = 1$, and the other entries are zero. On the contrary, suppose a randomized policy for i, and the allowable actions are $\mathcal{A}_i = \{1, 2\}, \{1, 3\}$ associated with frequencies $[0.5, 0.5]$. This translates to $r_{i1} = 1.0, r_{i2} = 0.50, r_{i3} = 0.50$ and the remaining r_{ij} are zero.

To ease the notation for the later part of this chapter, for every content i, we have a vector $\mathbf{r}_i$ which includes all the content frequency for all the other contents. We also concatenate these vectors as $\boldsymbol{\pi} = [\mathbf{r}_1^T; ...; \mathbf{r}_K^T] \in \mathbb{R}^{K \times K}$ and refer to it as the RS policy.

4.3 Formulation

As we saw earlier, there can be many cases where a myopic policy fails to act in the optimal way. An appropriate mathematical tool that fits our framework and fills exactly this gap between look-ahead and myopic policies is the one of Markov Decision Problems (MDP). In this section we will focus on formulating the minimization of long user sessions in the expected cost sense. Along these lines, we mathematically formalize the physical entities mentioned in 4.2 using tools from the MDP toolbox.

4.3.1 Defining the MDP

Before going into detail on what we aim to optimize, we have to establish the state space, a basic transition model for our user and the costs of state transitions.

Markovian State Transitions. Importantly, as the recommendation policy is state dependent and the random jump statistics remain the same independently of the content, the next state the user visits is fully determined by the state he currently resides. Therefore the sequence of contents viewed by the user $(s_t)_{t \in \mathbb{N}}$ is a discrete time Markov process with state space $\mathcal{K}$. The following equation describes the state evolution in a probabilistic manner.

$$P\{i \to j\} = \alpha_{ij} \cdot r_{ij} + (1 - \sum_{l=1}^{K} a_{il} \cdot r_{il}) \cdot p_0(j) \tag{4.5}$$

Importantly, Eq.(4.5) expresses the average movement of the user as perceived by the RS. Following our discussion from Section 4.2, it becomes clear that with each state we associate the content costs of Def. 10.

Lemma 10. *The MDP defined by the $(\mathcal{K}, \mathcal{A}, P, \mathbf{c})$, (state/action space, dynamics, costs) is unichain, i.e., has only one class of states, for any policy if all $\alpha_{ij} < 1$ and $p_0^i > 0 \; \forall \; i \in \mathcal{K}$.*

Proof. Under some policy $\boldsymbol{\pi}$, we have a fixed state transition matrix $P^{\boldsymbol{\pi}}$. Note however that since $\alpha_{ij} < 1$, the random jump transition part of Eq.(4.5) *is always active*. In addition, as all entries of $\mathbf{p}_0$ are strictly positive, the user starting from any state, may end up in any state infinitely many times since there is a path from any state to all the states. This concludes the statement. □

4.3.2 Optimization Objective

Here we focus on formulating the optimization objective. As stated earlier, we consider some user that consumes a random number of contents before leaving the session. We denote the cost of requesting item s at time instant t as $c(s_t)$. Thus, the total cost induced by the requests of the user (in a randomly selected session) is $\sum_{t=1}^{L} c(s_t)$. The objective we wish to minimize is the *average* total cost.

Lemma 11. *The average total cost starting from state s can be cast an infinite horizon problem with discounts.*

$$\mathbf{E}_s\Big(\sum_{t=1}^{L} c(s_t)\Big) = \mathbf{E}_s\Big(\sum_{t=1}^{\infty} \lambda^{t-1} \cdot c(s_t)\Big) \tag{4.6}$$

where the subscript in the expectation stands for s as the starting state.

Proof. As stated in earlier, the probability that the session length L is equal to l is $\mathbf{P}(L = l) = (1-\lambda)\lambda^{l-1}$ for values of $l \in \mathbb{N}^+$. We essentially need to find the expectation of the sum of costs of *random* length. Therefore, we need to use the total expectation law as follows.

$$\mathbf{E}_s\Big(\sum_{t=1}^{L} c(s_t)\Big) = \sum_{l=1}^{\infty} \mathbf{P}(L = l) \cdot \mathbf{E}_s\Big(\sum_{t=1}^{l} c(s_t)\Big) \tag{4.7}$$

To ease the notation, we denote $\mathbf{E}_s\Big(\sum_{t=1}^{l} c(s_t)\Big) = E_l$. Then the RHS of Eq.(4.7) becomes.

$$\begin{aligned}
&(1-\lambda)E_1 + (1-\lambda)\lambda E_2 + (1-\lambda)\lambda^2 E_3 + ... \\
&\quad = E_1 - \lambda E_1 + \lambda E_2 - \lambda^2 E_2 + \lambda E_3 + ... \\
&\quad = E_1 + \lambda(E_2 - E_1) + \lambda^2(E_3 - E_2) + ...
\end{aligned} \tag{4.8}$$

For the difference of $E_{l+1} - E_l$ we can use linearity as the length is not random anymore.

$$E_{l+1} - E_l = \mathbf{E}_s\Big(\sum_{t=0}^{l+1} c(s_t)\Big) - \mathbf{E}_s\Big(\sum_{t=0}^{l} c(s_t)\Big)$$

$$= \mathbf{E}_s\Big(\sum_{t=0}^{l+1} c(s_t) - \sum_{t=0}^{l} c(s_t)\Big) = \mathbf{E}_s\Big(c(s_{t+1})\Big) \tag{4.9}$$

Therefore, we started from the LHS of Eq.(4.7) and were able to describe it as in Eq.(4.8). Given these, and using the relation of Eq.(4.9), we have the LHS of Eq.(4.7) written as

$$\mathbf{E}_s\Big(\sum_{t=1}^{L} c(s_t)\Big) = \mathbf{E}_s\Big(c(s_1)\Big) + \lambda \mathbf{E}_s\Big(c(s_2)\Big) + \dots \tag{4.10}$$

Using linearity for the constant λ (to plug it back in the expectation), the last relation coincides with the relation of Lemma 11. □

Constraints. The feasible space will be shaped by the set of constraints we impose on the policy. The policy of the RS has to obey four specifications.

- Fixed budget, suggest exactly N items.
- Guarantee a level of recommendation quality.
- r_{ij} is a probability.
- Do not recommend yourself.

Our objective is to have the minimum expected total cost from *every* state s possible under the following constraints.

OP 8.

$$\underset{\mathbf{r}_1,..,\mathbf{r}_K}{\text{minimize}} \left\{\mathbf{E}_s\Big(\sum_{t=1}^{\infty} \lambda^{t-1} c(s_t)\Big)\right\} \tag{4.11}$$

$$\text{subject to} \quad \sum_{j=1}^{K} r_{ij} = N \ \forall i \in \mathcal{K}, \tag{4.12}$$

$$\sum_{j=1}^{K} r_{ij} \cdot u_{ij} \geq Q_{MIN} \cdot q_i^{max} \ \forall i \in \mathcal{K}, \tag{4.13}$$

$$0 \leq r_{ij} \leq 1, \ \forall i \neq j \in \mathcal{K} \tag{4.14}$$

$$r_{ii} = 0, \ \forall i \in \mathcal{K}. \tag{4.15}$$

To keep a compact notation, we will denote the feasible set of policies of content i as $\mathcal{R}_i = \{r_{ij} : (4.12), (4.13), (4.14), (4.15)\}$.

4.3.3 Optimality Principle

In the MDP framework, the cost of starting from a *given* state s is simply called $v(s)$, the value of the state. Our aim is to find the optimal value function $\mathbf{v}^* = [v(1)^*, ..., v(K)^*]$. Essentially **OP 8** has optimal substructure and therefore the Bellman Principle is a necessary condition that has to hold.

In general, given an MDP and a *specific policy*, we get in return a *unique* Markov Chain whose dynamics are governed by the transition matrix $P^{\boldsymbol{\pi}}$. To this end, we bring into the picture a vital quantity in the MDP framework, the value function (a vector $\in \mathbb{R}^K$) which is defined as follows

Definition 13 (Value Function)**.** *The expected total cost of starting from state $s \in \mathcal{K}$, with a given λ and when a specific policy $\boldsymbol{\pi}$ is applied, is defined as*

$$v_{\lambda}^{\boldsymbol{\pi}}(s) = \lim_{L \to \infty} \mathbf{E}^{\boldsymbol{\pi}} \left(\sum_{t=1}^{L} \lambda^{t-1} c(s_t) \mid s_0 = s \right) \tag{4.16}$$

where the superscript $\boldsymbol{\pi}$ stands for some fixed policy and the s is the starting state. The above limit exists for values of $\lambda \in [0, 1)$ and if the costs are bounded from below. In our case all costs are nonnegative real numbers, thus the limit exists. For the remainder of this work we will denote this limit as $v^{\boldsymbol{\pi}}(s)$.

For a stationary policy $\boldsymbol{\pi}$, the value of a certain state S is recursively related to the values of all the states $s^{'} \in \mathcal{K}$ with the following relation [74].

$$v^{\boldsymbol{\pi}}(s) = c_s + \lambda \sum_{s^{'} \in \mathcal{K}} P\{s \to s^{'}\} \cdot v^{\boldsymbol{\pi}}(s^{'}) \tag{4.17}$$

where $P\{s \to s^{'}\}$ is a function of the policy $\boldsymbol{\pi}$ and is taken directly from Eq.(4.5).

Definition 14 (Optimal Policy)**.** *A policy $\boldsymbol{\pi}^*$ is called optimal if it achieves the minimum value function, i.e. the the vector $\mathbf{v}^* = [v^*(1), ..., v^*(K)]^T$. For this we have*

$$v^*(s) \leq v^{\mathbf{r}'}(s) \ \forall \ s \in \mathcal{K} \ \textit{and} \ \forall \ \boldsymbol{\pi}' \in \mathcal{R}. \tag{4.18}$$

Bellman Optimality Equations. It becomes evident that the quest to optimal policies coincides with the quest of finding the optimal value function $\mathbf{v}^*$. For optimality, $\mathbf{v}^*$ has to obey the following set of K equations, one per state.

$$v^*(i) = c_i + \lambda \min_{\mathbf{r}_i \in \mathcal{R}_i} \left\{ \sum_{j=1}^{K} P_{ij}(\mathbf{r}_i) \cdot v^*(j) \right\} \ \forall i \in \mathcal{K}. \tag{4.19}$$

where $P_{ij}(\mathbf{r}_i)$ is the probability defined in Eq.(4.5). Note that we use i or s to denote the state/content interchangeably.

In the remainder of this section we discuss two known alternatives from the rich MDP literature [74, 75] to compute the NFR policy.

Value Iteration (VI). The VI is based on iteratively applying the Bellman optimality operator onto an initial arbitrary $\mathbf{v}_0$ until some convergence criterion has been satisfied. The algorithmic steps are described in Algorithm 4.

Algorithm 4 Gauss-Seidel VI

1: $v(i) = v_{\text{old}}(i) = 0 \ \forall \ i \in \mathcal{K}$
2: **repeat**
3: **for** $i \in \mathcal{K}$ **do**
4: $v(i) \leftarrow v_{\text{old}}(i)$
5: $v(i) = c_i + \lambda \min_{\mathbf{r}_i \in \mathcal{R}_i} \left\{ \sum_{j=1}^{K} P_{ij}(\mathbf{r}_i) \cdot v(j) \right\}$
6: **end for**
7: **until** $||\mathbf{v} - \mathbf{v}_{\text{old}}||_\infty < \epsilon$
8: return $\mathbf{r}_i = \text{argmin}\{v_i\} \ \forall i \in \mathcal{K}$

Lemma 12. *The Gauss-Seidel VI returns the ϵ-value function of the NFR problem. In fact it does so with convergence that is linear to some $\gamma \leq \lambda$.*

The proof of that statement can be found in [74].

Remark 3. *Notice that the probability α_{ij} could be defined as any function of the policy, i.e., $\alpha_{ij} = f(\mathbf{r}_i, u_{ij})$.*

The second alternative that can be used is the Policy Iteration (PI) algorithm [74]. PI consists of two basic steps

1. Policy evaluation: Given a policy $\boldsymbol{\pi}$ find the $\mathbf{v}^{\boldsymbol{\pi}}$.
2. Policy improvement: Given a $\mathbf{v}^{\boldsymbol{\pi}}$, sweep over the states and greedily optimize.

There is a fundamental tradeoff between the two approaches. As we saw, VI performs value improvements until the error between consecutive state sweeps becomes arbitrarily small. On the bright side, the inner loops do not cost much. However this means that during the last state sweeps, the value may not have converged but the policy could remain unchanged. On the other hand, one full iteration of PI is way more costly, as it consists of a policy evaluation (worst case $O(K^3)$, matrix inversion) and a greedy improvement. Yet interestingly, PI terminates when policy remains unchanged, which in practice happens rapidly in most cases. The basic VI enjoys a number of complexity and optimality guarantees, however there are splitting methods such as Gauss-Seidel (which is the one we implemented) that perform much better in practice (while enjoying the same performance guarantees).

The second approach to solving the same problem is the algorithm known as policy iteration (PI). It is essentially a two-stage algorithm that consists of

1. Policy evaluation: Given a policy $\mathbf{r}$ find the $\mathbf{v}^{\mathbf{r}}$.
2. Policy improvement: Given a $\mathbf{v}^{\mathbf{r}}$, sweep over the states and optimize.

The steps of the algorithm are described as follows.

Algorithm 5 PI

Input: $\alpha, \lambda, \epsilon, Q^{\%}_{MIN}, \mathbf{c}, \mathbf{U}, \mathbf{p}_0$ ▷ (system parameters)

Converged $\leftarrow$ False
Assign $v_0(S) = 0 \; \forall \; S \in \mathcal{K}$
Assign $\mathbf{r}^0$ an arbitrary policy
$t \leftarrow 0$
repeat
  $k \leftarrow 0$
  repeat ▷ Policy Evaluation
    $k \leftarrow k + 1$
    for $i \in \mathcal{K}$ **do**
      $v_{k+1}(i) = c_i + \lambda \left\{ \sum_{j=1}^{K} P_{ij}(\mathbf{r}_i) \cdot v(j) \right\}$
    end for
  until $||\mathbf{v}_k - \mathbf{v}_{k-1}||_2^2 < \epsilon$
  $t \leftarrow t + 1$
  for $i \in \mathcal{K}$ **do**
    $\mathbf{r}_i^t = \text{argmin}_{\mathbf{r}_i \in \mathcal{R}_i} \left\{ \sum_{j=1}^{K} P_{ij}(\mathbf{r}_i) \cdot v(j) \right\}$
  end for
until Converged == $(\mathbf{r}^t == \mathbf{r}^{t-1})$
$\mathbf{R} = \texttt{concatenate}(\mathbf{r}_1^t, ...\mathbf{r}_K^t)$
return $\mathbf{R}$

4.3.4 Versatility of look-ahead policies through λ

Interestingly, the MDP has the upside of being quite flexible on the range of problems it can tackle. Observe that λ plays essentially the role of "predicted average length of user session". Existing works that focus on longer user sessions such as [44, 76], implicitly assume an *infinitely long* session, which is of course unrealistic, whereas in our framework λ can simply act as some tuning parameter that comes from measured user data. We will investigate three cases.

Case: $\lambda \to 0$. This coincides with $v(s) = \mathbf{E}_s(0^0 c_{s_1} + 0^1 c_{s_1} + \dots) = \mathbf{E}_s(c_{s_1})$ (assuming that $0^0 = 1$) i.e., the user is *already* at some some file s_0 and does *exactly* one more request and the system incurs the losses c_{s_1}. Substituting at the Bellman Equations $\lambda = 0$ yields

$$v^*(i) = c(i) + \cancelto{0}{\lambda} \min_{\mathbf{r}_i \in \mathcal{R}_i} \left\{ \sum_{j=1}^{K} P_{ij}(\mathbf{r}_i) \cdot v^*(j) \right\} \tag{4.20}$$

Essentially $v^*(i) = c(i)\ \forall\ i \in \mathcal{K}$. Therefore, solving the above problem using ViT, translates to solving K independent optimization problems using $v^*(i) = c(i)$, exactly once (only one iteration will be needed).

Case: $\lambda \to 1/m$ $(m = 2, 3, \dots)$**.** This case captures a user who from past statistics, has been measured to have m sequential requests on average after his first request.

Case: $\lambda \to 1$**.** For the Infinite Horizon model, the value $v(s)$ diverges for $\lambda = 1$ as this value of λ means we are adding infinitely many costs. However, using [Puterman Cor 8.2.5], we can find the average long term cost to be $\lim_{\lambda\to 1}(1-\lambda)v^{\lambda}(s)$ (where λ is an index). The latter limit exists for unichain MDPs (as in our case) and it expresses the time-average long term cost.

In practice, since the $Geo(\lambda)$ mean length is $1/(1-\lambda)$, one could use the empirical average of the user session to find an estimate $\hat{\lambda}$ and then the RS should solve the MDP with this parameter value, to derive appropriate recommendations.

4.4 Quality Driven Users: Some Use Cases

Fundamentally, solving the NFR can be seen as the generalized task of recommending batches of objects that are associated with some network cost and some user utility. The goal is to optimally balance two competing interests, which are

1. Miss rate minimization (from the operators viewpoint)
2. User's satisfaction (by default the goal of RS)

From the modeling/optimization's perspective, all existing works have approached the problem by considering cost minimization as the criterion while encompassing the user satisfaction as an explicit constraint [44], or by controlling some distortion metric [35], which are basically modeling approaches along the same direction.

4.4.1 User Behavior: Model 1

Given that the RS explicitly constrains its average recommendation quality to exceed some Q_{MIN}, the user will click on any of the contents in the recommendation batch with some fixed click-through rate $\frac{\alpha}{N}$. Such a user can be captured by setting $\alpha_{ij} = \frac{\alpha}{N}\ \forall i, j \in \mathcal{K}$.

The model implies that the user *does not assess his* q_i (received recommendation quality) since his click-through rate for any item j is fixed and independent of the policy or the content similarity. The user transition is depicted below as

$$P\{i \to j\} = \frac{\alpha}{N} \cdot r_{ij} + (1-\alpha) \cdot p_0(j) \tag{4.21}$$

where notice that $\sum_{l=1}^{K} \frac{\alpha}{N} \cdot r_{ij} = \alpha$. Thus the total average rejection rate with which the user ignores the recommendations is some $\alpha \in [0, 1]$. Using Eq.(4.21), the Bellman equations become

$$v^*(i) = c_i + \bar{v} + \lambda \frac{\alpha}{N} \min_{\mathbf{r}_i \in \mathcal{R}_i} \left\{ \sum_{j=1}^{K} r_{ij} \cdot v(j)^* \right\} \forall i \in \mathcal{K}. \tag{4.22}$$

where $\bar{v} = \lambda(1-\alpha) \sum_{j=1}^{K} p_0(j) \cdot v^*(j)$.

Lemma 13. *The minimization step of Algorithm 4, has been reduced to solving an LP.*

Proof. Observe in Eq.(4.22) that the objective in the minimization is linear in the variable r_{ij}. Moreover, the solution space is convex as it is an intersection of a linear inequality, a linear equality and bound constraints (see Eqs.(4.12 - 4.15). □

Notice that solving the MDP for *Model 1*, returns a policy in the class of MR.

Moreover, the Bellman equations reveal some structural characteristics of the optimal policy.

