Algorithms and Cooperation Models in Caching and Recommendation Systems

Dissertation

submitted to

Sorbonne Université

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

Author:

Dimitra TSIGKARI

Scheduled for defense on May 25, 2022, before a committee composed of:

Reviewers

Prof.	**Viktoria FODOR**	KTH Royal Institut of Technology, Sweden
Prof.	**Gentian JAKLLARI**	Toulouse INP-ENSEEIHT, France

Examiners

Prof.	**Giuseppe BIANCHI**	University of Rome Tor Vergata, Italy
Prof.	**George IOSIFIDIS**	Delft University of Technology, Netherlands
Prof.	**Adlen KSENTINI**	Eurecom, France

Thesis Advisor

Prof.	**Thrasyvoulos SPYROPOULOS**	Eurecom, France

Algorithmes et Modèles de Coopération pour les Systèmes de Mise en Cache et de Recommandation

Thèse

soumise à

Sorbonne Université

pour l'obtention du Grade de Docteur

présentée par:

Dimitra TSIGKARI

Soutenance de thèse prévue le 25 mai 2022 devant le jury composé de:

Rapporteurs

Prof.	**Viktoria FODOR**	KTH École Royale Polytechnique, Suède
Prof.	**Gentian JAKLLARI**	Toulouse INP-ENSEEIHT, France

Examinateurs

Prof.	**Giuseppe BIANCHI**	Université de Rome Tor Vergata, Italie
Prof.	**George IOSIFIDIS**	Université de Technologie de Delft, Pays-Bas
Prof.	**Adlen KSENTINI**	Eurecom, France

Directeur de Thèse

Prof.	**Thrasyvoulos SPYROPOULOS**	Eurecom, France

Abstract

With on-demand video streaming services dominating today's Internet traffic, Content Providers (CPs) strive to offer high streaming quality to their users in order to keep them engaged with the service. At the same time, Content Delivery Network (CDN) providers aim to minimize costs related to the content delivery. In this setting, caching contents close to the users is crucial in order to address these objectives. Typically, caching decisions are based on the predicted content popularities or on the observed flow of content requests. However, these requests are heavily influenced by the recommendations that appear on each user's interface. Their primary role is to help users navigate through the ever-growing catalog of available contents by suggesting the ones that are close to each user's tastes. Due to their ability to shape requests, recommendations have become a powerful tool for CPs who invest financial and research resources to improve them and increase user engagement and revenues.

It is clear that both the caching allocation and the recommendation policy have an impact on the user satisfaction and financial implications for the CP and the CDN. Although caching and recommendations are traditionally decided independently of each other, the idea of co-designing these decisions has been recently proposed in the literature as a way to minimize delivery costs and traffic at the backbone Internet. This thesis follows this direction of exploiting the interplay of caching and recommendations in the setting of streaming services. It approaches the subject through the perspective of the users, and then from a network-economical point of view.

First, we study the problem of jointly optimizing caching and recommendations with the goal of maximizing the overall experience of the users. This joint optimization is possible today for CPs that simultaneously act as CDN owners, implying that the same entity may handle both caching and recommendation decisions. In future wireless networks, CPs who lease or own a slice at the edge of the network could also control both decisions. We introduce the metric of users' streaming experience as a balanced sum of streaming quality (affected by caching decisions) and recommendation quality (determined by recommendation decisions). This is based on findings according to which users value both the content itself and the streaming quality in which it is delivered. We then model the joint optimization of caching and recommendations as a maximization problem of this metric. Although we show that this problem is NP-hard, through a careful analysis, we prove that it is submodular. Using this property, we provide the first approximation algorithm for the joint problem. Indeed, we propose a polynomial-time algorithm that has $\frac{1}{2}$-approximation guarantees (or $\frac{1-1/e}{2}$ in the case of contents of heterogeneous sizes).

Our numerical evaluations validate these theoretical guarantees and show that a better approximation ratio is achieved in practice for some problem instances.

We then study the case where recommendations and caching are decided by two separate entities (the CP and the CDN, respectively) who want to maximize their individual profits. In particular, adjusting recommendations to favor cached items could greatly benefit the CDN by reducing content delivery costs. However, this approach might lead to recommendations for contents less relevant to the users, and could potentially negatively affect user engagement, and thus, the CP's revenues. We study the arising financial tradeoffs for the two entities as a result of recommendations and caching. Based on tools from game theory and optimization theory, we propose a novel cooperation mechanism between the two entities on the grounds of recommendations. This cooperation allows them to design a cache-friendly recommendation policy that assures a fair split of the resulting gains. We also discuss how the proposed scheme could be extended in order to include caching decisions in this cooperation. We show through simulations that the proposed mechanism can lead to an important increase in profit for both parties.

Abrégé

Les services de streaming vidéo à la demande dominent le trafic Internet d'aujourd'hui et les fournisseurs de contenu (CPs) s'efforcent d'offrir une haute qualité de streaming à leurs utilisateurs afin de les maintenir engagés avec le service. En même temps, les fournisseurs du réseau de diffusion de contenu (CDNs) visent à minimiser les coûts liés à la distribution du contenu. Dans ce contexte, la mise en cache des contenus à proximité des utilisateurs est cruciale pour répondre à ces objectifs. En règle générale, les décisions de mise en cache sont basées sur les popularités prévues de contenu ou sur le flux observé de demandes de contenu. Cependant, ces demandes sont fortement influencées par les recommandations qui apparaissent sur l'interface de chaque utilisateur. Leur rôle principal est d'aider chaque utilisateur à découvrir les contenus du catalogue et suggérer ceux qui sont susceptibles de les intéresser. Comme ces recommandations influencent les demandes de contenu, elles sont devenues un outil puissant pour les CPs qui investissent des ressources financières et de recherche pour les améliorer et augmenter l'engagement des utilisateurs et les revenus.

Il est clair que l'allocation de la mise en cache et la politique de recommandation ont un impact sur la satisfaction des utilisateurs, ainsi que des implications financières pour le CP et le CDN. Bien que la mise en cache et les recommandations soient traditionnellement décidées indépendamment les unes des autres, l'idée de concevoir ensemble ces décisions a été récemment proposée dans la littérature comme un moyen de minimiser les coûts de distribution et le trafic Internet. Cette thèse suit cette direction d'exploitation de la codépendance de la mise en cache et des recommandations dans le cadre des services de streaming. Elle aborde le sujet du point de vue des utilisateurs, puis d'un point de vue économique.

Dans un premier temps, nous étudions le problème de l'optimisation simultanée de la mise en cache et des recommandations dans le but de maximiser l'expérience globale des utilisateurs. Cette optimisation simultanée est possible aujourd'hui pour les CPs qui possèdent leur propre CDN, ce qui implique que la même entité peut gérer à la fois les décisions de mise en cache et de recommandation. Dans les futurs réseaux sans fil, les CPs qui louent ou possèdent une tranche virtuelle du réseau pourraient également contrôler les deux décisions. Nous introduisons la métrique de l'expérience de streaming des utilisateurs comme une somme équilibrée de la qualité du streaming (affectée par les décisions de mise en cache) et de la qualité des recommandations (déterminée par les décisions de recommandation). Ceci est basé sur des résultats expérimentaux selon lesquels les utilisateurs apprécient à la fois le contenu lui-même et la qualité de streaming

dans laquelle il est diffusé. Nous modélisons ensuite l'optimisation de la mise en cache et des recommandations comme un problème de maximisation de cette métrique. Bien que nous montrions que ce problème est NP-difficile, notre analyse nous permet de prouver qu'il est sous-modulaire. En utilisant cette propriété, nous fournissons un algorithme d'approximation pour le problème. En effet, nous proposons un algorithme en temps polynomial qui a des garanties d'approximation $\frac{1}{2}$ (ou $\frac{1-1/e}{2}$ dans le cas de contenus de tailles hétérogènes). Nos évaluations numériques valident ces garanties théoriques et montrent qu'un meilleur rapport d'approximation est atteint pour certaines instances du problème.

Nous étudions ensuite le cas où les recommandations et la mise en cache sont décidées par deux entités distinctes (le CP et le CDN, respectivement) qui veulent maximiser leurs profits individuels. En particulier, l'adaptation des recommandations pour favoriser les éléments en cache pourrait grandement bénéficier le CDN en réduisant les coûts de diffusion du contenu. Cependant, cette approche pourrait conduire à des recommandations de contenus moins pertinentes pour les utilisateurs, et pourrait potentiellement affecter négativement l'engagement des utilisateurs, et donc, les revenus du CP. Nous étudions les aspects financiers des recommandations et de la mise en cache pour les deux entités. Sur la base d'outils issus de la théorie des jeux et de la théorie de l'optimisation, nous proposons un nouveau mécanisme de coopération entre les deux entités sur la base de recommandations. Cette coopération leur permet de concevoir une politique de recommandation qui favorise les contenus en cache et qui assure une répartition équitable des gains qui en résultent. Nous discutons également de la manière dont le schéma proposé pourrait être étendu afin d'inclure les décisions de mise en cache dans cette coopération. Nous montrons à travers des simulations que le mécanisme proposé peut conduire à une augmentation importante du profit pour les deux parties.

Acknowledgements

First of all, I would like to thank my supervisor Thrasyvoulos Spyropoulos for guiding me through this thesis. I am grateful for everything I have learned from him, for our discussions, and for giving me the opportunity to work on such an appealing topic. I would also like to thank George Iosifidis for collaborating with me and sharing his insights.

I want to thank my friends in France and Greece for all the great moments we shared, and a special thanks goes to Gui, Karyna, Vitalii, Flavio, and Katerina.

I could not have arrived at this point of defending my thesis without a well-oiled support mechanism that was present since my early years at school and, then, throughout my studies at university. I am extremely grateful for the constant support, encouragement, love, and generosity of my family: my parents, my brother, my aunt, and my uncle. I dedicate this thesis to them. Finally, the person who played a vital role during the thesis duration (the Arkenstone of the support mechanism) was Fionn. I thank him for his love, patience, and for making my life better.

Contents

Abstract i
Abrégé [Français] iii
Acknowledgements v
Contents vii
List of Figures viii
List of Tables xi
Acronyms xiii

1 Introduction **1**
1.1 Video Streaming Services 1
1.2 Caching Systems 3
1.2.1 The Present and the Future of Content Delivery 3
1.2.2 Caching Policies and Related Work 6
1.3 Recommendations in Video Streaming Services 7
1.3.1 Recommender Systems 8
1.3.2 Recommendation Decisions 9
1.4 Interweaving Caching and Recommendation Decisions 10
1.4.1 The Benefits 12
1.4.2 Related Work 13
1.5 Thesis Contributions and Outline 15
1.5.1 The Joint Problem (Chapter 2) 15
1.5.2 Recommendations as a Means of Cooperation (Chapter 3) 16

2 An Approximation Algorithm for the Joint Problem **19**
2.1 Introduction 19
2.2 Problem Setup 20
2.2.1 Caching Network 20
2.2.2 Recommendations and User Model 21
2.2.3 Example 22
2.2.4 Metric of Streaming Experience (MoSE) 22
2.2.5 Joint Caching and Recommendations (Toy Example) 26

2.3 Problem Formulation and Analysis . . . 29
2.3.1 Intuition on Joint Optimization . . . 29
2.3.2 Towards Efficient Algorithms . . . 30
2.3.3 MoSE Algorithms and Guarantees . . . 36
2.3.4 The Single-Cache Case . . . 39
2.4 Performance Evaluation . . . 40
2.4.1 Scenario 1 . . . 41
2.4.2 Scenario 2 . . . 43
2.4.3 Scenario 3 . . . 46
2.5 Related Work . . . 48
2.6 Conclusion . . . 49