Property 1. *When $u_{ij} \in [0,1]$, and $Q_{MIN} = 1$, the optimal policy for content i is unique and is $\mathbf{r}_i = \{r_{ij} : r_{ij} = 1$ if $j \in \mathcal{U}_i(N), r_{ij} = 0$ if $j \notin \mathcal{U}_i(N)\}$*

Proof. For Q_{MIN}, the rhs of Eq.(4.13) becomes $\sum_{l \in \mathcal{U}_i(N)} u_{il}$. Assume that the optimal policy for content i is to assign $r_{ij} = 1$ to contents in $\mathcal{U}_i(N-1)$, $r_{ij} = x > 0$ to some content $j \notin \mathcal{U}_i(N)$ and $r_{im} = 1 - x$ to the least related item $m \in \mathcal{U}_i(N)$. Then the constraint Eq.(4.13) reads

$$\begin{aligned} \sum_{l \in \mathcal{U}_i(N-1)} u_{il} \; + (1-x)u_{im} + x u_{ij} &\geq \sum_{l \in \mathcal{U}_i(N-1)} u_{il} \; + u_{im} \\ (u_{ij} - u_{im}) \cdot x &\geq 0 \end{aligned} \tag{4.23}$$

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

Property 2. *In the case where $Q_{MIN} = 0$, the optimal policy is to assign $r_{ij} = 1$ to the N lowest cost contents excluding of course one self due to (4.15). Otherwise $r_{ij} = 0$.*

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

$$v^*(i) = c(i) + \bar{v} + \lambda \big(\sum_{j=1: j \neq i}^{N} v^*(j) + v^*(N+1) \big) \tag{4.24}$$

Table 4.1 – Summary of Models

	Model 1	Model 2	Model 3
user follows recommender	constant	$f(\mathbf{r}_i, \mathbf{u}_i)$	$f(\mathbf{r}_i, \mathbf{u}_i)$
user content selection	$1/N$	$1/N$	$f(\mathbf{r}_i, \mathbf{u}_i)$
user leaves session	λ	λ	λ
user satisfaction	constraint	α	α + content selection

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

$$v^*(i) = c(i) + \bar{v} + \lambda \sum_{j=1}^{N} v^*(j) \tag{4.25}$$

For the pairs though, there are three cases (1): $1 \leq i, j \leq N$ and $i < j$, from the sorting we have

$$v^*(i) - v^*(j) < 0 \Rightarrow (c(i) - c(j)) + \lambda(v^*(j) - v^*(i)) < 0 \tag{4.26}$$

where for the second term above, there are $N - 1$ terms that have cancelled out. Notice that due to the ordering, $\lambda(v^*(j) - v^*(i)) > 0$, so it must hold that $c(i) - c(j) < 0$ in order that the above expression to have a negative sign. Observe that for case (2): $1 \leq i \leq N$ and $N < j$, the exact same as above will hold. Then for case (3): $N < i < j$ we have

$$v^*(i) - v^*(j) < 0 \Leftrightarrow \tag{4.27}$$

$$(c(i) + \lambda \sum_{j=1}^{N} v^*(j)) - (c(j) + \lambda \sum_{j=1}^{N} v^*(j)) < 0 \tag{4.28}$$

which immediately states that if $v^*(i) - v^*(j) < 0$ then $c(i) - c(j) < 0$. Therefore, the optimal costs-to-go $v^*(i)$ are ordered exactly as the immediate costs $c(i)$, which concludes that for content i, choosing the N lowest costs excluding content i is optimal. □

4.4.2 User Behavior: Model 2

The major difference of this approach is that we *incorporate* the quality of the recommendations in the probability of transition. Hence, it is no longer necessary to explicitly constrain the recommendation quality (like in Model 1), thus $Q_{MIN} = 0$. Two observations on the modeling side of things can be made as a result.

1. If we *fail* to deliver good recommendations, the user click-through rate will be reduced, and the network cost will go up. Thus there is *no incentive* to make low cost and bad recommendations.

2. Instead of assuming an arbitrary Q_{MIN}, we allow the dataset $(\mathcal{U},\ \mathbf{p}_0)$ to decide how much quality is indeed the best you have to offer.

We use $\alpha_{ij} = \frac{f(\mathbf{u}_i,\mathbf{r}_i)}{N} = \frac{\mathbf{u}_i^T \cdot \mathbf{r}_i}{q_i^{max} \cdot N}$; this implies that the click-through probability depends on the policy and how good it is, and that the contents in the recommendation batch are clicked uniformly by the user.

Observation. The quantity $\frac{\mathbf{u}_i^T \cdot \mathbf{r}_i}{q_i^{max}}$ depends on the object appearance frequencies r_{ij}. Consider the case where the policy $\mathbf{r}_i$ is randomized. Then, for consecutive realizations of the policy, the user may view different recommendation batches, which could have a different aggregate similarity. Hence, when in content i, the user will see in $t = 1$ the batch w_1 and so on, which in the limit will be equal to

$$\lim_{T\to\infty} \frac{1}{T} \sum_{t=1}^{T} \sum_{m\in w_t} u_{im} = \sum_{w\in\mathcal{A}_i} \mu_i(w) \sum_{m\in w} u_{im} = \sum_{j=1}^{K} u_{ij} r_{ij} \tag{4.29}$$

Therefore, the scalar quantity $\mathbf{u}_i^T \cdot \mathbf{r}_i$ normalized by q_i^{max}, i.e., the average recommendation quality is user's click-through rate. Consequently, using Eq.(4.5) and α_{ij} as defined above, the transition probability from content i to j is written as

$$P\{i \to j\} = \frac{\mathbf{u}_i^T \cdot \mathbf{r}_i}{q_i^{max} \cdot N} \cdot r_{ij} + (1 - \frac{\mathbf{u}_i^T \cdot \mathbf{r}_i}{q_i^{max}}) \cdot p_0^j \tag{4.30}$$

Remark 4. *In the case of MD policies, the transition of Eq.(4.30) describes some user who reacts to the instantaneous recommendation quality he receives. For MD policies, the user can observe only one batch w, which is of course associated with a fixed $\sum_{m\in w} u_{im}$.*

In every iteration of VI or PI algorithm, we have to minimize the following function $g(\mathbf{r}_i)$.

$$g(\mathbf{r}_i) = \sum_{j=1}^{K} \left(\frac{\mathbf{u}_i^T \cdot \mathbf{r}_i}{q_i^{max} \cdot N} \cdot r_{ij} + (1 - \frac{\mathbf{u}_i^T \cdot \mathbf{r}_i}{q_i^{max}}) \cdot p_0(j) \right) \cdot v(j) \tag{4.31}$$

Since we aim on minimizing g, we can remove additive and multiplicative constants.

$$\begin{aligned} g(\mathbf{r}_i) = (\mathbf{u}_i^T \cdot \mathbf{r}_i) \cdot (\frac{\mathbf{v}^T \cdot \mathbf{r}_i}{N}) - (\mathbf{u}_i^T \cdot \mathbf{r}_i) \cdot (\mathbf{p}_0^T \cdot \mathbf{v}) = \\ \mathbf{r}_i^T \cdot \frac{1}{N} \mathbf{u}_i \cdot \mathbf{v}^T \cdot \mathbf{r}_i - (\mathbf{p}_0^T \cdot \mathbf{v}) \cdot \mathbf{u}_i^T \cdot \mathbf{r}_i \end{aligned} \tag{4.32}$$

Our optimization problem can be then cast as a QP

OP 9.

$$\underset{\mathbf{r}_i}{\text{minimize}} \left\{ \mathbf{r}_i^T \cdot \frac{1}{N} \mathbf{u}_i \cdot \mathbf{v}^T \cdot \mathbf{r}_i - (\mathbf{p}_0^T \cdot \mathbf{v}) \cdot \mathbf{u}_i^T \cdot \mathbf{r}_i \right\} \tag{4.33}$$

$$\text{subject to} \quad \mathbf{r}_i \in \mathcal{R}_i \tag{4.34}$$

Lemma 14. ***OP 9*** *is a nonconvex optimization problem.*

Proof. To show convexity of a generic Quadratic Program (QP), we investigate whether the $\mathbf{u}_i \cdot \mathbf{v}^T$ is positive semidefinite (PSD). Clearly as an outer product of two arbitrary vectors $\mathbf{u}_i$ and $\mathbf{v}$, there is no guarantee about the PSD property of the matrix $\mathbf{u}_i \cdot \mathbf{v}^T$ and thus the convexity of **OP 9**. Note that $\mathbf{v}$ changes in every iteration. Just to be symmetric, we would need $\mathbf{Q}$ to be an outer product of a vector with itself. In our case $\mathbf{u}_i \cdot \mathbf{v}^T$ is an outer product of two different nonnegative vectors, therefore $\mathbf{u}_i \cdot \mathbf{v}^T$ cannot even be symmetric, and therefore the function is nonconvex. □

The above lemma stands as an obstacle towards finding an optimal MR policy ($r_{ij} \in [0, 1]$). However, if we restrict our attention to MD policies, i.e. $r_{ij} \in \{0, 1\}$, we know that we can *always* find an optimal policy. Using standard VI algorithm, we can enumerate all possible actions and pick the best one, i.e., the one with the minimum objective.

A Speed-Up for the Caching Problem

We discuss a practical heuristic algorithm that solves the NFR problem for a user who behaves according to Model 2. We propose that in each minimization step of the VI algorithm, it is only sufficient to search for either the cached or the $\mathcal{U}_i$ contents and nowhere else. This direction comes quite natural as our goal is to achieve many hits of the cache by increasing the user's probability to click on the content we suggest. Thus, we only search for contents that can contribute on one of the two dimensions or both. Along these lines we propose an approximate VI algorithm; "Approximate" in the sense that each minimization of the VI is not carried out exactly as we do not look over all the possible actions.

The difference in the implementation is the preprocessing we need to do before running the VI algorithm. Earlier, if we looked for an MD policy related to content i, we would simply enumerate **all** possible N-tuples that do not include i and pick the best one, whereas now we narrow down the solution space and look only for the union of the related and cached items.

1. (Initialization): For each i, we find the set of items $\mathcal{SR}^i$, the set of the strongly related items, i.e. the ones for which we have $u_{ij} >$ thresh. Then form the union $\mathcal{F}_i = \mathcal{C} \cup \mathcal{SR}^i$

2. (Line 5 of VI): To minimize, iterate over all $\binom{|\mathcal{F}_i|}{N}$ combinations and pick the best.

Discussion on the Heuristic. Essentially, the set of cached contents $\mathcal{C}$ has a cardinality which is orders of magnitude smaller than K. In addition, our focus is on contents that are closely related to i (ones with a significant u_{ij} value, which are usually not too many). Therefore thresh. filters out these entries (the "not very" related items) and then we end up with a search/solution space of cardinality $\binom{|\mathcal{F}_i|}{N}$, which boosts dramatically the runtime of our algorithm without loosing in performance in most cases.

Finally, note that the suboptimality of picking contents only out of the set $\mathcal{F}_i$ heavily depends on the parameter λ and on the graph properties of $\mathcal{U}$. To be suboptimal under

the approximate version of VI for the caching problem, we would need a user with high value of λ (actually $\lambda \to 1$), then it is possible that optimal actions involve contents $\notin \mathcal{F}_i$.

What should we expect? An example

Here we will try to give some intuition on how the RS policy will look like for model 2 and how much user satisfaction it will achieve for some very short session. For the sake of discussion suppose $u_{ij} \in \{0, 1\}$, $c_i \in \{0, 1\}$ and $\mathbf{p}_0 \sim Uni(1, K)$, where $Uni(1, K)$ denotes the uniform distribution over the events $1, \ldots, K$ and K is large; the latter implies $p_0(i) \approx 0 \; \forall \; i$.

For a user who consumes always two contents in sequence; first is the one he finds from the search bar and the second one which will come either from the recommendations *if the user satisfaction is quite large* or from the search bar *if the user satisfaction is quite low*. We remind the reader at this point, that for model 2 the user satisfaction is directly the clickthrough probability (α). Suppose also that the RS suggests $N = 2$ contents as an example and let us say that the user currently views content i. Then for content i, we investigate two extreme cases:

1. There are $N = 2$ items that are both cached AND related.
2. There is no item that is cached AND related.

The hit probability of the next step (the one that can be influenced by the RS) is computed as

$$P_{hit}(L, M) = \frac{L}{N} \cdot \frac{M}{N} + (1 - \frac{L}{N}) \cdot \cancelto{\approx 0}{\sum_{i \in \mathcal{C}} p_0(i)} \tag{4.35}$$

where L is the number of related recommended and M of the cached and recommended contents in the recommendation batch. In the case (1): it is obvious that we should simply recommend the two cached and related items, then we would have $L = M = N$, in which case the hit probability in the next step would be 1. In this case, the quality satisfaction is also 1.

However in case (2), it is no longer obvious what is the best action. What should we recommend? (a): two cached items (b): one related and one cached or (c): two related items. Interestingly, when there is no overlap of the two categories we have $N = L + M$, which means that we can substitute in Eq.(4.35) $L = N - M$, and for some fixed N the hit probability is

$$P_{hit}(M) = \frac{N - M}{N} \cdot \frac{M}{N} \tag{4.36}$$

which is a function of one variable and we can easily find its maximum value. In the example we described, it is easy to see that we need to find the M that maximizes

$P_{hit} = f(M) = M(1 - 0.5M)$. It is easy to see that $\max_M f(M) = 0.25$ for $M = 1$. Hence an unexpected and nontrivial conclusion is that in the extreme case where the RS has no overlap in the cached and related items and the user is likely to go everywhere equally likely from the text bar, it is myopically optimal to give 50-50 items, that is $\frac{N}{2}$ should be cached (to have cache hits), and the rest should be related (in order to increase his click-through).

4.4.3 User Behavior: Model 3

We consider a third model of a user which has a clear distinction from the previous two. His click-through on the recommended items is *no longer* uniform. We assume that the user *can asses* and is driven by the relevance of the objects that appear on his suggestions' list relatively to the best possible action, i.e., the higher the u_{ij} of the content, the more likely the user will click on the item. Using $\alpha_{ij} = \frac{\sum_{j=1}^{K} r_{ij}u_{ij}}{q_i^{max}} \cdot \frac{u_{ij}}{\sum_{j=1}^{K} r_{ij}u_{ij}} = \frac{u_{ij}}{q_i^{max}}$, the probability of transition from content i to j is written as

$$P\{i \to j\} = \frac{u_{ij}}{q_i^{max}} \cdot r_{ij} + (1 - \sum_{m=1}^{K} \frac{u_{im}}{q_i^{max}} \cdot r_{im}) \cdot p_0(j) \tag{4.37}$$

Essentially, the user decides to click on recommended content with $\frac{\sum_{j=1}^{K} r_{ij}u_{ij}}{q_i^{max}}$ and then given the fact he decides to click on one of them, he chooses the content j with probability $\frac{u_{ij}}{\sum_{j=1}^{K} r_{ij}u_{ij}}$. However as we can see these terms cross out (for nonzero $\sum_{j=1}^{K} r_{ij}u_{ij}$) and finally we arrive at Eq.(4.37). As in *Model 2*, the quantity $1 - \frac{\mathbf{u}_i^T \cdot \mathbf{r}_i}{q_i^{max}}$ expresses the recommendation average rejection rate; whereas for $\mathbf{r}_i$ discrete, it reduces to $1 - \frac{\sum_{m \in w} u_{im}}{\sum_{m \in \mathcal{U}_i(N)} u_{im}}$, thus it expresses the *actual* dislike of the user towards irrelevant recommendations.

Our aim is to come up with optimal policies for the NFR problem for some user such as the one of Model 3. In doing so, we need to better understand the optimization problem that arises during the runtime of VI (or PI). Substituting the expression for P_{ij}, Eq.(4.37) in the Bellman Equations gives rise to the following

$$\begin{aligned} g(\mathbf{r}_i) = \sum_{j=1}^{K} \left(\frac{u_{ij}}{q_i^{max}} r_{ij} + (1 - \sum_{m=1}^{K} \frac{u_{im}}{q_i^{max}} r_{im}) p_0(j) \right) v_j = \\ \sum_{j=1}^{K} r_{ij}u_{ij}v_j - \sum_{j=1}^{K} \sum_{m=1}^{K} \left(r_{im}u_{im} \right) p_0^j v_j = \\ \mathbf{r}_i^T \cdot (\mathbf{u}_i \odot \mathbf{v}) - \mathbf{r}_i^T \cdot \mathbf{u}_i \cdot \mathbf{p}_0^T \cdot \mathbf{v} = \\ \mathbf{r}_i^T \cdot (\mathbf{u}_i \odot \mathbf{v} - \mathbf{u}_i \cdot \mathbf{p}_0^T \cdot \mathbf{v}) \end{aligned} \tag{4.38}$$

Thus, the optimization problem we have at hand is the following

OP 10.

$$\underset{\mathbf{r}_i}{\text{minimize}} \left\{ (\mathbf{u}_i \odot \mathbf{v} - \mathbf{u}_i \cdot \mathbf{p}_0^T \cdot \mathbf{v})^T \cdot \mathbf{r}_i \right\} \tag{4.39}$$

$$\text{subject to} \quad \mathbf{r}_i \in \mathcal{R}_i \tag{4.40}$$

Lemma 15. *OP 10 is an LP which can be solved optimally in* $\mathcal{O}(K \log(K))$

Proof. The calculations to Eq.(4.38) reveal a linear objective. Moreover, the feasible solution set has one linear equality Eq.(4.12), bound constraints of Eq.(4.14) and since $Q_{MIN} = 0$, the constraint of Eq.(4.13) is inactive. However observe that all the weights on the linear equality 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 at the objective. Generalizing that, 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 10** can be reduced into a sorting problem which is known to need $\mathcal{O}(K \log(K))$ steps. □

Corollary. *The optimal MR policy is a MD policy.*

4.5 Results

In this section, we primarily aim in assessing the performance of the proposed algorithms, and in doing so, we will try to highlight key conclusions in order to gain a better understanding of the NFR problem.

4.5.1 Metrics of Interest

Execution Time. An important aspect of the algorithms we propose is their complexity. For that reason, we perform an experiment with which we investigate the fundamental tradeoff between objective accuracy and execution time. These experiments were carried out using a portable MacBook Air with (1) RAM: 8 GB 1600 MHz DDR3 and (2) Processor: 1,6 GHz Dual-Core Intel Core i5.

Cache Hit Rate (CHR). Although our formulation allows us to quantify any generic cost metric, in our simulations we focus on the caching case. Therefore, we assume a library of size K and a set of contents $\mathcal{C}$ that are locally stored. Our framework captures this scenario if we set the cost of content i as $c_i = 1 - I_{\mathcal{C}}(i)$, where $I_{\mathcal{C}}$ is the unit index function of set $\mathcal{C}$. In the plots however, we depict the *hit ratio*.

How we evaluate the CHR. Therefore, after having computed the recommendation policy, we simulate our user (depending on his model of behavior) by generating 5000 requests (large session), and averaging the results over 5 realizations. As we have assumed that there are different user behaviors, in each plot we will specify under which behavior (model 1, model 2, model 3) the simulation is carried out.

Parameters and Input. In our simulations, we set $C = 0.01 \cdot K$ or less (C is the number of cached items) and as a caching decision, we place in the cache the C most

popular items according to $\mathbf{p}_0$. In addition, content requests from the search bar are distributed according to a zipf distribution $\mathbf{p}_0 = zipf(K, s)$ (K is the library size and s the zipf exponent) which is considered as constant and known [5]. Finally, when we evaluate the user of *Model 1*, we assume that the user click-through α (in *Model 1* α is the same for all contents) is equal to the Q_{MIN} we guarantee to the user.

4.5.2 What we Evaluate

For the results part, we will take into consideration four distinct approaches.

- $\boldsymbol{\pi}_1$: A Randomized Myopic Policy assuming the user behavior of *Model 1*.
- $\boldsymbol{\pi}_2$: A Randomized Look-Ahead Policy assuming the user behavior of *Model 1*.
- $\boldsymbol{\pi}_3$: A Deterministic Look-Ahead Policy assuming the user behavior of *Model 2*.
- $\boldsymbol{\pi}_4$: A Deterministic Look-Ahead Policy assuming the user behavior of *Model 3*.

Note that due to the exploding complexity of computing the optimal policy of the *Model 2* case, we approximate it by employing the sub-optimal policy $\boldsymbol{\pi}_3$ which searches over the reduced solution space of only the related or cached items.

4.5.3 Traces

last.fm. ($K = 757$) We consider a dataset from the last.fm database [71]. We applied the "getSimilar" method to the content IDs' to fill the entries of the matrix $\mathbf{U}$ with similarity scores in [0,1]. Then, we (a): find the largest component of the graph, then (b): set scores above 0.1 to $u_{ij} = 1$ to obtain a dense binary $\mathbf{U}$ matrix and finally (c): remove rows and columns with less than three related items.

YouTube ($K = 2098$) We also consider the YouTube dataset found in [77]. We are mainly interested in acquiring the list of related items (i.e., the true recommendation list as YouTube would present it) of each item. To this end, we (a): get the largest component of the corpus and build a graph of 2098 nodes (contents) and (b): set $u_{ij} = \text{rand}(0.5, 1)$ if there is a link from $i \to j$. Note that since u_{ij} represents the prob. the user will click to a content if offered, (b) helps to achieve a more realistic structure of graph $\mathcal{U}$.