3 Models for Cooperative Recommendations **51**
3.1 Introduction . . . 51
3.2 Problem Setup . . . 53
3.2.1 Recommendations, Content Requests, and Caching . . . 53
3.2.2 Revenue/Cost Models and Utility Functions . . . 54
3.2.3 Towards Cooperative Decisions . . . 57
3.2.4 Toy Example . . . 57
3.3 Problem Formulation and Algorithms . . . 59
3.3.1 Centralized Cooperative Recommendations . . . 59
3.3.2 Distributed Cooperative Recommendations with Minimum Information Sharing . . . 61
3.3.3 Ensuring High Quality of Cooperative Recommendations . . . 63
3.4 Extension to Caching Decisions . . . 64
3.5 Performance Evaluation . . . 68
3.5.1 Scenario I . . . 68
3.5.2 Scenario II . . . 74
3.6 Related Work . . . 75
3.7 Conclusion . . . 76

4 Conclusions and Perspectives **77**

Bibliography **81**

List of Figures

1.1 Three paradigms of content delivery. 4
1.2 Network-friendly recommendations: from the user's data to the user's interface. 9
1.3 Potential benefits of interweaving caching and recommendation decisions. 13

2.1 Illustration of the variables and parameters considered for the joint caching and recommendations problem in a network of caches. Detailed description of this example can be found in Section 2.2.3. 23
2.2 Toy example of Sec. 2.2.5. On the left: illustration of the network together with the caching and recommendation decisions made by policies A, C, and J. On the right: the matrix of content relevances/utilities per user. . 28
2.3 Scenario 1, SQ-RQ tradeoff points for some values of the parameters β and r_d. 43
2.4 Scenario 2, SQ-RQ tradeoff points. Comparison of our policy with a policy proposed in the literature (CAwR) and the baseline policy γ. 44
2.5 Scenario 3, MoSE versus β for different types of SQ ($s_{uj} = \psi(b_{uj})$) and RQ ($\varphi(r_{ui})$) values/functions. Comparison of our policy with the A-femto and C-femto policies. 47

3.1 The CP and CDN cooperate by agreeing on the recommendations the users receive. The incentives of the cooperation are provided by the reduced price the CP is charged for the content delivery and the resulting increase in the number of cache hits that leads to lower retrieval costs for the CDN. 52
3.2 Sources of revenue and cost for the CP and the CDN as described in Sec. 3.2.2. The delivery contract is the foundation of their economic relationship. 54
3.3 Toy example presented in Sec. 3.2.4. In this example, the CP-CDN cooperation leads to financial gains of 20% and 21% respectively. These gains derive from the discount on the delivery fees (for the CP) and the fetching/retrieval savings (for the CDN). 58
3.4 Illustration of the DCR algorithm's steps. Each entity solves its sub-problem (without sharing its utility function) based on the other's local solution and the dual variables. They communicate their local solutions to the cooperation mediator that updates and communicates the dual variables. 63

3.5 Scenario I: Relative increase in utility for the CP and the CDN, total cooperative gains, cache hit rate for recommendations, and quality of recommendations achieved through the proposed cooperation scheme (CCR algorithm) for different values of discount $\rho \in [0.05, 0.5]$. 69
3.6 Scenario I: Relative increase in utility for different discount values ρ and for different relative cache sizes $(1 - 30\%)$. 72
3.7 Scenario I: Relative increase in utility for different ranges of R_{ui} (CP's revenues per content) as obtained by the CCR algorithm. 73
3.8 Convergence of DCR algorithm in Scenario I: Primal residual, *i.e.*, $||\Psi^{(k)} - \widetilde{\Psi}^{(k)}||_F$, and suboptimality gap, *i.e.*, $|DO^{(k)} - p^*|/|p^*|$, versus the number of iterations for two different values of the penalty parameter q. 74
3.9 Scenario II: Relative gains in utility (for the two entities) and optimal objective function values achieved by the CCRCache and the CCR algorithms for different values of discount $\rho \in [0.05, 0.4]$. 75

List of Tables

2.1 Notation Summary for Chapter 2 24
2.2 Example functions/values for the MoSE components 27
2.3 State-of-the-art works on caching and/or recommendations 30
2.4 Approximation ratio $(f(X^*, Y^*)/OPT)$ 42
2.5 Execution Time (AVG) Per Policy 42

3.1 Notation Summary for Chapter 3 56
3.2 Relative gains obtained by DCR* and CCR 73

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.,* CP (Content Provider) and CPs (Content Providers). In most cases, the meaning of an acronym is indicated the first time that it is used.

5G	5th Generation
ADMM	Alternating Direction Method of Multipliers
AVG	(on) Average
BS	Base Station
CCR	Centralized Cooperative Recommendations
CCRCache	Centralized Cooperative Recommendations and Caching
CDN	Content Delivery Network
CP	Content Provider
CPU	Central Processing Unit
DASH	Dynamic Adaptive Streaming over HTTP
DCR	Distributed Cooperative Recommendations
Def.	Definition
Eq.	Equation
Fig.	Figure
Gb	Gigabytes
HTTPS	Hypertext Transfer Protocol Secure
ICN	Information Centric Networking
ILP	Integer Linear Program
IRM	Independent Reference Model
ISP	Internet Service Provider
LFU	Least Frequently Used
LRU	Least Recently Used
Mb	Megabytes
Mbps	Megabytes per second
MEC	Multi-access Edge Computing
MOS	Mean Opinion Score
MoSE	Metric of Streaming Experience
m-DMoSE	Multi-processor (distributed) version of the MoSE algorithm

s-MoSE	Size-aware MoSE algorithm
NBS	Nash Bargaining Solution
OTT	Over-the-Top (media service)
PoP	Point of Presence
QoE	Quality of Experience
QoS	Quality of Service
RAN	Radio Access Network
RS	Recommender System
RQ	Recommendations Quality
Sec.	Section
SQ	Streaming Quality
TLS	Transport Layer Security
w.l.o.g.	Without loss of generality

Chapter 1

Introduction

This introductory chapter describes the main ideas, the framework, and motivations of the problems studied in this thesis. We present an overview of relevant technologies and related work. Finally, we discuss the contributions of this thesis and give an outline of the next chapters.