Synthetic We build binary ($u_{ij} \in \{0, 1\}$) and continuous ($u_{ij} \in \{0, 1\}$) synthetic content relation graphs $\mathcal{U}$ as follows. We decide the size of the corpus K, and then choose randomly for each content, how many related items it will have, to be a number drawn uniformly from $\{0, ..., 0.1 \cdot K\}$. Then if we want u_{ij} to be continuous we set $u_{ij} = \text{rand}(0.5, 1)$.

4.5.4 Results

Runtime-Accuracy tradeoff for $\boldsymbol{\pi}_2$. Initially, we investigate the tradeoff of optimality accuracy and runtime when considering *different* algorithmic approaches policies for the *Model 1*. Essentially, for $\lambda \to 1$, that is exactly the model of Chapter 2, i.e., clickthrough prob. is fixed (and equal to some α) and the user clicks uniformly among

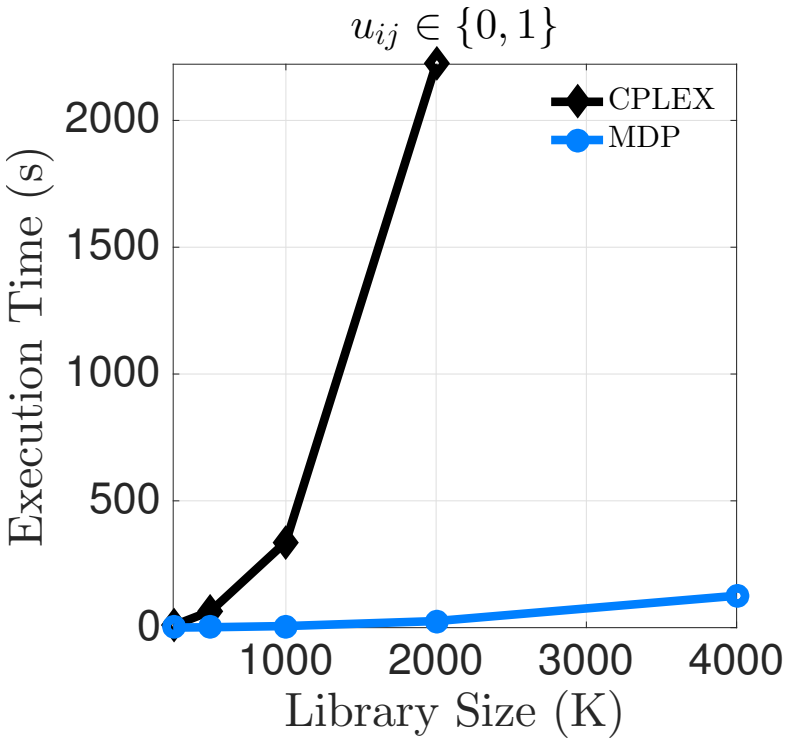

Figure 4.1 – Execution Time: MDP (Policy Iteration) vs CPLEX ($u_{ij} \in \{0, 1\}$)

the N recommendations. Thus the State-of-the-Art solution for that problem would be to solve **OP 5** using the LP solver of IBM ILOG CPLEX. We then use the MDP solver for some value of λ close to 1 and compare the actual hit rate result and the execution times for increasing library size. When (a) $u_{ij} \in \{0, 1\}$, where we use the PI algorithm with the $\mathcal{O}(\cdot)$ inner minimization loops complexity and (b): $u_{ij} \in [0, 1]$, where the inner minimization loops (which are much smaller LPs) are solved through CPLEX.

Observation #1 In this simulation we are not interested in presenting the absolute CHR values of the policies produced by CPLEX and $\boldsymbol{\pi}_2$. It is a given that CPLEX returns the optimal policy for the *Model 1* user, however we need to highlight the dramatic savings in execution time of $\boldsymbol{\pi}_2$ in both cases (see Fig(4.1, 4.2)) which is accompanied

Table 4.2 – Accuracy in CHR performance of CPLEX and PI

	$\text{CHR}_{\text{CPLEX}} - \text{CHR}_{\text{PI}}$	
$K \downarrow$	Fig. 4.1	Fig. 4.2
250	0.62e-4	0.0031
500	0.49e-4	0.0027
1000	0.12e-4	0.0047
2000	0.58e-4	0.0031
4000	CPLEX crashed	

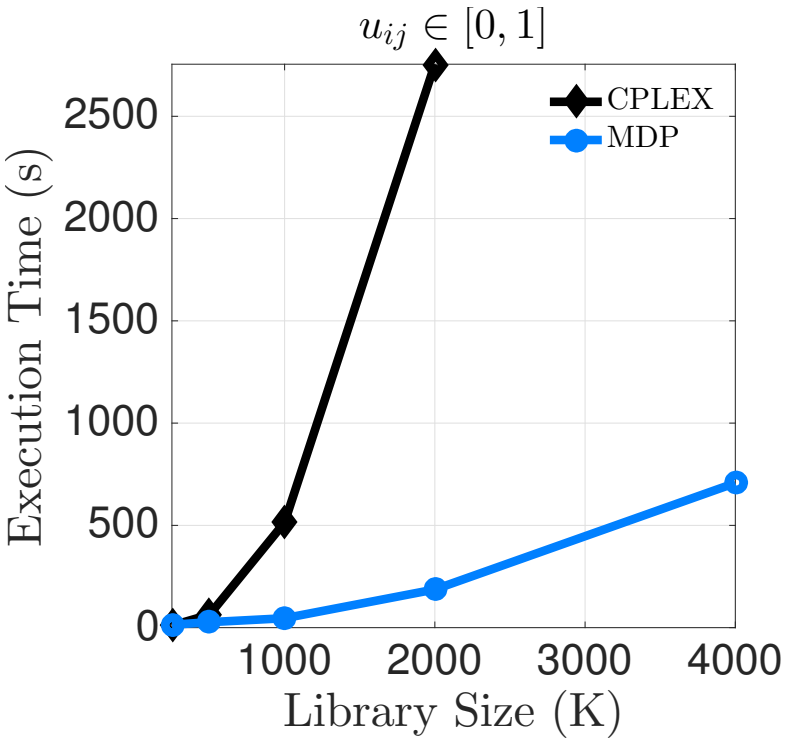

Figure 4.2 – Execution Time: MDP (Policy Iteration) vs CPLEX ($u_{ij} \in [0,1]$)

by really high accuracy of the performance. Observe that in Table 4.2, we report the performance difference, i.e., $CHR_{LP} - CHR_{MDP}$ and as we can see in both columns the difference is negligible. This suggests that the ϵ-optimal $\boldsymbol{\pi}_1$ is in reality really close to the absolute optimal one. Moreover, as $\boldsymbol{\pi}_2$ can run in reasonable time for large scenarios, this suggests that the algorithm could be of practical use even for real time scenarios. It is very important to highlight here, that when we execute the VI algorithm, it is not necessary to use $\lambda \to 1$, as this can slow down the runtime of the algorithm. However even for lower values of $\lambda \approx 0.8$, the algorithm performs almost equally well with LP which essentially optimizes over an actual infinitely long request session.

Observation #2 One can easily see that the runtime saving in the $u_{ij} \in \{0,1\}$ case is even more impressive. That is due to the fact that the LP problem created in the minimization step of the VI (or PI) algorithm is a "degenerate" LP, which can be solved by direct assignment. More specifically, in the minimization step one needs to assign full budget, that is $r_{ij} = 1$ to the contents with lowest v that also have $u_{ij} = 1$ until the quality (user satisfaction constraint) is satisfied. Then assign the remaining budget (recommendations) to the lowest contents (irrespectively of their u_{ij} value). This is considerably faster than solving an LP in every iteration (even if it is a small LP).

$\boldsymbol{\pi}_3$ vs $\boldsymbol{\pi}_2$ vs $\boldsymbol{\pi}_1$ (Tested under *Model 2* User Behavior). A question we wish to address in this paragraph is "What quality should we offer the user at each content in order to minimize the network cost?", which is a highly non-trivial question. In other

words, we ask how much quality should we give the user in order that (1): he clicks frequently on recommended contents (the ones we have control over) and (b): he clicks on low-cost items at the same time as frequently as possible. One could say that in a way, we attempt to find the best operating point of Q_{MIN} (and therefore α).

Thus, for comparison purposes we increase Q_{MIN} (see Eq.(4.13)) and as a result the user $\alpha \uparrow$, remember we have assumed that $Q_{MIN} = \alpha$, and we measure the performance in CHR. Therefore, for $\boldsymbol{\pi}_1$ (myopic) and $\boldsymbol{\pi}_2$ (look-ahead), we find many different operating points, and we see usually the best value is achieve close to $Q_{MIN} = 1/2$, which hints that probably when at content i, the recommender does *not* have many related and cached items to suggest. On the other hand, $\boldsymbol{\pi}_3$ finds exactly one point (see where the red lines intersect in Fig. 4.3, 4.4), which obviously coincides with the max hit rate (y-axis). That is somethig we should have expected as $\boldsymbol{\pi}_3$ is designed to solve exactly that problem. In addition, in the x-axis we show $\bar{\alpha} = \frac{1}{K}\sum_{i=1}^{K}\sum_{j=1}^{K}\frac{r_{ij}\cdot u_{ij}}{q^i_{max}}$ which is a bit more than 0.5 in both cases. That is the average of all the α values (one for each content) the recommender decided. Observe that the hit rate achieved by $\boldsymbol{\pi}_3$ is essentially a bit higher than the other two policies, and this due to the freedom we gave to the optimizer to decide different values of α for every content; in contrast, for the other two policies, the α was the same for every content and was equal to the value of Q_{MIN}.

Observation #3. What we need to understand regarding policy $\boldsymbol{\pi}_3$ is that by *jointly* deciding the quality (for each content), along with which contents j to offer when in i, in a way it is able to use the inherent structure of the dataset, that being the content relation graph $\mathcal{U}$ and the $\mathbf{p}_0$. Essentially, this policy is allowed to balance the "usefulness" of a content (how many related items it has) in combination with its popularity through the search bar (p_0^j). We see that on average, from the $N = 2$, we have to give one good and one low cost recommendation. This hints that for most contents i, the relevant ones, *are also not* cached. As a result, the RS resorts to recommending one good content as to keep α_i high and then offers one cached content because as *assumed* for *Model 2*, the user will click on any of the N recommended contents equally likely. Thus roughly, when following such a policy, in half the times the RS receives a hit.

$\boldsymbol{\pi}_4$ vs $\boldsymbol{\pi}_3$ (Tested under *Model 3* Behavior). We aim to present the merits of policy $\boldsymbol{\pi}_4$ when applied to the most realistic user scenario, i.e., *Model 3*. As $\boldsymbol{\pi}_4$ is an ϵ-optimal policy for the *Model 3* behavior, we expect that it performs better than $\boldsymbol{\pi}_3$, however we want to see "how much better and how much faster it actually is". To quantify these metrics, we present Table 4.3 where for different datasets, we see the performance and runtime improvement offered against $\boldsymbol{\pi}_3$. There, it is evident that $\boldsymbol{\pi}_4$ can outperform policy $\boldsymbol{\pi}_3$ in terms of hit rate, as $\boldsymbol{\pi}_3$ does not take into account that our user selects contents according to how relevant they are individually (proportionally to u_{ij}). Then, in terms of runtime there is an obvious gap, which is explained by the complexity of the minimization loops of $\boldsymbol{\pi}_4$ and $\boldsymbol{\pi}_3$. In the minimization step, for $\boldsymbol{\pi}_3$, we need to enumerate the objective of many actions whereas for $\boldsymbol{\pi}_3$ the minimization is simply a sorting operation.

In addition, an interesting plot is the one depicted at Fig 4.5. On the x-axis we see the possible values of $\alpha_i = \sum_{j=1}^{K}\frac{r_{ij}\cdot u_{ij}}{q^i_{max}} \in [0,1]$ for all K contents, and on the y-axis we see

Table 4.3 – Comparing for $\boldsymbol{\pi}_4$ and $\boldsymbol{\pi}_3$

	Cache Hit Rate (%)		Execution Time (s)	
dataset	$\boldsymbol{\pi}_4$	$\boldsymbol{\pi}_3$	$\boldsymbol{\pi}_4$	$\boldsymbol{\pi}_3$
Synthetic (1K)	51.86	34.43	30.0	5464.5
lastfm	40.17	20.00	14	1136

the frequency that these α values appear (after the optimization, when they are actually decided). For a dense dataset (with many nonzero u_{ij} entries), policy $\boldsymbol{\pi}_4$ indeed decides quite high value of α (it invests a lot on related contents as to increase user satisfaction); as we can see that 700 contents out of 1000 have an α value of ≈ 0.8 which is quite high. However, observe that since the user model $\boldsymbol{\pi}_4$ has a more demanding content selection process (user select proportionally to u_{ij}) the RS, which explains why $\boldsymbol{\pi}_4$ assigns more contents with higher α value.

A Comment. As we have allowed both policies $\boldsymbol{\pi}_3$ and $\boldsymbol{\pi}_4$ to schedule/decide *how much quality* should be given per content, it is useful to understand whether those policies indeed decide high or low user satisfaction. The latter is of great interest since *how these policies behave* heavily depends on the dataset (inputs). To get a better grasp on that, suppose a very extreme scenario where the cached items have no items as related. Then even the policies with the long vision would fail to find paths with high recommendation quality contents that will eventually lead to high hit rates. Hence, as the policies ultimate goal is to achieve high hit rates, the only way to achieve that would be through the search bar. What we mean by that? If the pmf $\mathbf{p}_0$ is very skewed, the cached content have very high probability to be requested by the users *if they ignore the recommendations*. Thus the RS may choose to recommend irrelevant content, so the user decides to ignore it, and finally click on contents with high $\mathbf{p}_0$ (which are the cached ones). However, this is a an extreme case and we explain it in order to gain a better understanding on how this policy behaves.

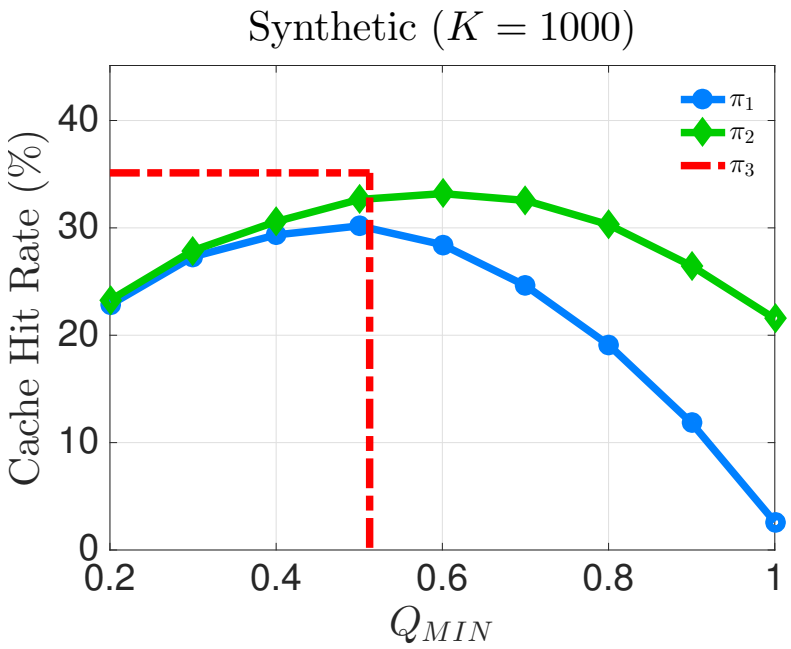

Figure 4.3 – CHR performance of $\boldsymbol{\pi}_1$, $\boldsymbol{\pi}_2$ for increasing Q_{MIN}. The optimal point CHR and q_i found by $\boldsymbol{\pi}_3$ is the point of intersection of the red lines - synthetic 1K

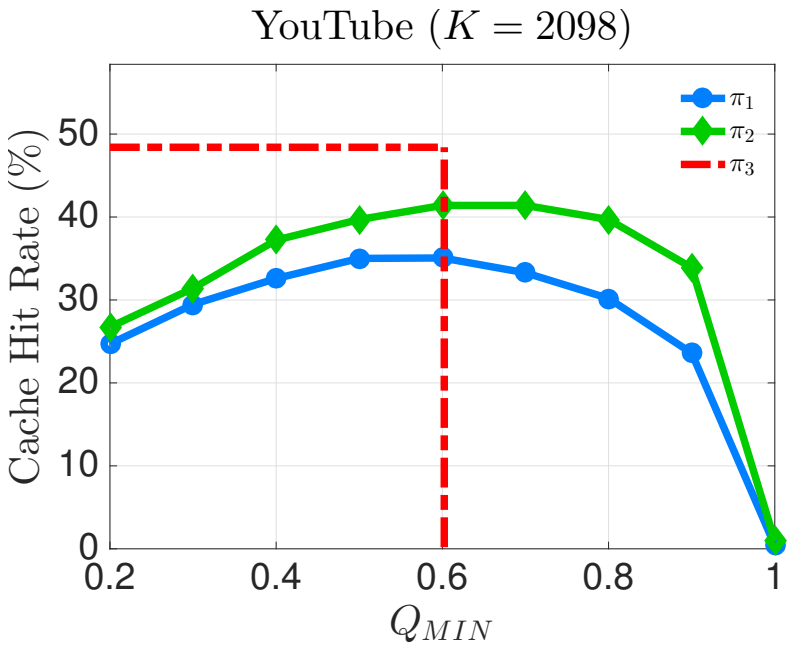

Figure 4.4 – CHR performance of $\boldsymbol{\pi}_1$, $\boldsymbol{\pi}_2$ for increasing Q_{MIN}. The optimal point CHR and q_i found by $\boldsymbol{\pi}_3$ is the point of intersection of the red lines - YouTube

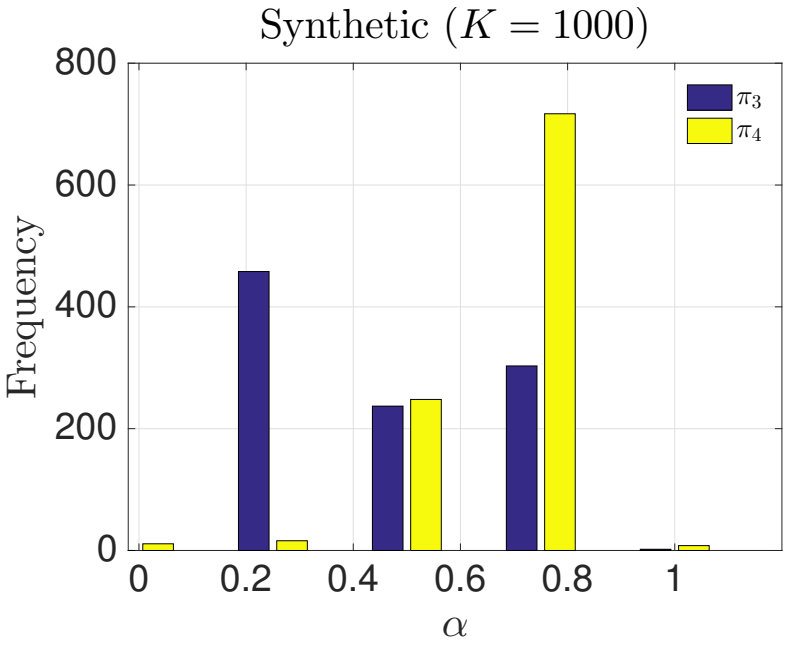

Figure 4.5 – Histogram of the variable α_i - Synthetic 1K

Chapter 5

Soft Cache Hits

5.1 Introduction

This chapter constitutes a standalone part of the thesis, in the sense that it does not add new results on the basic long session NFR (like the previous chapters) but rather deals with a whole different problem in the caching and recommendations interplay. One may reasonably wonder that since the recommender has such "nudging" capabilities to favor cached/low-cost content, the question becomes whether the caching algorithm *should also change* to come up with a different/better allocation.

In this chapter, we only tackle a preliminary version of this question, in a somewhat standalone manner. This is both due to timing (we started looking first at the caching side of the problem) and the high complexity of the caching problem if one assumes dynamic caching and sequential content access. Hence, this chapter is meant to serve as a first step towards treating the joint problem of both caching and recommendation, a topic which goes beyond the scope of this thesis. Some initial investigations into the joint problem suggest that this problem is rather hard, with no obvious way of optimality or approximation guarantees yet, for the more complex, markovian content access setup. However, we managed to face some versions of the joint problem (more specifically two use cases described in the sequel of this chapter) where the caching allocation no longer considers only the probability mass of the contents (in the IRM sense), but also takes into account the power of every content in the recommendation sense. In other words, the caching decision is heavily affected by whether a content is useful when recommending contents, i.e., if many contents have it as "a related content" then its chances of being requested are increased, and thus a caching decision that includes it has better hit rate performance.