1.1 Video Streaming Services

Video streaming services and, in particular, Over-the-Top (OTT) services, such as Netflix, YouTube, and Disney+, steadily gain ground in our leisure time. It has been reported that a Netflix subscriber/user spends on average 2 hours per day on the service [1]. But as multimedia technologies and wireless communications evolve, users' expectations increase. For example, there has been a constant increase in devices that support Ultra-High-Definition/4K streaming quality over the last 5 years [2]. Streaming a content of such high quality (bitrate) to a user increases the Internet traffic, especially considering that large Content Providers (CPs), like Netflix, have more than 200 million subscribers [3]. This is confirmed by measurements [4] which state that, in 2021, video streaming represented 54% of the total Internet traffic worldwide.

Due to the high demand for video streaming over the Internet or wireless networks coupled with bandwidth and physical layer limitations, during their viewing session, a user can experience phenomena such as image freezing, start-up delays, and so on. When this happens, the user is likely to abandon the viewing session or the service all together [5]. User abandonment has a direct financial impact on the CP, a long-term impact on brand image [6], and an increase in user churn. Indeed, according to estimates [7], a 1% abandonment rate can already lead to a loss in the CP's revenue of $85,000$ US dollars ($) per year from ad impressions. It is thus not surprising that CPs, involved entities in the content delivery, and the research community focus on quantifying these phenomena of streaming quality, and strive to find ways to eliminate them in practice. A term that is often used is Quality of Service (QoS), which refers to measurements of such phenomena and to the research domain that studies them. QoS usually concerns network-related measurements with typical examples being the bitrate, delays, throughput, and jitter [5]. Another term that is closely related to QoS is Quality of Experience (QoE). QoE attempts

to measure how the QoS is perceived by the user [8]. A common way to measure QoE is the Mean Opinion Score (MOS). However, in the past few years, several models have been proposed. For example, QoE can be modeled as a weighted sum of different qualitative and quantitative measurements in order to capture its multidimensional nature [9].

The performance of streaming services in terms of QoS and QoE metrics is closely related to network conditions and underlying policies. Traditionally, streaming services employ techniques/algorithms that control the flow of the playback. A commonly used technique is the Dynamic Adaptive Streaming over HTTP (DASH). In particular, every video is partitioned into segments (of a few seconds typically), and each segment is encoded into different bitrates (resolutions and formats). Given the network conditions and bandwidth capacity, the video player (of limited buffer size/capacity) fetches each segment at the appropriate bitrate/quality. Therefore, the player can switch to a segment of lower bitrate in order to avoid rebuffering (which is experienced as video freezing). There is a line of works in the literature that propose such schemes with a focus on QoS and/or QoE, that are based on various network conditions (*e.g.*, [10]). However, these network conditions heavily rely on the location where the video (or each segment) is stored/cached. Caching is crucial for content delivery (especially in the setting of video streaming) and can reduce latency and traffic at the backbone network [11]. Today's research proposes methods to optimize caching with a focus on QoS/QoE in the framework of video services or caching in conjunction with DASH schemes. We will elaborate on such frameworks in the next section.

As we explained, caching has an important impact on the streaming quality perceived by the user. But streaming quality is not the only aspect that users value in a streaming service. A recent consumer survey showed that the "recommendations menu" is one of the important attributes that makes a streaming service attractive to users [12]. This refers to the list of recommendations that appears when a user lands on the service's homepage or after watching a content (for example, in YouTube, the related videos section). In most of the cases, these recommendations are personalized and are the result of sophisticated algorithms that are called Recommender Systems (RSs). The role of these recommendations is to help users discover the ever-growing catalog and increase retention rates and revenue. Therefore, CPs invest financial and research resources in order to improve the RS and its accuracy. A notable example is the 1 million dollars Netflix Prize.

This thesis provides models and solution methods towards an orchestration of caching and recommendation decisions in the setting of streaming services. We focus on on-demand video streaming services (whose contents are static, and thus, cacheable) and we will use, hereafter, the terms streaming and OTT interchangeably when referring to such services. Of course, video and audio/music streaming services are very similar since both rely on streaming quality, and both offer recommendations to the users. Examples of audio/music streaming services are Spotify, Amazon Music, and Apple Music. Our framework (that we will develop in what follows) could, of course, be applied to these services. However, the problem is more interesting in the case of video services since streaming video in the highest possible quality on Netflix uses up to 7Gb per hour [13], while on Spotify only up to 144Mb per hour [14].

1.2 Caching Systems

In this section, we provide an overview of caching systems. We present today's caching systems and we discuss how caching has been envisioned for future wireless networks beyond 5G. Finally, we elaborate on different caching policies and we present the advances of the related work on the subject.

Before presenting today's caching architectures, it is important to mention the two main objectives of caching systems:

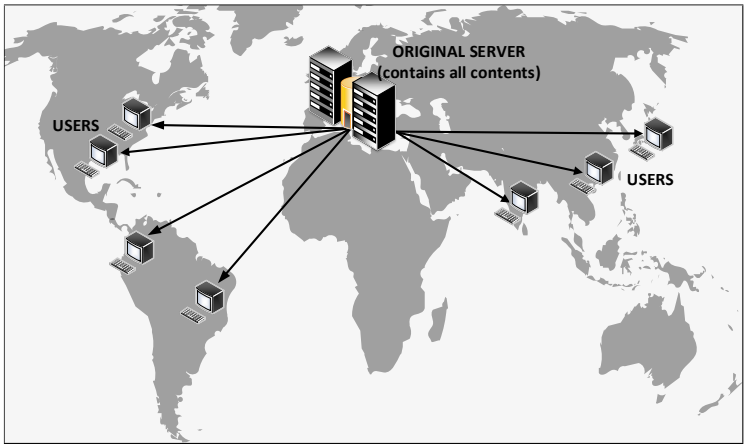
- maximizing efficiency while minimizing operating and capital expenditures;
- reducing latency (which can improve users' satisfaction).

These goals are combined with the fundamental challenge of caching systems, which is large catalogs of contents. This is even more pronounced in video streaming services since, as explained earlier, every content exists in several video/audio qualities. Therefore, in order to achieve the two goals listed above, it is crucial to correctly choose the cache allocation (*i.e.,* which contents are cached in the cache), design the caching policy (*i.e.,* how the cache is filled), and conceive the cache deployment (*i.e.,* how many caches should be deployed and in which architecture).

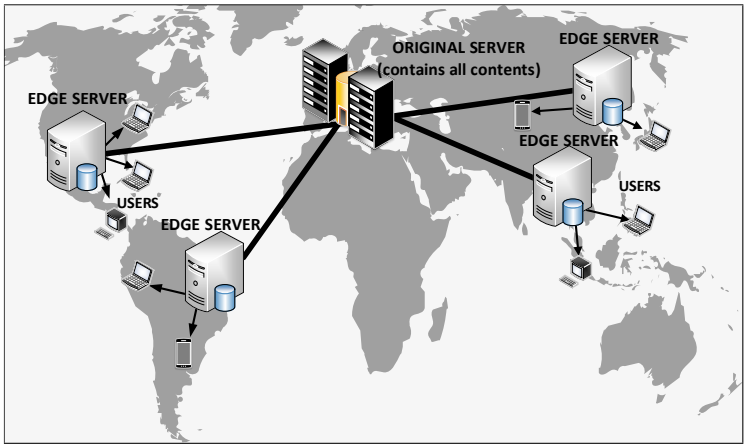
The most common metric to measure the performance of a cache (and, in extension, of a caching system) is the cache hit or, equivalently, the (cache) hit ratio. A cache hit is the event where the requested content is stored at the cache and, therefore, the hit ratio measures the fraction of the traffic that is served by the cache. Other metrics include throughput, response time/delay, or cost/price-related metrics.

1.2.1 The Present and the Future of Content Delivery

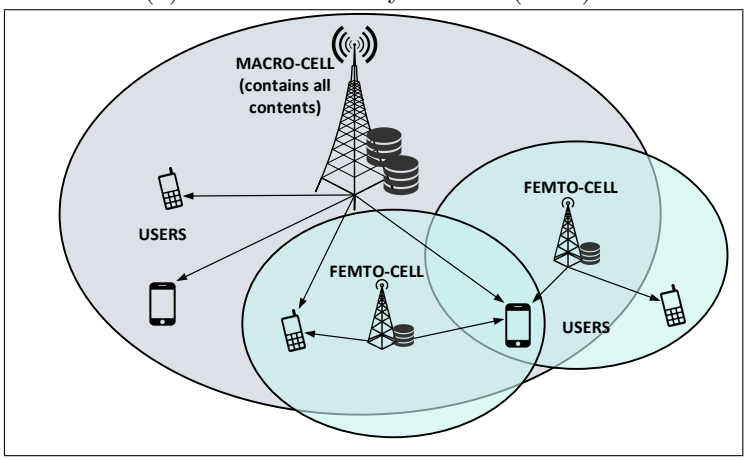
Today's Web contents are served through Content Delivery Networks (CDNs) which are large networks of servers/caches spread all over the world. The idea behind CDNs is to place contents in caches that are close to the users. This can reduce delivery latency by minimizing the physical hops in the network and distributing the traffic across multiple servers. This idea significantly improved the "client-server" paradigm of the early days of the Internet (see Fig. 1.1a). Before CDNs, all requests were directed to the CP's central server, where the entire catalog of contents were stored. This model turned out to be impractical since a burst of requests could greatly increase access delays and even cause the failure of the server [15]. On the other hand, in a CDN, upon request, the content is served by one of servers/caches (also called edge or surrogate servers) that are located in Points of Presence (PoPs) in the user's proximity. The edge servers are inter-connected through the backbone network as illustrated in Fig. 1.1b. When the requested content is not found in the edge server, an appropriate route to the original server (where all contents are stored) is selected. Some examples of today's CDNs are Akamai, Level 3, and Limelight. Akamai, in particular, is one of the leading CDNs today, and its network counts 365,000 servers in more than 135 countries [16]. When the content is not cacheable (live video streaming, gaming, virtual reality applications, etc.), the



(a) The client-server paradigm in the early days of the Internet.



(b) A content delivery network (CDN).



(c) The femto-caching paradigm for future wireless networks.

Figure 1.1: Three paradigms of content delivery.

CDN's role still remains crucial, as it provides the network for distributed request routing and good throughput of the servers [17].

Typically, a CP subscribes to a CDN through a delivery contract where the pricing depends on the size of the delivered content, the geographical location of the user, etc. [18]. Given that the CDN's network management has a financial impact on the CP (through its impact on the users' image of its service), as discussed in Sec. 1.1, it is not surprising that some CPs opt for a mutli-CDN approach, *i.e.,* they employ more that one CDN for redundancy and for distribution of the traffic load. Disney+, for example, employs 6 different CDNs for the delivery of its content, as reported in [19].

Large CPs have gone one step further by building their own CDN networks. This is, for example, the case for Netflix and YouTube that have built Open Connect and Google Global Cache, respectively [20, 21]. This seems to be the trend even outside the OTT industry since other providers, such as Facebook [22], own their in-house CDN. These CPs partner with Internet Service Providers (ISPs) and provide them with caches that are implemented within the ISP network [23]. Then, the CPs are responsible of filling their caches [24] and they decide how content requests are steered to caches based on the ISP's preferred paths and on their own routing algorithms [25, 26]. Such CPs, that decide to built their own CDN solutions, seek efficiency, backbone traffic reduction, performance, and flexibility [20].