IMPORTANT: Dr. Pavlos Sermpezis and Prof. Spyropoulos started working on this problem when Pavlos was with FORTH (Greece) as a PostDoc. When I started the PhD, they already had some preliminary results and thought enough over the problem. The JSAC paper, after which this chapter has been structured, was led by Pavlos and not myself. My contribution on that paper/chapter was twofold. The paper essentially has two main optimization problems. I came up with model and proved the properties for

the optimization problem described in section 5.5, and additionally I was also responsible for part of the code that produced the results of the simulations. We include the full work here for completeness, as in the case where I included only my theoretical part, it would not make much sense as a reading material.

5.1.1 Background and Motivation

Mobile edge caching has been identified as one of the five most disruptive enablers for 5G networks [78], both to reduce content access latency and to alleviate backhaul congestion. However, the number of required storage points in future cellular networks will be orders of magnitude more than in traditional CDNs [4] (e.g., 100s or 1000s of small cells (SCs) corresponding to an area covered by a single CDN server today). As a result, the storage space per local edge cache must be significantly smaller to keep costs reasonable. Even if we considered a small subset of the entire Internet catalogue, e.g., a typical torrent catalogue (1.5 PB) or the Netflix catalogue (3 PB), edge cache hit ratio would still be low even with a relatively skewed popularity distribution and more than 1 TB of local storage [6, 7].

Additional caching gains have been sought by researchers, increasing the "effective" cache size visible to each user. This could be achieved by: (a) *Coverage overlaps*, where each user is in the range of multiple cells, thus having access to the aggregate storage capacity of these cells, as in the femto-caching framework [22, 48]. (b) *Coded caching*, where collocated users overhearing the same broadcast channel may benefit from cached content in other users' caches [79]. (c) *Delayed content access*, where a user might wait up to a TTL for her request, during which time more than one cache (fixed [23] or mobile [24, 26, 27, 25]) can be encountered. While each of these ideas can theoretically increase the cache hit ratio (sometimes significantly), the actual practical gains might not suffice by themselves, e.g., due to high enough cell density required for (a), sub-packetization complexity in (b), and imposed delays in (c).

To get around this seeming impasse, we propose to *move away from trying to satisfy every possible user request, and instead try to satisfy the user*. In an Internet which is becoming increasingly entertainment-oriented, one can make the following observations: (a) a user's content requests are increasingly influenced by various recommendation systems (YouTube, Netflix, Spotify, or even Social Networks) [15]; (b) some related contents (e.g. two recent NBA games, two funny cat clips) might have similar utility for a user; in micro-economic terms, these are often called *substitute goods*; we will use the terms *alternative*, *related*, and *substitute* content inter-changeably.

5.1.2 Soft Cache Hits: Idea and Implications

Based on these observations, we envision a system where "soft cache hits" can be leveraged to improve caching performance. As one example, consider the following, for the case of YouTube (or, any similar service). If a user requests a content, e.g., by typing on the YouTube search bar, and the content is not available in the local cache(s), then a local app proxy located near the cache and having knowledge of the cached contents (e.g. a YouTube recommender code running at a Multi-access Edge Computing (MEC)

server [80]), could recommend a set of *related contents* that *are* also locally available. If the user prefers or accepts (under some incentives; see below) one of these contents, instead of the one she initially typed/requested, a soft cache hit (SCH) occurs, and an expensive remote access is avoided. We will use the term *soft cache hit* to describe such scenarios.

Of course, appropriate incentives would be needed to nudge a user towards substitute content. While perhaps a somewhat radical concept in today's ecosystem, we believe there are a number of scenarios where soft cache hits are worth considering, as *they could benefit both the user and the operator*. (i) A *cache-aware recommendation* plugin to an existing application could, for example, let a user know that accessing the original content X is only possible at low quality and might be choppy, freeze, etc., due to congestion, while related contents $A, B, C, ...$ could be streamed at high resolution, as shown in Fig. 5.1. (ii) Alternatively, the operator could activate this system only during predicted congestion periods, while giving some incentives to users to accept the alternative contents during that time (e.g., *zero-rating* services [81, 82]). (iii) In some cases, the operator might even "enforce" an alternative (but related) content (see Fig. 5.2), e.g., offering low rate plans with higher data quotas with the agreement that, during congestion, only locally cached content can be served.

While, in the above cases, a potential unwillingness or utility loss needs to be counterbalanced with appropriate incentives, this is not always the case. Sometimes soft cache hits could be leveraged in a relatively seamless manner without the potential psychological impact on user quality of experience (QoE) due to *conscient* content replacement[1]. For example, after a user watches a video X, the recommendation system could re-order its list of recommendations (among related contents of roughly equal similarity to X) to favor a cache hit in the next request, without the user being aware of this change or liking the recommended contents less. Such systems have already been considered, and would be complementary to our proposal [33, 53]. A similar case could be made for online radio type of apps (like lastFM, Pandora, Spotify, etc.). While a lot more can be said about each of the above preliminary incentive ideas, and plenty more thinking might be needed to go from these to concrete business cases, we believe these suffice to motivate an investigation of the potential impact of soft cache hits.

While soft cache hits could provide some benefits on top of an existing caching policy, a first key observation is that the optimal caching policy might differ, sometimes radically, when soft cache hits are allowed. As a simple example, consider a single cache with a tiny content catalog with contents A, B, C of popularities 3, 2, 2, respectively (e.g. number of requests per minute). If the cache could fit only a single content, traditional caching will choose to store the most popular content (A), leading to a cache hit ratio of $3/(3+2+2)$, approx. 43%. However, assume we knew that 1 out of 2 users requesting A, would be willing to watch content C instead (e.g. because C is highly related to A, and available locally at HD). Same for users requesting content B. Then, caching content C would satisfy all requests for C (2), half the requests for B ($0.5 \cdot 2$), and half the requests

[1] A user that has already chosen a content might over-value her original choice and feel unhappy to swap it to an objectively equally interesting content. This effect is somewhat akin to the well-known *endowment effect* from Behavioral Economics [83].

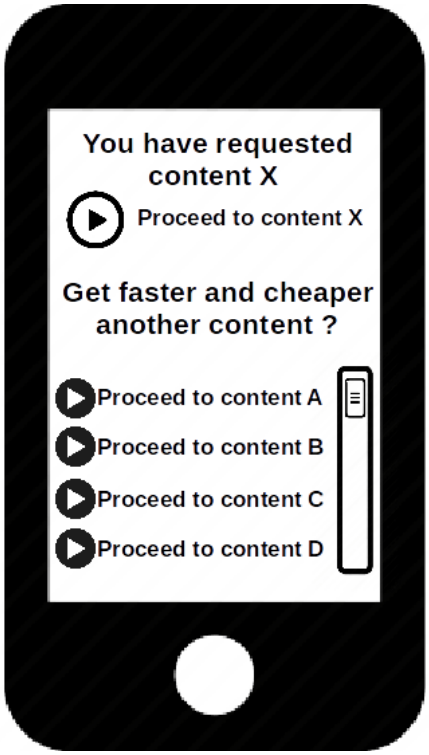

Figure 5.1 – Mobile app example for *Soft Cache Hits* with related content *recommendation* (that the user might not accept)

Figure 5.2 – Mobile app example for *Soft Cache Hits* with related content *delivery*

for A ($0.5 \cdot 3$), leading to a cache hit ratio of 4.5/7, approximately 64% (an almost 50% improvement over the standard policy). This simple example motivates the significant potential of the approach, but also the need for a fundamental reconsideration of caching policies, even in relatively simple networking setups. Finally, while this simple example might tempt the reader to think that the new optimal policy is simply to (re-)rank contents based on *total* hit rate each can achieve (including SCHs), and then apply standard policies (e.g. picking the highest ranked ones), in fact we will show that the optimal policy is a hard combinatorial (cover) problem.

5.1.3 Contributions

The main contributions of the paper are summarized as follows.

- *Soft Cache Hits (SCH) concept:* We introduce the novel concept of soft cache hits. To our best knowledge, this is the first time that this idea has been applied to edge caching for cellular networks (besides our own preliminary work [34]).

- *Soft Cache Hits (SCH) model:* We propose a generic model for mobile edge caching with soft cache hits that can capture a number of interesting content substitution scenarios (e.g. both Fig. 5.1 and Fig. 5.2) and is versatile enough to apply on top of both non-cooperative (i.e. single cache) and cooperative caching frameworks [22], as well as various networking assumptions.

- *Analytical Investigation:* We prove that the problem of optimal edge caching with SCH is NP-hard even when considering a single cache only. This is in stark contrast to the standard case without SCH. We then prove that, despite the increased complexity, the generic problem of femto-caching with SCH still exhibits properties that can be taken advantage of to derive efficient approximation algorithms with provable performance.

- *Trace-based Validation:* We corroborate our SCH proposal and analytical findings through an extended evaluation on 5 real datasets containing information about related content, demonstrating that promising additional caching gains could be achieved in practice.

As a final remark, it is important to stress that *we do not propose to modify the recommendation systems themselves* (unlike [33, 53], for example). Instead, *our focus is on the caching policy side*, using the output of a state-of-art recommendation system for the respective content type as input to our problem (this will be further clarified in Section 5.2). Of course, a content provider with a recommendation system can benefit from our approach to optimize its caching policies, or even modify its recommendations to incorporate soft cache hits. For example, upon peak hours, a content provider can carefully "steer" recommendations to optimize the network performance and user experience (e.g., from lower latency). Moreover, jointly optimizing both the caching and the recommendation sides of the problem could offer additional benefits. We defer this to future work. Overall, we believe that such a convergence between recommendation

and caching systems is quite timely, given that dividing lines between Mobile Network Operators (MNO) and content providers are becoming more blurry, due to architectural developments like Multi-access Edge Computing (MEC) [80] and RAN Sharing [84].

In the following section we introduce the problem setup and our soft cache hits model corresponding to the example application of Fig. 5.1. In Section 5.3 we formulate and analyze the problem of edge caching with SCH for a single cache, and propose efficient caching algorithms. Then, in Section 5.4, we generalize the problem, analysis, and algorithms to the femto-caching case. In Section 5.5 we extend our model to capture scenarios as in the example application of Fig. 5.2, and show that our analytic findings are applicable to these scenarios as well. The performance evaluation is presented in Section 5.6.

5.2 Problem Setup

5.2.1 Network and Caching Model

Network Model: Our network consists of a set of users $\mathcal{N}$ ($|\mathcal{N}| = N$) and a set of SCs (or, *helpers*) $\mathcal{M}$ ($|\mathcal{M}| = M$). Users are mobile and the SCs with which they associate might change over time. Since the caching decisions are taken in advance (e.g., the night before, as in [22, 48], or once per few hours or several minutes), it is hard to know the exact SC(s) each user will be associated at the time she requests a content. To capture user mobility, we propose a more generic model than the fixed bipartite graph of [22]:

$$q_{ij} \doteq Prob\{\text{user i in range of SC j}\},$$

or, equivalently, q_{ij} is the percentage of time a user i spends in the coverage of SC j. Hence, deterministic q_{ij} ($\in \{0, 1\}$) captures the static setup of [22], while uniform q_{ij} ($q_{ij} = q, \forall i, j$) represents the other extreme (no advance knowledge).

Content Model: We assume each user requests a content from a catalogue $\mathcal{K}$ with $|\mathcal{K}| = K$ contents. A user $i \in \mathcal{N}$ requests content $k \in \mathcal{K}$ with probability p_k^i.[2] We will initially assume that all contents have the same size, and relax the assumption later.

Cache Model (Baseline): We assume that each SC/helper is equipped with storage capacity of C contents (all our proofs hold also for different cache sizes). We use the integer variable $x_{kj} \in \{0, 1\}$ to denote if content k is stored in SC j. In the traditional caching model (baseline model), if a user i requests a content k which is stored in some nearby SC, then the content can be accessed directly from the local cache and a *cache hit* occurs. This type of access is considered "cheap", while a *cache miss* leads to an "expensive" access (e.g., over the SC backhaul and core network).

For ease of reference, the notation is summarized in Table 5.1.

[2]This generalizes the standard femto-caching model [22] which assumes same popularity per user. We can easily derive such a popularity p_k from p_k^i.

Table 5.1 – Important Notation

$\mathcal{N}$	set of users ($\|\mathcal{N}\| = N$)
$\mathcal{M}$	set of SCs / helpers ($\|\mathcal{M}\| = M$)
C	storage capacity of a SC
q_{ij}	probability user i in range of SC j
$\mathcal{K}$	set of contents ($\|\mathcal{K}\| = K$)
p_k^i	probability user i to request content k
x_{kj}	k is stored in SC j ($x_{kj} = 1$) or not ($x_{kj} = 0$)
u_{kn}^i	utility of content n for a user i requesting content k
$F_{kn}(x)$	distribution of utilities u_{kn}^i, $F_{kn}(x) = P\{u_{kn}^i \leq x\}$
u_{kn}	avg. utility for content pair $\{k, n\}$ (over all users)
s_k	size of content k

5.2.2 Soft Cache Hits

Up to this point the above model describes a baseline setup similar to the popular femto-caching framework [22]. The main departure of our setup is the following.

Related Content Recommendation: When a user consumes a content (or initially requests a content) that is not found in the local cache, we assume that an app proxy (e.g. YouTube, Netflix, Spotify), collocated or near this cache, looks at the list of contents its recommendation system deems related to the currently consumed content (or the initial request), checks which of them are available in the local cache, and recommends them to the user (see the example in Fig. 5.1). If a user selects to consume next (or instead of the initial) one of them, a *(soft) cache hit* occurs, otherwise there is a cache miss and the network must fetch and deliver the original content.

Below, we first propose a soft cache hit model that captures the scenario of Fig. 5.1. We will use this model throughout Sections 5.3 and 5.4, to develop most of our theory. However, in Section 5.5, we will modify our model to also analyze the scenario of Fig. 5.2, which we will refer to as *Related Content Delivery*.

Definition 15. *A user i that requests a content k that is not available, accepts a recommended content n with probability u_{kn}^i, where $0 \leq u_{kn}^i \leq 1$, and $u_{kk}^i = 1, \forall i, k$.*

These utilities/probabilities (in the remainder we use these terms interchangeably) define a content relation matrix $\mathbf{U^i} = \{u_{kn}^i\}$ for each user. They could be estimated from past statistics and/or user profiles, and are closely related to the output of the recommender for that user and that content app. For example, if a collaborative filtering algorithm suggested that the cosine distance [62] between files k and n for user i is 0.5, we could set $u_{kn}^i = 0.5$[3].

In some cases, the system might have a coarser view of these utilities (e.g., item-item recommendation [85]). We develop our theory and results for the most generic case of

[3]Going from a content relation value to a value for the willingness of a user to accept that related content arguably entails some degree of subjectivity, given that this also depends on the amount and type of incentives offered to the user. Nevertheless, it is clear that whatever the actual value of u_{kn}^i, it will be some function of and positively correlated to underlying content relevance, which can be readily available from the respective recommendation system.

Definition 15, but we occasionally refer to the following two subcases, which might appear in practice:

Sub-case 1: The system does not know the exact utility u_{kn}^i for each node i, but only how they are distributed among all nodes, i.e., the distributions $F_{kn}(x) \equiv P\{u_{kn}^i \leq x\}$.

Sub-case 2: The system knows only the *average utility* u_{kn} per content pair $\{k, n\}$.

5.3 Single Cache with Soft Cache Hits

In order to better understand the impact of the related content matrices $\mathbf{U^i}$ on caching performance, we first consider a scenario where a user i is served by a single small cell, i.e., each user is associated to exactly one SC, but we might still not know in advance which. Such a scenario is in fact relevant in today's networks, where the cellular network first chooses a single SC to associate a user to (e.g., based on signal strength), and then the user makes its request [86]. In that case, we can optimize each cache independently. We can also drop the second index for both the storage variables x_{kj} and connectivity variables q_{ij}, to simplify notation.

In the remainder, we select the cache hit ratio (CHR) as the basic performance metric, similarly to the majority of the related work. However, the analysis for CHR maximization can be generalized to utility maximization [87], where "utility" can be the content access cost or delay, energy consumption, etc.

5.3.1 Soft Cache Hit Ratio

A request (from a user to a SC/helper) for a content $k \in \mathcal{K}$ would result in a (standard) cache hit only if the SC/helper stores the content k in its cache, i.e., if $x_k = 1$. Hence, the (baseline) *cache hit ratio* for this request is simply

$$CHR(k) = x_k$$

If we further allow for soft cache hits, the user might be also satisfied by receiving a different content $n \in \mathcal{K}$. The probability of this event is, by Definition 15, equal to u_{kn}^i. The following Lemma derives the total cache hit ratio in that case.

Lemma 16 (Soft Cache Hit Ratio (SCHR))**.** *Let $SCHR$ denote the expected cache hit ratio for a single cache (including regular and soft cache hits), among all users. Then,*

$$SCHR = \sum_{i=1}^{N} \sum_{k=1}^{K} p_k^i \cdot q_i \cdot \left(1 - \prod_{n=1}^{K} \left(1 - u_{kn}^i \cdot x_n\right)\right). \tag{5.1}$$

Proof. The probability of satisfying a request for content k by user i with related content n is $P\{n|k,i\} = u_{kn}^i \cdot x_n$, since u_{kn}^i gives the probability of acceptance (by definition), and x_n denotes if content n is stored in the cache (if the content is not stored, then $P\{n|k,i\} = 0$). Hence, it follows easily that the probability of a *cache miss*, when content

k is requested by user i, is given by[4] $\prod_{n=1}^{K}(1 - u_{kn}^{i} \cdot x_n)$. The complementary probability, defined as the *soft cache hit ratio* (SCHR), is then

$$SCHR(i, k, \mathbf{U}) = 1 - \prod_{n=1}^{K}(1 - u_{kn}^{i} \cdot x_n). \tag{5.2}$$

Summing up over all users that might be associated with that BS (with probability q_i) and all contents that might be requested (p_k^i) gives us Eq.(5.1). □

Lemma 16 can be easily modified for the sub-cases 1 and 2 of Definition 15 presented in Section 5.2.2. We state the needed changes in Corollary 5.3.1.

Corollary. *Lemma 16 holds for the the sub-cases 1 and 2 of Definition 15, by substituting in the expression of Eq. (5.1) the term u_{kn}^{i} with*

$$u_{kn}^{i} \rightarrow E[u_{kn}^{i}] \equiv \int (1 - F_{kn}(x))\, dx \qquad \textit{(for sub-case 1)} \tag{5.3}$$

$$u_{kn}^{i} \rightarrow u_{kn} \qquad \textit{(for sub-case 2)} \tag{5.4}$$

Proof. The proof is given in Appendix .3. □

5.3.2 Optimal SCH for Equal Content Sizes

The (soft) cache hit ratio depends on the contents that are stored in a SC/helper. The network operator can choose the storage variables x_k to maximize SCHR by solving the following optimization problem.

OP 11. *The optimal cache placement problem for a single cache with soft cache hits and content relations described by the matrix $\mathbf{U^i} = \{u_{kn}^{i}\}$, $\forall i \in \mathcal{N}$, is*

$$\underset{X=\{x_1,\ldots,x_K\}}{maximize} \; f(X) = \sum_{i=1}^{N}\sum_{k=1}^{K} p_k^i \cdot q_i \cdot \left(1 - \prod_{n=1}^{K}\left(1 - u_{kn}^{i} \cdot x_n\right)\right) \tag{5.5}$$

$$s.t. \quad \sum_{k=1}^{K} x_k \leq C. \tag{5.6}$$

In the following, we prove that the above optimization problem is NP-hard (Lemma 17), and study the properties of the objective function Eq.(5.5) (Lemma 18) that allow us to design an efficient approximate algorithm (Algorithm 6) with provable performance guarantees (Theorem 2).

Lemma 17. *The Optimization Problem 11 is NP-hard.*

[4]To simplify our analysis, throughout our proofs we will assume that the user is informed about *all cached contents n with non-zero relevance u_{kn}^{i} to the original content k*. In practice, only a limited number of them would be recommended (e.g. the N most related among the cached ones, as in [53])). Our analysis also holds for this case, with limited modifications.

Algorithm 6 $\left(1-\frac{1}{e}\right)$-approximation Greedy Algorithm for Optimization Problem 11. *computation complexity:* $O\left(C \cdot K\right)$

Input: *utility* $\{u^i_{kn}\}$, *content demand* $\{p^i_k\}$, *mobility* $\{q_i\}$, $\forall k, n \in \mathcal{K},\ i \in \mathcal{N}$
1: $S_0 \leftarrow \emptyset;\ t \leftarrow 0$
2: **while** $t < C$ **do**
3: $\quad t \leftarrow t + 1$
4: $\quad n \leftarrow \underset{\ell \in K \setminus S_{t-1}}{argmax} f(S_{t-1} \cup \{\ell\})$
5: $\quad S_t \leftarrow S_{t-1} \cup \{n\},$
6: **end while**
7: $S^* \leftarrow S_t$
8: **return** S^*

Lemma 18. *The objective function of Eq.(5.5) is submodular and monotone (non-decreasing).*

The proofs for the previous two Lemmas can be found in Appendices .4 and .5, respectively.

We propose Algorithm 6 as a greedy algorithm for Optimization Problem 11: to select the contents to be stored in the cache, we start from an empty cache (line 1), and start filling it (one by one) with the content that increases the most the value of the objective function (line 4), till the cache is full. The computation complexity of the algorithm is $O\left(C \cdot K\right)$, since the loop (lines 2-6) denotes C repetitions, and in each repetition the objective function is evaluated y times, where $K \geq y \geq K - C + 1$. An efficient implementation of the step in line 4 can be based on the method of *lazy evaluations of the objective function*; due to space limitations, we refer the interested reader to [88].

The following theorem gives the performance bound for Algorithm 6.

Theorem 2. *Let OPT be the optimal solution of the Optimization Problem 11, and* S^* *the output of Algorithm 6. Then, it holds that*

$$f(S^*) \geq \left(1 - \frac{1}{e}\right) \cdot OPT \tag{5.7}$$

Proof. Lemma 18 shows that the Optimization Problem 11 belongs to the generic category of maximization of submodular and monotone functions (Eq. 5.5) with a cardinality constraint (Eq. 5.6). For such problems, it is known that the greedy algorithm achieves (in the worst case) a $\left(1-\frac{1}{e}\right)$-approximation solution [89, 88]. □

While the above is a strict worst case bound, it is known that greedy algorithms perform quite close to the optimal in most scenarios. In Sec. 5.6 we show that this simple greedy algorithm can already provide interesting performance gains.

5.3.3 Optimal SCH for Different Content Sizes

Till now we have assumed that all contents have equal size. In practice, each content has a different size s_k and the capacity C of each cache must be expressed in Bytes.

Additionally, if a user requests a video of duration X and she should be recommended an alternative one of similar duration Y (note that similar duration does not always mean similar size). While the latter could still be taken care of by the recommendation system (our study of a real dataset in Sec. 5.6 suggests that contents of different sizes might still be tagged as related), we need to revisit the optimal allocation problem: the capacity constraint of Eq.(5.6) is no longer valid, and Algorithm 6 can perform arbitrarily bad [88].

OP 12. *The optimal cache placement problem for a single cache with soft cache hits and variable content sizes, and content relations described by the matrix* $\mathbf{U^i} = \{u^i_{kn}\}$, $\forall i \in \mathcal{N}$, *is*

$$\underset{X=\{x_1,\dots,x_K\}}{maximize} \; f(X) = \sum_{i=1}^{N}\sum_{k=1}^{K} p^i_k \cdot q_i \cdot \left(1 - \prod_{j=1}^{K}\left(1 - u^i_{kn} \cdot x_n\right)\right) \tag{5.8}$$

$$s.t. \quad \sum_{k=1}^{K} s_k x_k \leq C. \tag{5.9}$$

Remark: Note that the objective is still in terms of cache hit ratio, and does not depend on content size. This could be relevant, e.g., when the operator is doing edge caching to reduce *access latency* to contents (latency is becoming a core requirement in 5G).

The problem is a set cover problem variant with a knapsack type constraint. We propose the approximation Algorithm 7 for this problem, which is a "fast greedy" algorithm (based on a modified version of the greedy Algorithm 6) and has complexity $O\left(K^2\right)$.

Theorem 3.
(1) The Optimization Problem 12 is NP-hard.
(2) Let OPT be the optimal solution of the Optimization Problem 12, and S^* the output of Algorithm 7. Then, it holds that

$$f(S^*) \geq \frac{1}{2}\left(1 - \frac{1}{e}\right) \cdot OPT \tag{5.10}$$

Proof. A sketch of the proof can be found in Appendix .6. □

In fact, a polynomial algorithm with better performance $\left(1 - \frac{1}{e}\right)$-approximation could be described, based on [90]. However, the improved performance guarantees come with a significant increase in the required computations, $O\left(K^5\right)$, which might not be feasible in a practical scenario when the catalog size K is large. We therefore just state its existence, and do not consider the algorithm further in this paper (the algorithm can be found in [91]).

5.4 Femtocaching with Related Content Recommendation

Building on the results and analytical methodology of the previous section for the optimization of a single cache with soft cache hits, we now extend our setup to consider

Algorithm 7 $\frac{1}{2} \cdot \left(1 - \frac{1}{e}\right)$-approximation Algorithm for Optimization Problem 12. *computation complexity:* $O\left(K^2\right)$

Input: *utility* $\{u_{kn}^i\}$, *content demand* $\{p_k^i\}$, *content size* $\{s_k\}$, *mobility* $\{q_i\}$, $\forall k, n \in \mathcal{K},\ i \in \mathcal{N}$

$S^{(1)} \leftarrow$ModifiedGreedy($\emptyset$,[s_1, s_2,...,s_k])
$S^{(2)} \leftarrow$ModifiedGreedy($\emptyset$,[1, 1,...,1])
if $f(S^{(1)}) > f(S^{(2)})$ **then**
    $S^* \leftarrow S^{(1)}$
else
    $S^* \leftarrow S^{(2)}$
end if
return S^*

function ModifiedGreedy(S_0,[w_1, w_2,...,w_k])
    $\mathcal{K}^{(1)} \leftarrow \mathcal{K};\ c \leftarrow 0;\ t \leftarrow 0$
    while $\mathcal{K}^{(1)} \neq \emptyset$ **do**
        $t \leftarrow t + 1$
        $n \leftarrow \underset{\ell \in K \setminus S_{t-1}}{argmax} \frac{f(S_{t-1} \cup \{\ell\})}{w_\ell}$
        if $c + w_n \leq C$ **then**
            $S_t \leftarrow S_{t-1} \cup \{n\}$
            $c \leftarrow c + w_n$
        else
            $S_t \leftarrow S_{t-1}$
        end if
        $\mathcal{K}^{(1)} \leftarrow \mathcal{K}^{(1)} \setminus \{n\}$
    end while
    return $\leftarrow S_t$
end function

the complete problem with cache overlaps (referred to as "femtocaching" [22]). Note, however, that we *do* consider user mobility, through variables q_{ij}, unlike previous works in this framework that often assume static users. Here, we focus on the case of fixed content sizes.

In this scenario, a user $i \in \mathcal{N}$ might be covered by more than one SCs/helpers $j \in \mathcal{M}$, i.e. $\sum_j q_{ij} \geq 1, \forall i$. A user is satisfied, if she receives the requested content k or *any* other related content (that she will accept), from *any* of the SCs/helpers within range. Hence, similarly to Eq.(5.2), the total cache hit ratio SCHR (that includes regular and soft cache hits) is written as

$$SCHR(i, k, \mathbf{U}) = 1 - \prod_{j=1}^{M} \prod_{n=1}^{K} \left(1 - u_{kn}^i \cdot x_{nj} \cdot q_{ij}\right) \tag{5.11}$$

since for a cache hit a user i needs to be in the range of a SC j (term q_{ij}) that stores the

content n (term x_{nj}), and accept the recommended content (term u^i_{kn}).

Considering (i) the request probabilities p^i_k, (ii) every user in the system, and (iii) the capacity constraint, gives us the following optimization problem.

OP 13. *The optimal cache placement problem for the femtocaching scenario with soft cache hits and content relations described by* $\mathbf{U^i} = \{u^i_{kn}\}$, $\forall i \in \mathcal{N}$, *is*

$$\underset{X=\{x_{11},\ldots,x_{KM}\}}{maximize} f(X) =$$

$$= \sum_{i=1}^{N}\sum_{k=1}^{K} p^i_k \cdot \left(1 - \prod_{j=1}^{M}\prod_{n=1}^{K}\left(1 - u^i_{kn}\cdot x_{nj}\cdot q_{ij}\right)\right), \tag{5.12}$$

$$s.t. \quad \sum_{k=1}^{K} x_{kj} \leq C, \quad \forall j \in \mathcal{M}. \tag{5.13}$$

The following lemma states the complexity of the above optimization problem, as well as its characteristics that allow us to design an efficient approximation algorithm.

Lemma 19.
(1) The Optimization Problem 13 is NP-hard,
(2) with submodular and monotone (non-decreasing) objective function (Eq. 5.12) and a matroid constraint (Eq. 5.13).

Proof. We prove Lemma 19 by extending the basic ideas of the single-cache case, and following a similar methodology as in the proofs of Lemmas 17 and 18; the detailed proof is given in [91]. □

Lemma 19 states that the Optimization Problem 13 is a maximization problem with a submodular function and a matroid constraint. For this type of problems, a greedy algorithm can guarantee an $\frac{1}{2}$-approximation of the optimal solution [88]. The greedy algorithm is similar to Algorithm 6 and is of computational complexity $O\left(K^2M^2\right)$; i.e., instead of considering only contents in the allocation, now tuples {content, helper} need to be greedily allocated until the caches of helpers are full (for a detailed pseudocode of the algorithm, we refer the reader to [91]).

Theorem 4. *Let OPT be the optimal solution of the Optimization Problem 13, and* S^* *the output of the greedy algorithm. Then, it holds that*

$$f(S^*) \geq \frac{1}{2}\cdot OPT \tag{5.14}$$

Submodular optimization problems have received considerable attention recently, and a number of sophisticated approximation algorithms have been considered (see, e.g., [88] for a survey). For example, a better $\left(1-\frac{1}{e}\right)$-approximation (with increased computation complexity though) can be found following the "multilinear extension" approach [92], based on a continuous relaxation and pipage rounding. A similar approach has also been

followed in the original femto-caching paper [22]. Other methods also exist that can give an $\left(1-\frac{1}{e}\right)$-approximation [93]. Nevertheless, minimizing algorithmic complexity or optimal approximation algorithms are beyond the scope of this paper. Our goal instead is to derive fast and efficient algorithms (like greedy) that can handle the large content catalogues and content related graphs $\mathbf{U}$, and compare the performance improvement offered by soft cache hits. The worst-case performance guarantees offered by these algorithms are added value.

5.5 Femtocaching with Related Content Delivery

We have so far considered a system corresponding to the example of Fig. 5.1, where a cache-aware system *recommends* alternative contents to users (in case of a cache miss), but users might not accept them. In this section, we consider a system closer to our second example of Fig. 5.2, where the system *delivers* some related content that is locally available *instead of the original content*, in case of a cache miss. While a more extreme scenario, we believe this might still have application in a number of scenarios, as explained in Section 5.1 (e.g., for low rate plan users under congestion, or in limited access scenarios [94, 95]).

In the following, we model the related content delivery system, formulate the respective optimization problem, and show that it has the same properties with the problems in the previous sections, which means that our results and algorithms apply to this context as well. We present only the more generic femto-cache case of Sec. 5.4; the analysis and results for the single cache cases of Sec. 5.3 follow similarly.

Since now original requests might not be served, the (soft) cache hit ratio metric does not describe sufficiently the performance of this system. To this end, we modify the definition of content utility:

Definition 16. *When a user i requests a content k that is not locally available and the content provider delivers an alternative content n then the user satisfaction is given by the utility u^i_{kn}. $u^i_{kn} \in \mathbb{R}$ is a real number, and does not denote a probability of acceptance, but rather the happiness of user i when she receives n instead of k. Furthermore $u^i_{kk} = U_{max}, \forall i$.*

Note: we stress that the utilities u^i_{kn} in Definition 16 do not represent the probability a user i to accept a content n (as in Definition 15), but the satisfaction of user i given that she accepted content n. User satisfaction can be estimated by past statistics, or user feedback, e.g., by asking user to rate the received alternative content.

Let us denote as $G_i(t) \subseteq \mathcal{M}$ the set of SCs with which the user i is associated at time t. Given Definition 16, when a user i requests at time t a content k that is not locally available, we assume a system (as in Fig. 5.2) that delivers to the user the cached content with the *highest* utility[5], i.e., the content n where

$$n \equiv \arg\max_{\ell \in \mathcal{K}, j \in G_i(t)} \left\{ u^i_{k\ell} \cdot x_{\ell j} \right\} \tag{5.15}$$

[5]Equivalently, the system can recommend all the stored contents to the user and then allow the user to select the content that satisfies her more.

Hence, the satisfaction of a user i upon a request for content k is

$$\max_{n\in\mathcal{K},j\in G_i(t)} \left\{u^i_{kn}\cdot x_{nj}\right\} \tag{5.16}$$

Using the above expression and proceeding similarly to Section 5.4, we formulate the optimization problem that the network needs to solve to optimize the total user satisfaction (among all users and all content requests), which we call *soft cache hit user satisfaction* (SCH-US).

OP 14 (SCH-US)**.** *The optimal cache placement problem for the femtocaching scenario with related content delivery and content relations described by the matrix* $\mathbf{U}=\{u^i_{kn}\}$ *is*

$$\underset{X=\{x_1,\ldots,x_K\}}{maximize}\ f(X) =$$

$$= \sum_{i=1}^{N}\sum_{k=1}^{K} p^i_k \cdot E_{G_i}\left[\max_{n\in\mathcal{K},j\in\mathcal{M}}\left(u^i_{kn}\cdot x_{nj}\cdot Q_{ij}\right)\right], \tag{5.17}$$

$$s.t. \quad \sum_{k=1}^{K} x_{kj} \le C, \quad \forall j\in\mathcal{M}. \tag{5.18}$$

where $Q_{ij}=\begin{cases} 1 & , \text{ if } j\in G_i \\ 0 & , \text{ otherwise}\end{cases}$ *, and the expectation* $E_{G_i}[\cdot]$ *is taken over the probabilities* $P\{G_i\}=\prod_{j\in G_i} q_{ij}\cdot\prod_{j\notin G_i}(1-q_{ij})$.

For the sub-cases 1 and 2 of Definition 15 presented in Sec. 5.2.2, the following corollary holds.

The expression of Eq. (5.17) needs to be modified as

$$E_{G_i}\left[\max_{n\in\mathcal{K},j\in\mathcal{M}}\left(u^i_{kn}\cdot x_{nj}\cdot Q_{ij}\right)\right] \to E_{G_i}\left[\max_{n\in S}\left(u^i_{kn}\right)\right] =$$

$$= E_{G_i}\left[\int\left(1-\prod_{n\in S}F_{kn}(x)\right)dx\right] \tag{5.19}$$

$$u^i_{kn}\to u_{kn} \tag{5.20}$$

where $S=\{\ell:\ \ell\in\mathcal{K},\ m\in\mathcal{M},\ x_{\ell m}\cdot Q_{im}=1\}$, for the sub-cases 1 and 2 of Definition 15, respectively.

We now prove the following Lemma, which shows that Theorem 4 applies also to the Optimization Problem 14, and thus it can be efficiently solved by the same greedy algorithm (where the objective function of Eq. 5.17 is now used)).

Lemma 20.
(1) The Optimization Problem 14 is NP-hard,
(2) with submodular and monotone objective function (Eq. 5.17).

Proof. The proof is given in Appendix .7. □

5.6 Evaluation

In this section, we investigate the gains of employing soft cache hits and the performance of the proposed algorithms. We first analyze 5 real datasets collected from different content-centric applications/sources, such as YouTube and Amazon-TV, as well as other types of contents (e.g. Android applications) or data sources (like MovieLens) (Sec. 5.6.1). We have also tested our schemes with some data related to personalized radio (lastFM) with similar conclusions. The datasets contain information about content relations, based on which we build the utility matrices $\mathbf{U}$. We use these realistic utility matrices $\mathbf{U}$ in our simulations to study the performance of caching with or without soft cache hits. In Sec. 5.6.2 we describe the simulation setup, and present and discuss the results in Sec. 5.6.3.

5.6.1 Datasets of Content Relations

YouTube dataset. We consider a dataset of YouTube videos from [77][6]. The dataset contains several information about the videos, such as their popularity, size, and a list of related videos (as recommended by YouTube). We build the utility matrix $\mathbf{U} = \{u_{nk}\}$, where $u_{nk} = 1$ if video n is in the list of related videos of k (or vice-versa), and otherwise $u_{nk} = 0$.

Amazon datasets. We also analyze 3 datasets of product reviews from Amazon [96] for Android applications (*Amazon-App*), Movies and TV (*Amazon-TV*), and Videogames (*Amazon-VG*). The datasets include for each item a list of contents that are "also bought". [7] We consider for each dataset 10000 of its items, and build a utility matrix $\mathbf{U} = \{u_{nk}\}$, where $u_{nk} = 1$ if item n is also bought with item k (or vice-versa), and otherwise $u_{nk} = 0$.

MovieLens dataset. We finally consider a movies-rating dataset from the MovieLens website [61], containing 69162 ratings (from 0.5 stars to 5) of 671 users for 9066 movies. As these datasets contain only raw user ratings and not movie relations per se, to obtain content relation matrix U, in this case, we do an intermediate step and apply a standard concept from collaborative filtering [62]. Specifically, we calculate the similarity of each pair of contents based on their common ratings as their cosine-distance metric:

$$sim(n,k) = \frac{\sum_{i=1}^{\#users} r_i(n) \cdot r_i(k)}{\sqrt{\sum_{i=1}^{\#users} r_i^2(n)} \cdot \sqrt{\sum_{i=1}^{\#users} r_i^2(k)}}$$

where we normalized the ratings r_i, by subtracting from each rating the average rating of that item, so that we obtain similarity values $\in [-1, 1]$. Due to the sparsity of the dataset (few common ratings), we also apply an item-to-item collaborative filtering (using 10 similar items) in order to predict the missing user ratings per item, and thus the missing similarity values. We build the utility matrix $\mathbf{U} = \{u_{nk}\}$ with $u_{nk} = \max\{0, sim(n,k)\}$,

[6] Data from 27-07-2008, and depth of search up to 3; see details in [77]

[7] Our main motivation to use the game and app datasets was to also validate the robustness of our approach to other types of data. Soft cache hits though might be more relevant for free apps or games, rather than paid products.

Table 5.2 – Information contained in datasets.

	content popularity	content size	content relations u_{kn} $\in \{0,1\}$	content relations u_{kn} $\in [0,1]$
Amazon-*	×	×	✓	×
MovieLens	✓	×	×	✓
YouTube	✓	✓	✓	×

Table 5.3 – Dataset analysis.

	#contents	content relations $E[R]$ $\left(\frac{std[R]}{E[R]}\right)$	popularity $E[p]$ $\left(\frac{std[p]}{E[p]}\right)$
Amazon-App	8229	16.0 (2.2)	-
Amazon-TV	2789	7.8 (1.0)	-
Amazon-VG	5614	22.0 (1.1)	-
MovieLens	4622	125.8 (0.5)	15 (1.6)
YouTube	2098	5.3 (0.7)	500 (3.1)

i.e, $u_{nk} \in [0,1]$. Finally, we assign to each item a popularity value equal to the number of ratings for this item.

For ease of reference, Table 5.2 presents the information contained in each dataset.

Due to the sparsity of the YouTube dataset, we only consider contents belonging to the largest connected component (defining as adjacencies, the positive entries of the utility matrix). For consistency, we consider only the contents in the largest connected component for the other datasets as well. Moreover, since the Amazon and YouTube datasets do not contain per-user information, and the per-user data in the MovieLens dataset is sparse, we consider the sub-case-2 of Definition 15, i.e., $u^i_{kn} = u_{kn}$ for all users i.

The number of remaining contents for each dataset are given in Table 5.3. We also calculate for each content the number of its related contents $R_n = \sum_k u_{nk}$ (or the sum of its utilities for the MovieLens dataset where $u_{kn} \in [0,1]$), and present the corresponding statistics in Table 5.3 along with the statistics for the content popularity.