It is clear that CPs strive for at-scale caching solutions, while at the same time there is an explosive growth of Internet traffic, especially through mobile applications. Therefore, different scenarios have been envisioned for caching in future wireless networks beyond 5G that attempt to address these issues (see [27] and references therein). These scenarios include caching at small base stations (BSs) in close proximity to the users [28] and/or caching at users' devices by leveraging device-to-device communications [29]. Moreover, caching with coded multicast transmissions (also called coded caching) has been proposed for future wireless communications [30, 31]. The main idea behind all the different scenarios is to bring the (cached) content even closer to the user. This could be perceived as a rather natural continuation after the client-server paradigm and the CDN architecture.

We will briefly describe one of the paradigms envisioned for mobile egde caching: the femto-caching paradigm that was presented in [28]. In this framework, caches are placed at small-cell base stations (also called helper nodes) where popular files are proactively stored (*e.g.,* during off-peak hours). If the requested content is not cached in any BS within the user's proximity, the file is downloaded via a macro-cellular BS that contains the entire catalog (see Fig. 1.1c). This approach alleviates the backhaul links and reduces latency through efficient transmissions of short distance. The idea is that BSs form a dense network with several BSs covering a single part of a city. BSs can overlap and the users that are located within the overlapping range can be served by either of the caches. Of course, this raises the question of optimal request routing. In such networks, the problem of caching and request routing can be seen as a problem on a bipartite graph.

Moreover, today's research focuses on Multi-access Edge Computing (MEC) servers, *i.e.,* BSs that contain communication, storage, and CPU resources. In this case, network slicing can provide a flexible resource allocation. A network slice is a virtual network

that is built on top of the physical network in a way that each slice tenant can manages its resources independently (see [32] and references therein). More specifically, the vision is that CPs will own their own network slice where they can decide on caching and also perform computational actions, *e.g.,* transcoding of contents [33]. This could be of great value for CPs who do not own (or cannot invest on building) a CDN network, but seek efficient and at-scale caching solutions.

1.2.2 Caching Policies and Related Work

Caching policies are divided into two main categories:

- dynamic, where the cache is typically updated upon a cache miss (or based on time-related criteria);
- proactive, where the cache is filled before the arrival of requests and the placement remains fixed for a certain time period.

Typical examples of dynamic policies are the Least-Frequently-Used (LFU) and Least-Recently-Used (LRU). In particular, upon a miss, LFU policy evicts the least frequently requested (used) content and, under this policy, the cache will eventually fill with the most popular contents (if we assume that the content popularities are constant over time). On the other hand, upon a miss, LRU evicts the content that is least recently requested. LFU has been proven to be optimal under the Independence Reference Model (IRM), which is a commonly-used model to describe the patterns of content requests that arrive at a cache. Essentially, IRM assumes that the contents' popularities are static, and thus, it ignores the temporal locality effect, *i.e.,* the fact that the popularity of a content could be boosted over short periods of time. Despite this property of LFU, it is mostly LRU and its variations that are commonly used in today's CDNs, since they come with a simpler implementation than LFU. In general, today's CDNs mostly employ dynamic caching policies (for example, this is the case for the Google Global Cache [34] and the Facebook CDN [22]).

Dynamic caching policies raise an interesting question in future wireless networks concerning the locality of requests. In particular, how good can the performance of the caching policy be, in a BS, when this is based only on the requests arriving locally, versus the global image of requests over the network (as seen by the macro-cell)? This is, for example, studied in [35], where it is shown that the global view on the requests can improve the cache hit rate. Moreover, rather than making assumptions on the request patterns, the authors in [36] consider a framework where the caching decisions are made in an adversarial environment, and they devise dynamic caching policies by employing the theory of online optimization.

Given the challenges met by dynamic policies in the framework of wireless communications, the approach of proactive caching is proposed and studied in several works, *e.g.,* [28, 37, 38], as a means to alleviate backhaul links during high peak traffic. Proactive caching is based on predicted content popularities where the filling of the cache(s) is completed during off-peak hours (*e.g.*, the early hours of the day). Predicting the content popularities can be also challenging. For example, the prediction can be based

on previously observed dynamics in the content requests patterns. In the framework of device-to-device caching/communications, [37] suggests a popularity prediction by leveraging information stemming from social media platforms.

Even though proactive caching is mostly studied in the scenario of future wireless networks beyond 5G, it seems that it has become a trend even in today's caching networks. Netflix, for example, fills/updates its caches once per day during off-peak hours [24]. In a blog post, Netflix claims that it can predict with high accuracy what their subscribers will watch [39]. Since its cache appliances have limited capacity, Netflix tackles the problem of various qualities available for every content by assigning a (predicted) popularity per pair (content, quality), rather than an overall popularity per content [40]. This seems to be effective in practice since it has been reported that, during prime time (the weekend), Netflix serves 63% of its traffic through its caches located within the ISPs' networks [19]. For this reason, in this thesis, we focus on proactive caching policies in the setting of today's or tomorrow's content delivery architectures. We believe that caching decisions can be further enhanced when taking into account (or combining with) the recommendations that the users receive. In the next section, we will elaborate on the impact of recommendations on content requests, especially on video streaming platforms.

Except for the aforementioned references, there are several works focusing on caching. A large body of works focus on video streaming services due to the percentage of Internet traffic they generate. For example, [41] studies the problem of maximizing the balanced sum of delay and incurred costs on a streaming service where the contents are encoded in several qualities. Another approach is to optimize caching decisions (for contents available in various qualities) in conjunction with DASH schemes, *e.g.,* as in [42]. A problem greatly studied is the joint optimization of caching and request routing since the caches can be of limited bandwidth. For example, this problem was the focus of [43, 44]. Finally, although less studied, the network-economic side of caching can provide useful insights on the management of caching networks. It has been studied both in the framework of CDNs [45] and of wireless networks. In the case of wireless networks, related work addresses interesting questions such as caching policies or leasing strategies of cache capacity by the CP [46–48].