5.6.2 Simulation Setup

Cellular network. We consider an area of 1 km^2 that contains M SCs. SCs are randomly placed in the area (following a Poisson point process), which is a common assumption in related work [22, 97]. An SC can serve a request from a user, when the user is inside its communication range, which we set to 200 meters. We also consider N mobile users.

We select as default parameters: $N = 50$ users, and $M = 20$ SCs with caching capacity $C = 5$ (contents). This creates a relatively dense network, where a random user is connected to 3 SCs *on average*.

Content demand. We consider a scenario of content demand for each dataset of

Sec. 5.6.1, with the corresponding set of contents, content popularities and relations (utility matrix). For datasets without information on content popularity (see Table 5.2), we generate a random sample of popularity values drawn from a Zipf distribution in $[1, 400]$[8] with exponent $\alpha = 2$. For each scenario we generate a set of 20 000 requests according to the content popularity, over which we average our results. When soft cache hits are allowed, we assume the *related content recommendation* model of Definition 15 (see also Fig. 5.1).

Unless otherwise stated, the simulations use the default parameters summarized in Table 5.4.

Caching schemes / algorithms. We consider and compare the following schemes for single-SC (*single*) and multi-SC (*femto*) to user association.

- *Single*: A single cache accessible per user (e.g., the closest one). Only normal cache hits allowed, and the most popular contents are stored in each cache, which is the optimal policy in this simple setup. It will serve as the baseline for single cache scenarios.
- *SingleSCH*: Here soft cache hits are allowed. However, the caching policy is still based on popularity as before (i.e., is not explicitly optimized to exploit SCHs).
- *SingleSCH**: This is our proposed policy. Here soft cache hits are allowed, *and* the caching policy is optimized to fully exploit this (according to Algorithm 6 or Algorithm 7).
- *Femto*: Femto-caching without soft cache hits. This is the baseline scheme for this class of scenarios, where the proposed algorithm from [22] is applied.
- *FemtoSCH*: Femto-caching based content placement (same as in *Femto)*, but allowing soft cache hits on user requests (a posteriori).
- *FemtoSCH**: Our proposed policy. Femto-caching is explicitly optimized for soft cache hits, according the greedy algorithm (Sec. 5.4).

5.6.3 Results

1. Overall performance

We simulate scenarios for all datasets / utility matrices with the default parameters (Table 5.4), both under single and multi user-SC association. Fig. 5.3 shows the achieved cache hit ratio CHR (or soft cache hit ratio, SCHR) under the baseline caching (*Single/SingleSCH/Femto/FemtoSCH*) and the SCHR under a content placement using our algorithms (*SingleSCH*/FemtoSCH**).

Key Message: ***Allowing soft cache hits can lead to a dramatic increase in the cache hit ratio.***

Comparing the cache hit ratio (CHR) under the popularity-based caching (*Single/Femto* - red/pink bars) and the schemes we propose (*SingleSCH*/FemtoSCH** -

[8]The max value is selected equal to the max number of requests per user, i.e., $\frac{\#requests}{\#contents}$.

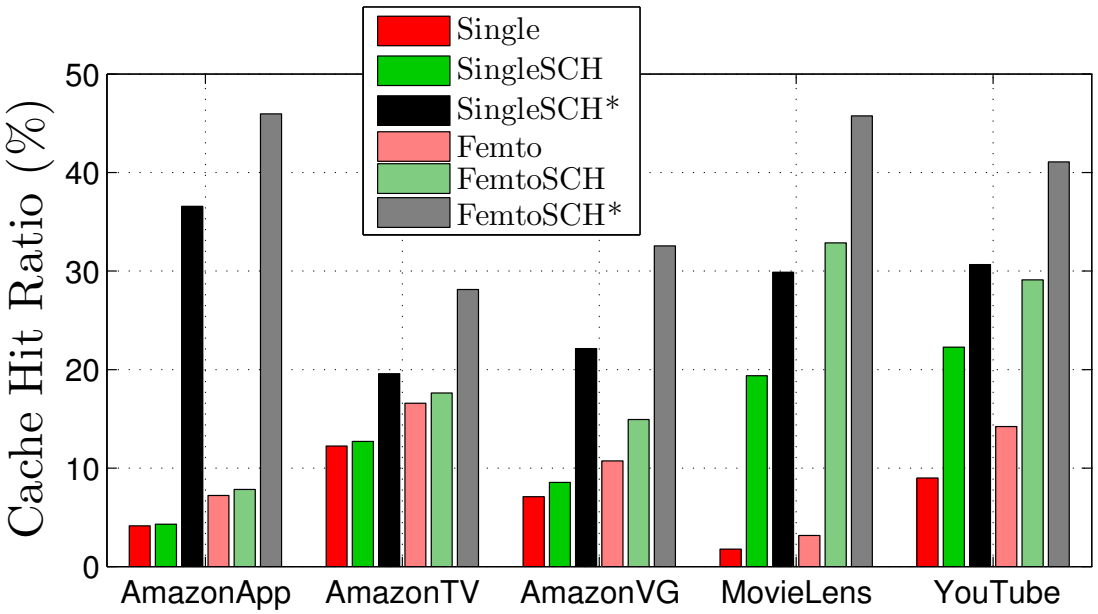

Figure 5.3 – Cache hit ratio for all datasets and caching schemes, for the default scenario.

Table 5.4 – Parameters used in simulations: default scenario.

Parameter	**Value**	**Parameter**	**Value**
Area	1 x 1 km	Cache size, C	5
nb. of SCs, M	20	nb. of users N	50
SC comm. range	200 m	Zipf distr.	$\in [1, 400]$, $\alpha = 2$

black/grey bars), shows that allowing soft cache hits brings a significant increase in the CHR for all datasets. The relative gain ranges from 60% in the *Amazon-TV* case, up to around 780% in the *Amazon-App* case, for the *Single* scenarios; the relative gains in the *Femto* scenarios are similarly impressing (from 70% up to 530%, respectively). These initial results indicate that soft cache hits can be a promising solution for future mobile networks, by increasing the virtual capacity of mobile edge caching.

Key Message: ***While gains can sometimes already be achieved just by allowing soft cache hits, to fully take advantage of soft cache hits, the caching policy should be redesigned to explicitly take these into account (through the utility matrix).***

Fig. 5.3 demonstrates that gains could already be achieved by simply introducing soft cache hits on top of existing (state-of-the-art) caching policy (*SingleSCH/FemtoSCH* - dark/light green bars), but these are scenario-dependent. For example, in the *Amazon* scenarios the increase in CHR by allowing soft cache hits is marginal (red vs. green bars), while in the *YouTube* scenario it is 1.5× higher. In contrast, explicitly designing the caching policy to exploit soft cache hits allows for important gains in all scenarios

(black/grey bars). Specifically, in the *Amazon* scenarios the performance gains are almost entirely due to the caching algorithm (just allowing soft cache hits, does not improve performance), while in the *YouTube* scenario our utility-aware algorithms outperform by around 40% popularity-based caching. These results show clearly that existing caching policies are not capable to exploit the full potential of soft cache hits.

2. Impact of network parameters

We proceed to study the effect of network parameters, on the performance of soft cache hits schemes. We consider the *YouTube* dataset, for which the soft cache hits schemes (*SingleSCH*/FemtoSCH**) have a moderate gain (around $1.5 - 3\times$) over the baseline schemes. We simulate scenarios where we vary the cache size C and the number of SCs M; the remaining parameters are set as in the default scenario (Table 5.4).

Key Message: ***(a) Soft cache hits improve performance irrespectively of the underlying network parameters (even for a few SCs with small capacity); (b) Combining femto-caching and soft cache hits achieves significantly higher CHR that today's solutions.***

Cache size impact: We first investigate the impact of cache size, assuming fixed content sizes. Fig. 5.4 depicts the total cache hit ratio, for different cache sizes C: we consider a cache size per SC between 2 and 15 contents. The simulations suggest that the *SingleSCH*/FemtoSCH** scenarios consistently achieve more than $2.5\times$ (single) and $1.2\times$ (femto) higher CHR than *Single/Femto*. What is more, these gains are applicable to both single- and femto- caching. The two methods (femto-caching and soft cache hits) together offer a total of $3.3\times$ to $7\times$ improvement compared to the baseline scenario *Single*. Finally, even with a cache size per SC of about 0.1% of the total catalog ($C = 2$), introducing soft cache hits offers 29% CHR (*SingleSCH**), whereas today's practices (popularity-based caching without SCH) would achieve only 4% CHR.

SC density impact: In Fig. 5.5 we consider the impact of SC density. In sparse scenarios (e.g., $M = 5$), a user usually is in the range of at most one SC. For this reason, Femto and Single perform similarly. As the SC density increases, the basic *Femto* is able to improve performance, as expected, by exploiting cache cooperation. However, every $2\times$ increase in density, which requires the operator doubling the infrastructure cost, offers roughly a relative improvement of $30 - 50\%$. Simply introducing soft cache hits instead, suffices to provide a $2\times$ improvement.

Key Message: ***The extra cost to incentivize soft cache hits might be quite smaller than the CAPEX/OPEX costs in infrastructure investment to achieve comparable performance gains.***

3. Impact of utility matrix

We further investigate the impact of the content relations as captured by the matrix $\mathbf{U}$ (and its structure). To quantify the content relations, we use as a metric the sum of the utilities per content $R_n = \sum_k u_{nk}$ (see also Sec. 5.6.1 and Table 5.3).

Key Message: ***The CHR increases with the density ($E[R]$) of the utility ma-***

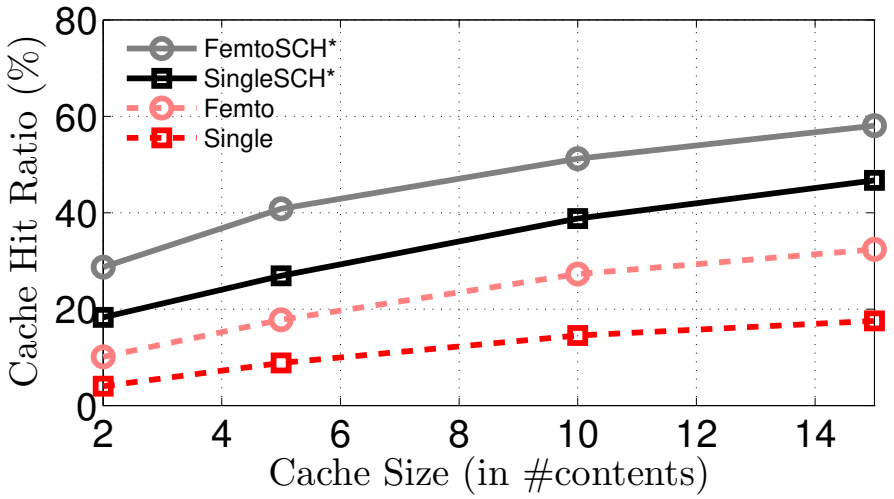

Figure 5.4 – Cache hit ratio vs. cache size C.

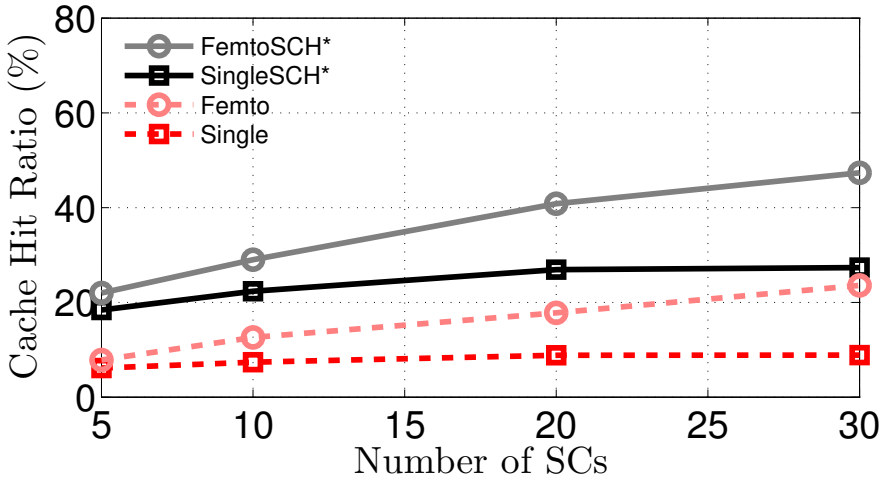

Figure 5.5 – Cache hit ratio vs. number of SCs M.

trix. Even low utility values u_{kn} ***can significantly contribute towards a higher CHR.***

The first important parameter to consider is the average value of R_n, i.e., the density of the utility matrix $\mathbf{U}$. We consider the *MovieLens* dataset, where the utilities u_{kn} are real numbers in the range $[0, 1]$. To investigate the impact of the density of $\mathbf{U}$, we consider scenarios where we vary the matrix $\mathbf{U}$: for each scenario we set a threshold u_{min} and take into account only the relations between contents with utility higher than u_{min}, i.e.,

$$\mathbf{U}' = \{u'_{kn}\} = \begin{cases} u_{kn} & \text{if } u_{kn} \geq u_{min} \\ 0 & \text{if } u_{kn} < u_{min} \end{cases}$$

Table 5.5 gives the density of the utility matrix for the different values of the threshold u_{min}, and Fig. 5.6 shows the cache hit ratio for these scenarios. When u_{min} is set to a large value (i.e., only few relations are taken into account and the matrix $\mathbf{U}$ is sparse), there is no ($u_{min} = 0.75$) or negligible ($u_{min} = 0.5$) improvement from soft cache hits. However, as the density of $\mathbf{U}$ increases (lower thresholds u_{min}), the gains in CHR significantly increase; note that these gains are from content relations with low utility values (i.e., less than 0.5 or 0.25 for the two rightmost scenarios, respectively). Moreover, it is interesting that in the scenario with $u_{min} = 0.25$, the gains are almost entirely due to the more efficient caching from our algorithms, i.e., popularity-based caching would not be efficient even if soft cache hits were allowed (green bars).

Table 5.5 – Utility matrix density for the MovieLens dataset for different u_{min} thresholds.

threshold u_{min}	0.75	0.5	0.25	0
$E[R]$	0.9	10.8	49.0	125.8

Key Message: ***Highly skewed distributions of #relations,*** R_n***, can lead to more efficient caching (in analogy to heavy tailed popularity distributions).***

Our simulation study demonstrates the effect of the variance of the number of related contents, i.e., $\frac{std[R]}{E[R]}$. We consider the three *Amazon* scenarios of Fig. 5.3 that have the same content popularity distribution and similar $E[R]$ values (see Sec. 5.6.2). Fig. 5.7 shows the relative gain (of soft cache schemes over baseline schemes) in these scenarios where the distribution of R_n has different variance (see Table 5.3). When the variance is very high (*Amazon-App*, $\frac{std[R]}{E[R]} = 2.2$) the CHR under soft cache hit schemes is almost an order of magnitude larger than the baseline scenarios. Large variance means that a few contents have very high R_n; thus storing these contents allows to serve requests for a large number of other (non-cached) contents as well. Finally, an interesting observation is that the variance of the distribution plays a more important role than the density of the utility matrix: although the utility matrix in the *Amazon-VG* scenario ($E[R] = 22$, $\frac{std[R]}{E[R]} = 1.1$) is denser than in the *Amazon-App* scenario ($E[R] = 16$, $\frac{std[R]}{E[R]} = 2.2$), the gain of the latter is higher due to the higher variance.

4. Performance of scheme extensions

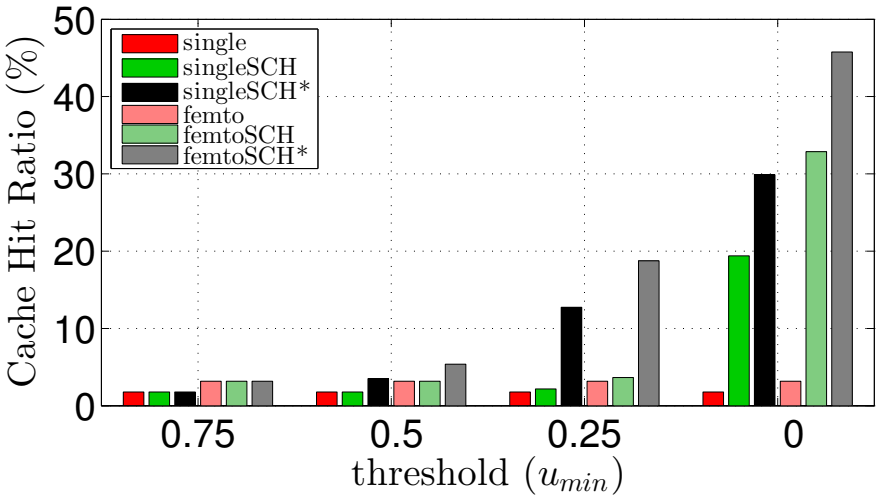

Figure 5.6 – Cache hit ratio for the MovieLens dataset for scenarios with different u_{min} thresholds; default scenario.

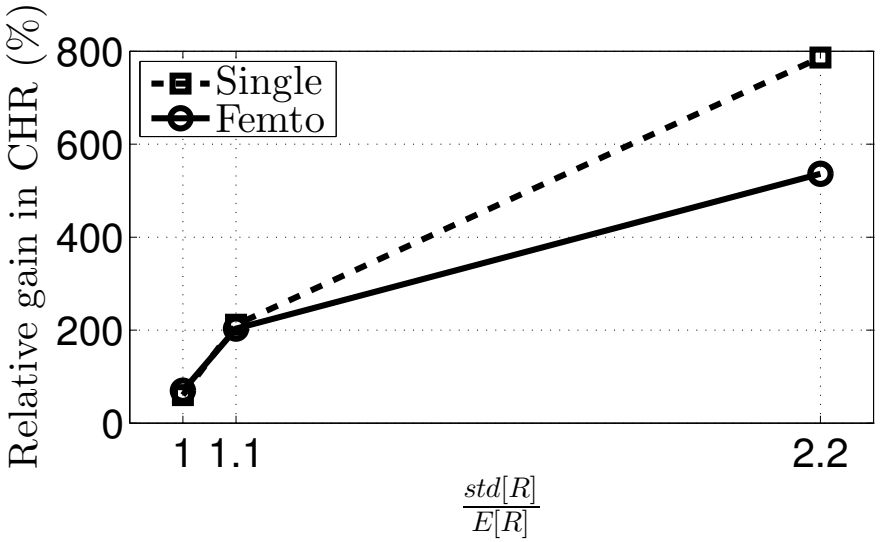

Figure 5.7 – Relative increase in the cache hit ratio due to soft cache hits (y-axis). *Amazon* scenarios with different variance of number of related contents (x-axis).

Key Message: ***The gain of our algorithms is consistent for all the considered variations of soft cache hits scenarios.***

Finally, we evaluated scenarios with (a) contents of different size and (b) *related content delivery* model (Def. 16), and observed the following. In the former scenarios (from the *YouTube* dataset), the performance improves considerably for all cache size values (e.g., similarly to the equal content sizes case). In the latter scenarios (from the *MovieLens* dataset; similar to scenarios of Fig. 5.6), the *user satisfaction* singificantly increases with related content delivery (i.e., soft cache hits), and denser matrices (i.e., higher willingness of users to accept related contents) lead to better performance.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The current thesis is an analytical approach to lay bridges between two already well studied areas: namely recommender systems and caching in wireless networks. The research of this work was primarily focused on freezing one variable of the problem, that being caching or more generally network state and dealing with ways to optimize the other one, i.e., recommendation policy. Along these lines, we initially started by posing a very challenging problem, that is

"How to favor low cost content in the sequential access regime by maintaining some lower bounded user satisfaction?"

We opted to model a user traversing the catalogue as a Markov Process which resembled a "recommendation driven PageRank" random walk. Subsequently, we formulated a continuous optimization problem over the content/object recommendation frequencies with lower bounded recommendation quality and specific recommendation budget constraints.

Having said that, we showed that our optimization problem is nonconvex and applied an ADMM algorithm on that. Interestingly, despite the lack of optimality guarantees, our proposed ADMM which aimed to optimize the cost over a very long horizon significantly outperformed existing myopic solutions and interesting gains were observed. In this work however, our main objective was to formalize a clear target (come up with a model), find ways to reach it (provide some solution) and eventually understand whether this whole approach is actually meaningful in practice (results). Importantly, this work lacked solid theoretical guarantees about our algorithm but definitely managed to draw our attention in order to further work on our preliminary results.

In Chapter 3, the goal was twofold. First it was really important to

"Establish performance guarantees for the long session NFR."

We started off from the original nonconvex problem as modeled in Chapter 2; through a series of variable manipulations and some intermediate steps the problem was transformed

into an LP. We consequently had to establish the necessary conditions on the problem data such that the transformation from one optimization problem to the other be a bijection. A problem baring some resemblances to our and dealt with PageRank optimization was also convexified in [98]. Turning this seemingly quite challenging problem into an LP gave us two important returns

1. Optimal solution for the long session NFR problem.
2. LP solution meant faster solution than the previously proposed ADMM.

The second goal of this work was to dive deeper into a practical aspect of a recommendation system, that is

"Can the user biasing to some positions (e.g. the higher ones) further enhance the system performance?"

And by that we mean that the ultimate goal is to understand whether a user with such position preferences can further decrease the access costs. Essentially, the two presented models have a fundamental difference, that is in Chapter 2 we assumed that users are presented with N suggestions and may choose any of those N equally likely, whereas now, by incorporating the click-through of the specific positions, we have to consider this as well in our optimization process. In an extreme scenario where some user has very skewed pmf over the positions, this knowledge can prove to be critical, as essentially it is like *decreasing our budget*, virtually the system has to decide $N = 1$ recommendation. In our problem, ideally we need low cost *and* related items, which obviously cannot be too many for ever item. Thus, exploiting which positions the user clicks, we can place our top candidate and most valuable contents (i.e., low cost and related) in the correct positions of the screen.

The long session NFR problem with position preferences proved to be some generalization of the the formulation of Chapter 2 which needed an expansion of the $K \times K$ matrix to a $K \times K \times N$ structure (or tensor). Thus, the LP transformation gave us the optimal solution of the problem described in Chapter 2, and using this result we also got the optimal solution of the problem with position preferences for free.

In Chapter 4, which is the final chapter dealing with the long session NFR, our main goal can again be summarized in two set of goals.

"Scale up the problem sizes and solve for more realistic use cases."

To this end, we decided to pick up from the arsenal of sequential decision making methods probably the most celebrated one, that is the Markov Decision Process. A very clear motivation we had in the start was that in the previous two chapters, the problem was formulated by assuming an infinitely long horizon of user requests. This caused two issues

1. Unrealistic infinite horizon.

2. Unnecessary computations because of the *infinite length* session.

The interesting thing about our problem is that it can be cast as an infinite horizon problem with discounts, *but* through the road of *random length session that is distributed as Geo*(λ). This formulation resolves for us the two problematic aspects of our previous approach; we can now solve the problem by using the statistics of user session length (λ) and more importantly avoid the extra computations that are needed in order to "reach the *infinite* session".

What is more, the MDP formulation reveals some easier problems that have clear structure and that is something that we had to capitalize. In return we got significantly improved execution times of *all of our previous solutions*, i.e., the customized ADMM and CPLEX solution of the LP by being ϵ-optimal at the same time.

The MDP framework, because of its natural decomposition over the problem files (each optimization is over some specific file *always*) further allowed us to relax our assumptions. We removed the quality constraints and assumed user click-through probabilities that are incorporated in our problem's objective. This intuitively allowed us to solve more general use cases without many unrealistic modeling assumptions.

Finally in Chapter 5, we took a completely different approach on the problem and considered a more "network" type of application. We modeled a network where each user is associated with some SCs, which are equipped with some storage space, in probabilistic (and thus deterministic if we want) rules. Moreover, there is an underlying graph between the contents, one can associate that with the utilities u_{ij} that are described throughout this manuscript, which show the level happiness a user would get when asking for content i and instead he received item j.

To this end, we were interested in finding

"Optimal caching policy in scenarios where the contents can be assumed as interchangeable."

We formalize two maximization problems of the cache hit rate by assuming an IRM type of request pattern for the wireless users. Importantly the problems that arise in this setup will be computationally very intensive if one is interested in realistic problem sizes. Along these lines, we prove that our problems are NP-hard, but are also submodular and monotonic with matroid constraints. The fomer was the red light which forced us to resort to greedy methods, while the latter provides a theoretical performance guarantee for a well defined and low complexity greedy cache allocation.

6.2 Future Work Suggestions

The PhD for an engineering student usually starts with some quite general question regarding some interesting application. Then, depending on (a): the knowledge of the student, (b): the natural difficulty of the problem and (c): the depth that the supervisor-student pair wants to reach in terms of understanding and results, the corresponding work will be produced. In a three years course however, you are entitled to answer only

some of the questions that others or you posed to yourself. The problem we dealt for the last three years has still many open theoretical and practical questions. We have already worked on some of them and have some results, while others are simply interesting ideas we did not really manage to have progress. For clarity I will enumerate them as follows

1. **Joint Caching and Recommendation:** This problem can indeed be attacked by many different directions. Initially, one can use the MDP framework in order to jointly consider the policy design of caching and recommendations. However this problem in its brute version is computationally pointless to attack for practical scenarios. However, if one systematically and controllably constrains this exploding state and action space could find the global optimal of the joint problem.

2. **Model-Free User Behavior:** This is a direction for which we already have some preliminary results. In essence, it would be interesting to formalize an optimization problem for which the user behavior is *not mathematically modeled, but it is rather learnt* by some agent who is trying to learn the user behavior and also exploit it at the same time.

3. **Dynamic Behavior:** Throughout this work, we have discussed about users whose behaviors and preferences stay static during the course "of the day" (the day is assumed to be the long time scale). However this leaves space in order to ask: "What if the recommender system actually affects the user preferences?" If so, how could a network friendly recommender drive the user preferences?

4. **Joint QoE and QoR Modeling:** Up to now, a recommender's performance is assessed as effective if it manages to predict accurately the user ratings for some content. However, an interesting implication of the "Recommender Over Wireless" (that we did not explicitly model or used throughout this thesis), is that essentially in the wireless setup there is a second dimension to take into account. When suggesting a content that will be very interesting to the user, but a content (s)he will not be able to stream in good quality it is evident that his *feedback ratings might significantly differ*. All in all, we claim that recommender over wireless is an area on its own which has not been investigated theoretically (modeling/optimization etc.). Interestingly some system works that collect measurements in order to jointly measure the QoE = QoS + QoR have peen performed recently in [70] and we firmly believe, that results from there could be an inspiration for new theoretical works on the topic.

Appendices

.1 Appendix A

Example. Assume for a moment a very simple scenario where $u_{ij} \in \{0, 1\}$ and that $c_i \in [0, 1]$. The recommender is obliged to offer $N = 2$ contents and 1 of them *has to be related*. The user randomly clicks one of the two contents always. Then a policy with no vision would give you *always* one related and the least cost item. We have $K = 8$ and $c_7 = \min\{c_1, ..., c_8\}$. When we are at content #5, we suggest file #4, as it is a related file to #5 and obviously #7. More importantly, $u_{81} = u_{82} = 1$, and #1, #2 are of very low cost. Interestingly, suggesting #8 instead of #7 leads to decreased expected cost for a walk starting from #5 (for appropriately selected values of the costs) assuming that the user requests only two contents. Hence, myopic strategies might (and most likely will) fail to explore the smarter actions, and potentially lead the user to "no man's land", i.e., a costly neighborhood of contents. For more details see the Appendix.

Proof. Assume that $K = 8$ and the nonzero u values are $u_{12} = u_{28} = u_{43} = u_{32} = u_{56} = u_{58} = u_{64} = u_{76} = 1$. We only check the expected cost of starting from #5 and doing exactly two steps under the model we gave. The fixed policies are the following, 6: $\{4, 7\}$, 7: $\{1, 6\}$, 8: $\{1, 7\}$.

Then we evaluate the expected cost for two different policies for content #5. A myopic approach $\mathcal{N}_5^{MYO} = \{6, 7\}$ and a policy with some vision $\mathcal{N}_5^{VIS} = \{6, 8\}$. Thus, starting from #5 and following $\mathcal{N}_5^{MYO}$ returns the following average cost.

$$E_5^{MYO} = \frac{1}{2}(c_6 + \frac{1}{2}(c_7 + c_4)) + \frac{1}{2}(c_7 + \frac{1}{2}(c_1 + c_6)) \tag{1}$$

and following $\mathcal{N}_5^{VIS} = \{6, 8\}$

$$E_5^{VIS} = \frac{1}{2}(c_6 + \frac{1}{2}(c_7 + c_4)) + \frac{1}{2}(c_8 + \frac{1}{2}(c_1 + c_7)) \tag{2}$$

The first parts of the cost are the same, for the rest and if we set $c_8 = c_7 + \epsilon$, we begin by $E_5^{MYO} \geq E_5^{VIS}$

$$c_7 + \frac{1}{2}(c_1 + c_6) \geq c_7 + \epsilon + \frac{1}{2}(c_1 + c_7) \Rightarrow \tag{3}$$

$$\frac{1}{2}c_6 \geq \epsilon + \frac{1}{2}c_7 \tag{4}$$

which in words translates to "if c_6 is larger than c_7 plus some constant, the acting myopically incurs a larger cost."

□

.2 Appendix B

We introduce K auxiliary variables $\mathbf{z}^T$, for which we will demand $\mathbf{z}^T = \mathbf{p}_0^T \cdot (\mathbf{I} - \alpha \cdot \sum_{n=1}^{N} v_n \mathbf{R}^n)^{-1}$. This introduces K new equality constraints, leading to the following equivalent problem.

Intermediate Step (Equivalent formulation)**.**

$$\underset{\mathbf{z},\ \mathbf{R}^1,\ldots,\mathbf{R}^N}{\text{minimize}} \quad \mathbf{c}^T \cdot \mathbf{z}, \tag{5}$$

$$\text{subject to} \quad \sum_{j=1}^{K}\sum_{n=1}^{N} v_n \cdot r_{ij}^n \cdot u_{ij} \geq q \cdot q_i^{max},\ \forall\ i \in \mathcal{K} \tag{6}$$

$$\sum_{j=1}^{K} r_{ij}^n = 1,\ \forall\ i \in \mathcal{K} \text{ and } n = 1, ..., N \tag{7}$$

$$\sum_{n=1}^{N} r_{ij}^n \leq 1,\ \forall\ \{i,j\} \in \mathcal{K} \tag{8}$$

$$0 \leq r_{ij}^n \leq 1\ (i \neq j),\ \ r_{ii}^n = 0\ \forall\ i,\ n. \tag{9}$$

The new objective is now convex (in fact, linear) in the new variable ($\mathbf{z}$). However, as the set of constraints Eq.(3.14) are all *quadratic equalities*, the problem remains nonconvex. To further deal with this additional complication, we define another set of variables as $f_{ij}^n = z_i \cdot r_{ij}^n$. Since the j-th element of the n-th vector $\mathbf{z}^T \cdot \mathbf{R}^n$ can be written as $\sum_i z_i \cdot r_{ij}^n$, we can write now $\mathbf{z}^T \cdot \mathbf{R}^n = \mathbf{1}^T \cdot \mathbf{F}^n$, and the new variables are $\mathbf{z}$ and $\mathbf{F}^1, \ldots, \mathbf{F}^N$, which are a $K \times 1$ vector, and N $K \times K$ matrices respectively.
In **OP .2**, we have a set of constraints which we should take care of for the new problem formulation. Eq.(6) becomes

$$\sum_{j=1}^{K}\sum_{n=1}^{N} v_n \cdot r_{ij}^n \cdot u_{ij} \geq q \cdot q_i^{max} \Rightarrow \sum_{j=1}^{K}\sum_{n=1}^{N} v_n \cdot f_{ij}^n \cdot u_{ij} - z_i \cdot q \cdot q_i^{max} \geq 0 \tag{10}$$

Eq.(7) becomes

$$\sum_{n=1}^{N} r_{ij}^n \leq 1 \Rightarrow \sum_{j=1}^{K} f_{ij}^n - z_i = 0 \tag{11}$$

Eq.(8) becomes

$$\sum_{j=1}^{K} r_{ij}^n \leq 1 \Rightarrow \sum_{j=1}^{K} f_{ij}^n - z_i \leq 0 \tag{12}$$

Therefore the final problem yields

OP 15 (LP formulation)**.**

$$\underset{\mathbf{z},\ \mathbf{F}^1,..,\mathbf{F}^N}{\text{minimize}} \quad \mathbf{c}^T \cdot \mathbf{z}, \tag{13}$$

$$\text{subject to} \quad \sum_{j=1}^{K}\sum_{n=1}^{N} v_n \cdot f_{ij}^n \cdot u_{ij} \geq z_i \cdot q \cdot q_i^{max},\ \forall\ i\ \in \mathcal{K} \tag{14}$$

$$\sum_{j=1}^{K} f_{ij}^n = z_i,\ \forall\ i\ \in \mathcal{K} \text{ and } n = 1,..,N \tag{15}$$

$$\sum_{n=1}^{N} f_{ij}^n \leq z_i,\ \forall\ \{i,\ j\} \in \mathcal{K} \tag{16}$$

$$f_{ij}^n \geq 0\ (i \neq j),\ \ f_{ii}^n = 0, \forall\ i, j\ \in\ \mathcal{K} \tag{17}$$

$$z_j - \alpha \cdot \sum_{n=1}^{N} v_n \cdot \sum_{i}^{K} f_{ij}^n = p_j,\ \forall j \in \mathcal{K} \tag{18}$$

.3 Proof of Corollary 5.3.1

Sub-case 1. Since the exact per-user utilities are not known, we calculate the SCHR given in Eq. (5.1) by taking the conditional expectations on $F_{kn}(x)$. Denoting the corresponding pdf as $f_{kn}(x)$, we proceed as follows:

$$\begin{aligned}
SCHR &= \sum_{i=1}^{N}\sum_{k=1}^{K} p_k^i \cdot q_i \cdot E\left[\left(1 - \prod_{n=1}^{K}\left(1 - u_{kn}^i \cdot x_n\right)\right)\right] \\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} p_k^i \cdot q_i \cdot \left(1 - E\left[\prod_{n=1}^{K}\left(1 - u_{kn}^i \cdot x_n\right)\right]\right) \\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} p_k^i \cdot q_i \cdot \left(1 - \prod_{n=1}^{K} E\left[\left(1 - u_{kn}^i \cdot x_n\right]\right)\right) \\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} p_k^i \cdot q_i \cdot \left(1 - \prod_{n=1}^{K} \int (1 - t \cdot x_n) \cdot f_{kn}(t)dt\right) \\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} p_k^i \cdot q_i \cdot \left(1 - \prod_{n=1}^{K}\left(1 - \left(\int t \cdot f_{kn}(t)dt\right) \cdot x_n\right)\right) \\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} p_k^i \cdot q_i \cdot \left(1 - \prod_{n=1}^{K}\left(1 - E[u_{kn}^i] \cdot x_n\right)\right)
\end{aligned}$$

where (i) the third equation holds since the utilities for different content pairs {k,n} are independent, and thus the expectation of their product is equal to the product of their expectations, and (ii) we denoted

$$E[u_{kn}^i] \equiv \int t \cdot f_{kn}(t)dt = \int (1 - F_{kn}(t))dt$$

and the above equation holds since u_{kn}^i is a positive random variable.

Sub-case 2 follows straightforwardly.

.4 Proof of Lemma 17

We prove here the NP-hardness of the optimal cache allocation for a single cache with soft cache hits. Let us consider an instance of Optimization Problem 11, where the utilities are equal among all users and can be either 1 or 0, i.e., $u_{kn}^i = u_{kn},\ \forall i \in \mathcal{N}$ and $u_{kn} \in \{0,1\},\ \forall k,n \in \mathcal{K}$. We denote as $\mathcal{R}_k$ the set of contents related to content k, i.e.

$$\mathcal{R}_k = \{n \in \mathcal{K} : n \neq k, u_{kn} > 0\} \ \text{(related content set)} \tag{19}$$

Consider the content subsets $\mathcal{S}_k = \{k\} \cup \mathcal{R}_k$. Assume that only content k is stored in the cache ($x_k = 1$ and $x_n = 0, \forall n \neq k$). All requests for contents in S_k will be satisfied (i.e. "covered" by content k), and thus SCHR will be equal to $\sum_{i\in\mathcal{N}} \sum_{n\in S_k} p_n^i \cdot q_i$. When more than one contents are stored in the cache, let $\mathcal{S}^{'}$ denote the union of all contents covered by the stored ones, i.e., $\mathcal{S}^{'} = \bigcup_{\{k:x_k=1\}} S_k$. Then, the SCHR will be equal to $\sum_{i\in\mathcal{N}} \sum_{n\in S'} p_n^i \cdot q_i$. Hence, the Optimization Problem 11 becomes equivalent to

$$\max_{\mathcal{S}^{'}} \sum_{n\in\mathcal{S}'} p_n^i \cdot q_i \qquad s.t. \quad |\{k : x_k = 1\}| \leq C.$$

This corresponds to the the *maximum coverage problem with weighted elements*, where "elements" (to be "covered") correspond to the contents $i \in \mathcal{K}$, weights correspond to the probability values $p_n^i \cdot q_i$, the number of selected subsets $\{k : x_k = 1\}$ must be less than C, and their union of covered elements is $\mathcal{S}^{'}$. This problem is known to be a NP-hard problem [99], and thus the more generic problem (with different u_{kn}^i and $0 \leq u_{kn} \leq 1$) is also NP-hard.

.5 Proof of Lemma 18

The objective function of Eq.(5.5) $f(X) : \{0,1\}^K \to \mathbb{R}$ is equivalent to a set function $f(S) : 2^{\mathcal{K}} \to \mathbb{R}$, where $\mathcal{K}$ is the finite *ground set* of contents, and $S = \{k \in \mathcal{K} : x_k = 1\}$. In other words,

$$f(S) \equiv \sum_{i=1}^{N} \sum_{k=1}^{K} p_k^i \cdot q_i \cdot \left(1 - \prod_{n\in S} \left(1 - u_{kn}^i\right)\right). \tag{20}$$

A set function is characterised as *submodular* if and only if for every $A \subseteq B \subset V$ and $\ell \in V \backslash B$ it holds that

$$[f(A \cup \{\ell\}) - f(A)] - [f(B \cup \{\ell\}) - f(B)] \geq 0 \tag{21}$$

From Eq.(5.5), we first calculate

$$
\begin{aligned}
f\left(A \cup \{\ell\}\right) - f\left(A\right) &= \sum_{i=1}^{N}\sum_{k=1}^{K} p_k^i q_i \left(1 - \prod_{n \in A \cup \{\ell\}} \left(1 - u_{kn}^i\right)\right) \\
&\quad - \sum_{i=1}^{N}\sum_{k=1}^{K} p_k^i q_i \left(1 - \prod_{n \in A} \left(1 - u_{kn}^i\right)\right) \\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} p_k^i \cdot q_i \cdot \left(\prod_{n \in A} \left(1 - u_{kn}^i\right) - \prod_{n \in A \cup \{\ell\}} \left(1 - u_{kn}^i\right)\right) \\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} p_k^i \cdot q_i \cdot \left(u_{k\ell}^i \cdot \prod_{n \in A} \left(1 - u_{kn}^i\right)\right).
\end{aligned}
$$

Then,

$$
\begin{aligned}
&\left[f\left(A \cup \{\ell\}\right) - f\left(A\right)\right] - \left[f\left(B \cup \{\ell\}\right) - f\left(B\right)\right] = \\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} p_k^i q_i \left(u_{k\ell}^i \prod_{n \in A} \left(1 - u_{kn}^i\right)\right) \\
&\quad - \sum_{i=1}^{N}\sum_{k=1}^{K} p_k^i q_i \left(u_{k\ell}^i \prod_{n \in B} \left(1 - u_{kn}^i\right)\right) \\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} p_k^i q_i \cdot u_{k\ell}^i \cdot \left(\prod_{n \in A} \left(1 - u_{kn}^i\right) - \prod_{n \in B} \left(1 - u_{kn}^i\right)\right) \\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} p_k^i q_i \cdot u_{k\ell}^i \cdot \prod_{n \in A} \left(1 - u_{kn}^i\right) \cdot \left(1 - \prod_{n \in B \setminus A} \left(1 - u_{kn}^i\right)\right)
\end{aligned}
$$

The above expression is always ≥ 0, which proves the submodularity for function f.