1.3 Recommendations in Video Streaming Services

Recommendations are ubiquitous in today's Internet. They appear not only on platforms of streaming services, but also on other applications such as e-commerce. Their role is to assist the users' decision making (since the amount of available information can be overwhelming), and to help the users explore the available contents/items. Content providers employ recommendations in order to increase user satisfaction, retention rates, and, of course, revenues. The role of recommendations becomes more crucial in streaming services as the catalog of contents grows and users struggle to choose which content to watch[1].

[1]In particular, today's streaming services make efforts to deal with the users' "decision fatigue" and they are experimenting with new recommendation models (*e.g.,* the linear recommendation model that imitates the traditional TV program) [49, 50].

Typically, a CP recommends a list of contents to each user that is tailored to match her tastes. The size of this list can vary according to the device the user uses. As mentioned earlier, the list of recommendations is considered one of the important virtues of OTT streaming services (among streaming quality, cost, ease of use, etc.). In fact, recommendations shape content requests. On Netflix, for example, 80% of requests stem from the recommendations, while on YouTube this percentage can be up to 53% for the "related videos" section [51, 52].

In the remainder of this section, we will review the techniques of today's state-of-the-art recommender systems, and how their outcome can be employed by the CPs in order to incorporate network-related decisions.

1.3.1 Recommender Systems

The name recommender system (RS) refers to the sophisticated algorithms (and, in extension, to the domain of research that studies/devises these algorithms) that are used to issue recommendations. Recommendations can be personalized, *e.g.,* the recommendations that a Netflix subscriber sees on the service's homepage, or non-personalized, *e.g.,* the list of articles/highlights shown on a news webpage. The latter does not normally fall under the umbrella of RSs.

In their early days, RSs intended to predict the rating a user would give to a movie. For example, this prediction can be based on the ratings already assigned for movies previously watched, the ratings given by other users with similar tastes, and the nature of the contents themselves (*e.g.,* movie genres). Today's research on RSs not only focuses on how to ensure accurate predictions of ratings, but also on how to incorporate other properties in the recommendations. In particular, an RS usually consists of two steps:

1. rating prediction;
2. ranking generation.

The first step tries to predict the missing ratings based on a variety of data that the CP might collect. Examples of such data include existing ratings, contents' attributes, user behavior (abandonment, acceptance of recommendations, etc.), and account information [51, 53]. A variety of methods can be applied based on the available data, such as collaborative filtering, machine learning techniques, etc. [53, 54].

The second step can be seen as a refinement step (over the outcome of the first step) that assigns rankings[2] (or scores, or relevances) to the contents. In order to facilitate computations, often only a subset of contents is subject to the second step. When this is the case, this subset is selected based on the predicted ratings, *e.g.,* the first few hundred contents with the top ratings or the set of contents whose ratings is above a certain threshold. In the second step, the CP might want to control some properties/aspects of the recommendations. For example, it might want to ensure diversity, *i.e.,* recommend a variety of genres of movies rather than suggesting only movies of a certain kind, or

[2]Throughout this thesis, we will interchangebly use the terms utilities, relevances, rankings, and scores to describe these values.

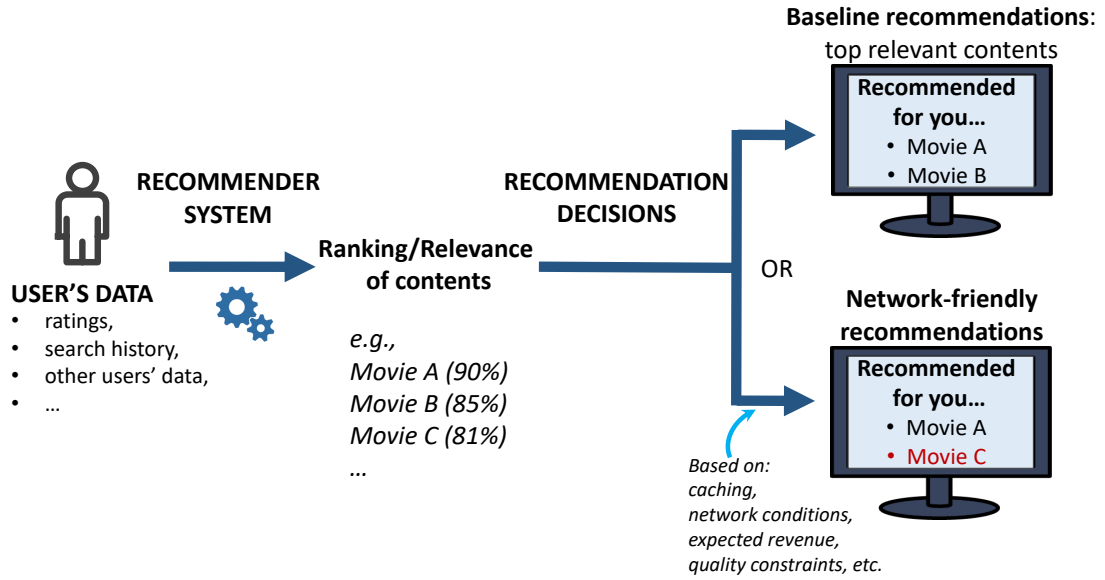


Figure 1.2: Network-friendly recommendations: from the user's data to the user's interface.

popularity, *i.e.*, steer the recommendations towards trending contents. It may also want to ensure freshness, that is, steer the recommendations towards newly available contents, or to incorporate information from social media platforms [55–57]. In practice, these desired properties can be more complex to describe and model. For example, in [58], the authors observe that, in e-commerce applications, most of the recommendation policies aim to minimize either the retailer's profit or the consumer's satisfaction. They suggest a scheme of issuing rankings for products with the objective of maximizing the total surplus, *i.e.*, the sum of the consumer's benefit and the retailer's benefit. Having identified the desired properties to be incorporated in the rankings, the ranking step is completed either by applying properly chosen ranking functions or through machine learning and optimization techniques [54–56].