Furthermore, the function f is characterised as *monotone* if and only if $f(B) \geq f(A)$ for every $A \subseteq B \subset V$. In our case, this property is shown as

$$
\begin{aligned}
f(B) - f(A) &= \sum_{i=1}^{N}\sum_{k=1}^{K} p_k^i q_i \cdot \left(1 - \prod_{n \in B} \left(1 - u_{kn}^i\right)\right) \\
&\quad - \sum_{i=1}^{N}\sum_{k=1}^{K} p_k^i q_i \cdot \left(1 - \prod_{n \in A} \left(1 - u_{kn}^i\right)\right) \\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} p_k^i q_i \cdot \left(\prod_{n \in A} \left(1 - u_{kn}^i\right) - \prod_{n \in B} \left(1 - u_{kn}^i\right)\right) \\
&= \sum_{i=1}^{N}\sum_{k=1}^{K} p_k^i q_i \cdot \prod_{n \in A} \left(1 - u_{kn}^i\right) \cdot \left(1 - \prod_{n \in B \setminus A} \left(1 - u_{kn}^i\right)\right) \geq 0
\end{aligned}
$$

.6 Proof of Theorem 3

Following similar arguments as in the proof of Lemma 17, the Optimization Problem 12 can be shown to be equivalent to the *budgeted maximum coverage problem with weighted*

elements, which is an NP-hard problem [99].

In Algorithm 7, we first calculate a solution $S^{(1)}$ returned by a modified version (ModifiedGreedy) of the greedy algorithm (line 1). The differences between the greedy algorithm (e.g., Algorithm 6) and ModifiedGreedy, are that the latter: (a) each time selects to add in the cache the content that increases the most the fraction of the objective function over its own size (line 13), and (b) considers every content, until there is no content that can fit in the cache (lines 14-20). Then, Algorithm 7 calculates the solution $S^{(2)}$ that the greedy algorithm would return if all contents were of equal size (line 2). The returned solution, is the one between $S^{(1)}$ and $S^{(2)}$ that achieves a higher value of the objective function (lines 3-7).

Hence, Algorithm 7 is a "fast-greedy" type of approximation algorithm. Since, the objective function was shown to be submodular and monotone in Lemma 18, our fast greedy approximation algorithm can achieve a $\frac{1}{2} \cdot \left(1 - \frac{1}{e}\right)$-approximation solution (in the worst case), when there is a Knapsack constraint, using similar arguments as in [100].

.7 Proof of Lemma 20

Item (1): Optimization Problem 14 is of the exact same nature as Optimization Problem 13, so it follows that it is NP-hard.
Item (2): We proceed similarly to the proof of Lemma 18. The objective function of Eq. (5.17) $f(X) : \{0,1\}^{K \times M} \rightarrow \mathbb{R}$ is equivalent to a set function $f(S) : 2^{\mathcal{K} \times \mathcal{M}} \rightarrow \mathbb{R}$, where $\mathcal{K}$ and $\mathcal{M}$ are the finite *ground sets* of contents and SCs, respectively, and $S = \{k \in \mathcal{K}, j \in \mathcal{M} : x_{kj} = 1\}$:

$$f(S) \equiv \sum_{i=1}^{N} \sum_{k=1}^{K} p_k^i \cdot E_{G_i} \left[\max_{(n,j) \in S} \left(u_{kn}^i \cdot Q_{ij} \right) \right] \tag{22}$$

For all sets $A \subseteq B \subset V$ and {content, SC} tuples $(\ell, m) \in V \backslash B$, we get

$$\begin{aligned}
&f\left(A \cup \{(\ell,m)\}\right) - f\left(A\right) = \\
&= \sum_{i=1}^{N} \sum_{k=1}^{K} p_k^i \cdot E_{G_i} \left[\max_{(n,j) \in A \cup \{(\ell,m)\}} \left(u_{kn}^i Q_{ij} \right) \right] \\
&\qquad\qquad - \sum_{i=1}^{N} \sum_{k=1}^{K} p_k^i \cdot E_{G_i} \left[\max_{(n,j) \in A} \left(u_{kn}^i Q_{ij} \right) \right] \\
&= \sum_{i=1}^{N} \sum_{k=1}^{K} p_k^i \cdot E_{G_i} \left[R \left(u_{k\ell}^i \cdot Q_{im} - \max_{(n,j) \in A} \left(u_{kn}^i Q_{ij} \right) \right) \right]
\end{aligned}$$

where in the last equation we use the *ramp function* defined as $R(x) = x$ for $x \geq 0$ and $R(x) = 0$ for $x < 0$. Subsequently,

$$\begin{aligned}
&\left[f\left(A \cup \{(\ell,m)\}\right) - f\left(A\right)\right] - \left[f\left(B \cup \{(\ell,m)\}\right) - f\left(B\right)\right] = \\
&= \sum_{i=1}^{N} \sum_{k=1}^{K} p_k^i \cdot E_{G_i} \left[R \left(u_{k\ell}^i Q_{im} - \max_{(n,j) \in A} \left(u_{kn}^i Q_{ij} \right) \right) \right.
\end{aligned}$$

$$-R\left(u^i_{k\ell}Q_{im} - \max_{(n,j)\in B}\left(u^i_{kn}Q_{ij}\right)\right)\Big]$$

The above equation is always ≥ 0 (which proves that the objective function Eq.(5.17) is *submodular*), since the ramp function is monotonically increasing and comparing the two arguments of the function $R(x)$ in the above equation, gives

$$\begin{aligned} &u^i_{k\ell}Q_{im} - \max_{(n,j)\in A}\left(u^i_{kn}Q_{ij}\right) - \left(u^i_{k\ell}Q_{im} - \max_{(n,j)\in B}\left(u^i_{kn}Q_{ij}\right)\right) \\ &= \max_{(n,j)\in B}\left(u^i_{kn}Q_{ij}\right) - \max_{(n,j)\in A}\left(u^i_{kn}Q_{ij}\right) \geq 0 \end{aligned}$$

since B is a superset of A and therefore its maximum will be at least equal or greater than the maximum value in set A.

Similarly, since $A \subseteq B$ it holds

$$\begin{aligned} &f(B) - f(A) = \\ &= \sum_{i=1}^{N}\sum_{k=1}^{K} p^i_k \cdot E_{G_i}\left[\left(\max_{(n,j)\in B}\left(u^i_{kn}Q_{ij}\right) - \max_{(n,j)\in A}\left(u^i_{kn}Q_{ij}\right)\right)\right] \geq 0 \end{aligned}$$

which proves that the Eq.(5.17) is *monotone*.

Bibliography

[1] E. Bastug, M. Bennis, and M. Debbah, "Living on the edge: The role of proactive caching in 5g wireless networks," *IEEE Communications Magazine*, vol. 52, no. 8, pp. 82–89, 2014.

[2] C. V. networking Index, "Forecast and methodology, 2016-2021, white paper," *San Jose, CA, USA*, vol. 1, 2016.

[3] "What is 5g and why should lawmakers care?" https://www.washingtonpost.com/news/innovations/wp/2015/10/26/what-is-5g-and-why-should-lawmakers-care/?utm_term=.b66c4efb89b6.

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

[5] L. A. Adamic and B. A. Huberman, "Zipf's law and the internet." *Glottometrics*, vol. 3, no. 1, pp. 143–150, 2002.

[6] M. Leconte, G. Paschos, L. Gkatzikis, M. Draief, S. Vassilaras, and S. Chouvardas, "Placing dynamic content in caches with small population," in *Proc. IEEE Infocom*, 2016.

[7] S. Elayoubi and J. Roberts, "Performance and cost effectiveness of caching in mobile access networks," in *Proc. ACM ICN*, 2015.

[8] T. V. Doan, L. Pajevic, V. Bajpai, and J. Ott, "Tracing the path to youtube: A quantification of path lengths and latencies toward content caches," *IEEE Communications Magazine*, vol. 57, no. 1, pp. 80–86, 2018.

[9] F. Ricci, L. Rokach, and B. Shapira, "Introduction to recommender systems handbook," in *Recommender systems handbook*. Springer, 2011, pp. 1–35.

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

[11] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, no. 8, pp. 30–37, 2009.

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

[13] X. Cheng and J. Liu, "Nettube: Exploring social networks for peer-to-peer short video sharing," in *Infocom 2009, Ieee*. IEEE, 2009, pp. 1152–1160.

[14] S. Gupta and S. Moharir, "Modeling request patterns in vod services with recommendation systems," in *International Conference on Communication Systems and Networks*. Springer, 2017, pp. 307–334.

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

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

[17] "Netflix Open Connect," https://openconnect.netflix.com/en_gb/.

[18] "Quality of the Viewer Experience is Most Significant Factor in Viewer Engagement, Conviva Report Finds," https://www.conviva.com/press-releases/quality-viewer-experience-most-significant-factor-in-viewer-engagement-conviva-report-finds/.

[19] A. Ghosh, N. Mangalvedhe, R. Ratasuk, B. Mondal, M. Cudak, E. Visotsky, T. A. Thomas, J. G. Andrews, P. Xia, H. S. Jo, H. S. Dhillon, and T. D. Novlan, "Heterogeneous cellular networks: From theory to practice," *IEEE Comm. Magazine*, vol. 50, no. 6, pp. 54–64, 2012.

[20] J. G. Andrews, H. Claussen, M. Dohler, S. Rangan, and M. C. Reed, "Femtocells: Past, present, and future," *IEEE Journal on Selected Areas in Communications*, vol. 30, no. 3, pp. 497–508, 2012.

[21] J. G. Andrews, "Seven ways that hetnets are a cellular paradigm shift," *IEEE Communications Magazine*, vol. 51, no. 3, pp. 136–144, 2013.

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

[23] P. Sermpezis and T. Spyropoulos, "Effects of content popularity on the performance of content-centric opportunistic networking: An analytical approach and applications," *IEEE/ACM Transactions on Networking*, vol. 24, no. 6, pp. 3354–3368, 2016.

[24] B. Han, P. Hui, V. S. A. Kumar, M. V. Marathe, J. Shao, and A. Srinivasan, "Mobile data offloading through opportunistic communications and social participation," *IEEE Trans. on Mobile Computing*, 2012.

[25] P. Sermpezis and T. Spyropoulos, "Offloading on the edge: Performance and cost analysis of local data storage and offloading in HetNets," in *Proc. IEEE WONS*, 2017, pp. 49–56.

[26] J. Whitbeck, M. Amorim, Y. Lopez, J. Leguay, and V. Conan, "Relieving the wireless infrastructure: When opportunistic networks meet guaranteed delays," in *Proc. IEEE WoWMoM*, 2011.

[27] L. Vigneri, T. Spyropoulos, and C. Barakat, "Storage on Wheels: Offloading Popular Contents Through a Vehicular Cloud," in *Proc. IEEE WoWMoM*, 2016. [Online]. Available: https://hal.inria.fr/hal-01315370

[28] A. Sadeghi, G. Wang, and G. B. Giannakis, "Deep reinforcement learning for adaptive caching in hierarchical content delivery networks," *IEEE Transactions on Cognitive Communications and Networking*, 2019.

[29] G. S. Paschos, A. Destounis, L. Vigneri, and G. Iosifidis, "Learning to cache with no regrets," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2019, pp. 235–243.

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

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

[32] D. K. Krishnappa, M. Zink, and C. Griwodz, "What should you cache?: a global analysis on youtube related video caching," in *Proc. ACM NOSSDAV Workshop*, 2013, pp. 31–36.

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

[34] P. Sermpezis, T. Spyropoulos, L. Vigneri, and T. Giannakas, "Femto-caching with soft cache hits: Improving performance with related content recommendation," in *Proc. IEEE GLOBECOM*, 2017.

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

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

[37] T. Spyropoulos and P. Sermpezis, "Soft cache hits and the impact of alternative content recommendations on mobile edge caching," in *Proc. ACM CHANTS workshop*, 2016, pp. 51–56.

[38] K. Qi, B. Chen, C. Yang, and S. Han, "Optimizing caching and recommendation towards user satisfaction," in *10th International Conference on Wireless Communications and Signal Processing (WCSP)*. IEEE, 2018, pp. 1–7.

[39] K. Guo, C. Yang, and T. Liu, "Caching in base station with recommendation via q-learning," in *2017 IEEE Wireless Communications and Networking Conference (WCNC)*. IEEE, 2017, pp. 1–6.

[40] M. Karaliopoulos and I. Koutsopoulos, "Poster: Infrastructure and service provider games in crowdsourced networks," 2019.

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

[42] L. Song, C. Fragouli, and D. Shah, "Interactions between learning and broadcasting in wireless recommendation systems," in *2019 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2019, pp. 2549–2553.

[43] E. Ie, V. Jain, J. Wang, S. Narvekar, R. Agarwal, R. Wu, H.-T. Cheng, T. Chandra, and C. Boutilier, "Slateq: A tractable decomposition for reinforcement learning with recommendation sets," 2019.

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

[45] ——, "Optimal network-friendly recommendations for sequential access," *submitted to IEEE Transactions on Wireless Communications*.

[46] T. Giannakas, A. Giovanidis, and T. Spyropoulos, "Mdprecommend: Optimally reducing network cost via recommendations for random sessions," *somewhere*, 2019.

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

[48] K. Poularakis, G. Iosifidis, and L. Tassiulas, "Approximation algorithms for mobile data caching in small cell networks," *IEEE Trans. on Communications*, vol. 62, no. 10, pp. 3665–3677, 2014.

[49] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the web." Stanford InfoLab, Tech. Rep., 1999.

[50] M. Harchol-Balter, *Performance Modeling and Design of Computer Systems: Queueing Theory in Action*. Cambridge Univ. Press, 2013.

[51] K. Avrachenkov and D. Lebedev, "Pagerank of scale-free growing networks," *Internet Mathematics*, vol. 3, no. 2, pp. 207–231, 2006.

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

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

[54] S. Boyd *et al.*, "Distributed optimization and statistical learning via the alternating direction method of multipliers," *Foundations and Trends® in Machine Learning*, vol. 3, no. 1, pp. 1–122, 2011.

[55] J. Park and S. Boyd, "General heuristics for nonconvex quadratically constrained quadratic programming," *preprint arXiv:1703.07870*, 2017.

[56] M. Grant and S. Boyd, "Cvx: Matlab software for disciplined convex programming."

[57] N. Parikh, S. Boyd *et al.*, "Proximal algorithms," *Foundations and Trends® in Optimization*, vol. 1, no. 3, pp. 127–239, 2014.

[58] J. Duchi, S. Shalev-Shwartz, Y. Singer, and T. Chandra, "Efficient projections onto the l1-ball for learning in high dimensions," in *in Proc. ICML*, 2008.

[59] R. L. Dykstra, "An algorithm for restricted least squares regression," *Journal of the American Statistical Association*, vol. 78, no. 384, pp. 837–842, 1983.

[60] S. Boyd and J. Dattorro, "Alternating projections," *EE392o, Stanford University*, 2003.

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

[62] X. Su and T. M. Khoshgoftaar, "A survey of collaborative filtering techniques," *Adv. in Artif. Intell.*, pp. 4:2–4:2, 2009.

[63] T. Bertin-Mahieux, D. P. Ellis, B. Whitman, and P. Lamere, "The million song dataset," in *Proceedings of the 12th International Conference on Music Information Retrieval (ISMIR 2011)*, 2011.

[64] W. Gao, D. Goldfarb, and F. E. Curtis, "Admm for multiaffine constrained optimization," *Optimization Methods and Software*, pp. 1–47, 2019.

[65] Y. Wang, W. Yin, and J. Zeng, "Global convergence of admm in nonconvex nonsmooth optimization," *Journal of Scientific Computing*, vol. 78, no. 1, pp. 29–63, 2019.

[66] P. Sermpezis, T. Giannakas, T. Spyropoulos, and L. Vigneri, "Soft cache hits: Improving performance through recommendation and delivery of related content," *IEEE Journal of Selected Areas in Communications*, 2018.

[67] C. M. Grinstead and J. L. Snell, *Introduction to probability*. American Mathematical Soc., 2012.

[68] S. Ermon, C. P. Gomes, A. Sabharwal, and B. Selman, "Designing fast absorbing markov chains." in *Proc. AAAI*, 2014, pp. 849–855.

[69] Y. Wang, W. Yin, and J. Zeng, "Global convergence of admm in nonconvex nonsmooth optimization," *Journal of Scientific Computing*, pp. 1–35, 2015.

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

[71] "https://labrosa.ee.columbia.edu/millionsong/lastfm."

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

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

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

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

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

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

[78] F. Boccardi, R. Heath, A. Lozano, T. Marzetta, and P. Popovski, "Five Disruptive Technology Directions for 5G," *IEEE Comm. Mag. SI on 5G Prospects and Challenges*, 2014.

[79] M. A. Maddah-Ali and U. Niesen, "Fundamental limits of caching," *IEEE Trans. on Information Theory*, vol. 60, no. 5, pp. 2856–2867, May 2014.

[80] Y. C. Hu, M. Patel, D. Sabella, N. Sprecher, and V. Young, "Mobile edge computing a key technology towards 5g," *ETSI White Paper No. 11*, 2016.

[81] "T-Mobile Music Freedom," https://www.t-mobile.com/offer/free-music-streaming.html, 2017.

[82] Y. Yiakoumis, S. Katti, and N. McKeown, "Neutral net neutrality," in *Proc. ACM SIGCOMM*, 2016, pp. 483–496.

[83] D. Kahneman. New York: Farrar, Straus and Giroux, 2011.

[84] C. Liang and F. R. Yu, "Wireless network virtualization: A survey, some research issues and challenges," *IEEE Communications Surveys & Tutorials*, vol. 17, no. 1, pp. 358–380, 2015.

[85] G. Linden, B. Smith, and J. York, "Amazon.com recommendations: Item-to-item collaborative filtering," *IEEE Internet computing*, vol. 7, no. 1, pp. 76–80, 2003.

[86] S. Sesia, I. Toufik, and M. Baker, *LTE, The UMTS Long Term Evolution: From Theory to Practice*. Wiley Publishing, 2009.

[87] M. Dehghan, L. Massoulie, D. Towsley, D. Menasche, and Y. Tay, "A utility optimization approach to network cache design," in *Proc. IEEE INFOCOM*, 2016.

[88] A. Krause and D. Golovin, "Submodular function maximization," *Tractability: Practical Approaches to Hard Problems*, vol. 3, no. 19, p. 8, 2012.

[89] G. L. Nemhauser and L. A. Wolsey, "Best algorithms for approximating the maximum of a submodular set function," *Mathematics of operations research*, vol. 3, no. 3, pp. 177–188, 1978.

[90] M. Sviridenko, "A note on maximizing a submodular set function subject to a knapsack constraint," *Operations Research Letters*, vol. 32, no. 1, pp. 41–43, 2004.

[91] P. Sermpezis, T. Spyropoulos, L. Vigneri, and T. Giannakas, "Femto-caching with soft cache hits: Improving performance through recommendation and delivery of related content," *available at https://arxiv.org/abs/1702.04943*, 2017.

[92] G. Calinescu, C. Chekuri, M. Pál, and J. Vondrák, "Maximizing a monotone submodular function subject to a matroid constraint," *SIAM Journal on Computing*, vol. 40, no. 6, pp. 1740–1766, 2011.

[93] Y. Filmus and J. Ward, "Monotone submodular maximization over a matroid via non-oblivious local search," *SIAM Journal on Computing*, vol. 43, no. 2, pp. 514–542, 2014.

[94] "Internet.org by Facebook," https://info.internet.org/, 2017.

[95] "free basics by Facebook," https://0.freebasics.com/desktop, 2017.

[96] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel, "Image-based recommendations on styles and substitutes," in *Proc. ACM SIGIR*, 2015, pp. 43–52.

[97] K. Poularakis, G. Iosifidis, A. Argyriou, and L. Tassiulas, "Video delivery over heterogeneous cellular networks: Optimizing cost and performance," in *Proc. IEEE INFOCOM*, 2014, pp. 1078–1086.

[98] O. Fercoq, M. Akian, M. Bouhtou, and S. Gaubert, "Ergodic control and polyhedral approaches to pagerank optimization," *IEEE Trans. on Automatic Control*, vol. 58, no. 1, pp. 134–148, 2013.

[99] S. Khuller, A. Moss, and J. S. Naor, "The budgeted maximum coverage problem," *Information Processing Letters*, vol. 70, no. 1, pp. 39–45, 1999.

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