The output of the RS is the set of rankings (either for the entire catalog or for a subset of them). Since the ranking step was based on the predicted ratings for each user, there is a set of rankings corresponding to each user. The rankings represent the utility or relevance of the contents to the specific user.

1.3.2 Recommendation Decisions

As we saw above, the outcome of an RS is an assignment of rankings to the contents, or to a subset of contents. On top of a rating-based model, these rankings incorporate some properties, as decided by the CP. In fact, these properties can be seen as biases that not only address the CP's needs, but also try to reflect the complex nature of human decision making. Throughout this thesis, we will refer to "recommendation decisions" to describe the decisions that can be made by the CP given the ranking of the RS and

before recommendations appear on the user's interface. A reasonable recommendation decision would be, for example, to recommend the N contents with the highest ranking (or expected revenue) for the user, where N is the number of recommendations to be listed. We will call them baseline recommendations. This list could be given in a decreasing order since it has been shown that the position of a content in the recommendations list has an impact on the user behavior [52].

Given the discussion in the previous sections, it is clear that caching and recommendations are inextricably linked when it comes to user satisfaction or the CP's revenues. This observation naturally leads us towards the so-called network-friendly or cache-friendly recommendations (see Fig. 1.2). More specifically, in this thesis, we claim that the recommendation decisions can be made in a way that they take into account the quality at which the recommended content can be delivered upon request or even economical factors (*e.g.,* expected revenue). The former could, for example, depend on the caching allocation or the network conditions. The idea is that one could steer the recommendations towards cached contents in order to ensure better streaming quality to the users. This can be also beneficial to the network itself since less traffic will be generated at the backbone network. We will elaborate in the next section about the benefits and the challenges of such an approach.

One could argue that the action of incorporating caching or network-related information could be performed during the ranking step of the RS. This might seem a good way to ensure that the network-friendly recommendations do not deviate too much from the user's preferences. However, caching or network management decisions might be made at different timescales than the RS. For example, the RS might be triggered every time the catalog is updated and/or the user gives feedback on a content she watched, while caching might take place every day or every few hours. Furthermore, we will see that it is possible to develop a framework where a potential deviation from the user's tastes can be controlled. A final argument about issuing network-friendly recommendations after running the RS concerns the computational complexity. In particular, network-related decisions can be complex and take into account various factors. On the other hand, the ranking vector that corresponds to each user (as issued by the RS) should be, in principle, sparse since only a subset of the catalog was subject to this step.

1.4 Interweaving Caching and Recommendation Decisions

In the previous section, we introduced the idea of nudging the recommendations towards cached items or items that can alleviate the network traffic. The idea of interweaving caching and recommendation decisions addresses the fact that these two decisions play an important role in the streaming services. The former has a strong impact on the streaming quality of the requested contents and the incurred delivery costs, while the latter shapes content requests and constitutes a significant feature of the service in the users' eyes. In this section, we will discuss the motivations for jointly designing caching and recommendation decisions, the possible ways to do so, as well as the potential benefits.

Aside from all the arguments presented earlier that support the co-design of caching

and recommendations, a strong motivation behind our work is the information we have gathered about real-world implementations and policies in streaming services. We provide here two examples that concern the Netflix service:

1. Netflix steers user recommendations towards popular (and possibly cached) contents. This is done, for example, through the "Trending Now" section of every user's main page. In particular, "Trending Now" recommendations include short-term popular contents "combined with the right dose of personalization" [51]. Meanwhile, at Netflix, caching decisions are based on (predicted) content popularities [39].

2. The Netflix mobile application recently introduced the feature "Downloads for You" [59]. When a user enables this feature, she allows Netflix to choose and proactively prefetch recommended contents on the user's device that the user can then watch while she is offline or over unstable Internet connections.

The first example could be seen as a hint that real-world systems might employ cache-friendly recommendations or recommendation-aware caching. However, the second example definitely illustrates a small-scale co-design of prefetching and recommendations. This further corroborates the problems that we will study in this thesis.

Caching and recommendation decisions can already be handled jointly in today's architectures, especially for the CPs who own CDN infrastructures, *e.g.,* Netflix and YouTube. The trend towards end-to-end network slicing in future wireless networks further supports such an approach. In this setting, handling caching and recommendations together seems appropriate, since each cache will cover fewer users (than today's CDN caches), and receive a limited amount of requests. This would render predicting content popularities, and thus, caching decisions particularly challenging [35].

Interweaving caching and recommendation decisions can be made in three main ways that we list and describe below:

- **Cache-friendly or network-friendly recommendations.** As the name suggests, this direction refers to recommendations that are modified when compared to the baseline recommendations (see Fig. 1.2). In particular, they are modified in order to incorporate information on network conditions or caching allocation or even financial considerations. In this framework, the caching and other related network conditions are supposed to be fixed. The main shortcoming of this approach is that the resulting recommendations depend on the other decisions made independently and ahead of time. For example, if the cache contains contents that are not close to the user's tastes, then the cache-friendly recommendations that this user will receive might be irrelevant to her preferences. This can severely affect the user satisfaction, and the user might abandon the service or request another content (*e.g.,* through the search bar). We will see that, in practice, the related work on cache-friendly recommendations tries to control this deviation from the user's tastes. This seems to be a good way to make this approach efficient in practice.

- **Recommendation-aware caching.** Acting in the inverse way when compared to the previous approach, this direction modifies caching decisions while taking into account

the recommendations that the users will receive. This approach is particularly useful for proactive caching policies. As we mentioned earlier, proactive policies are based on predicted content popularities. Through the framework of recommendation-aware caching, these predictions can rely on the recommendations (as issued from the baseline scheme, *i.e.,* recommending the top related contents to each user), as well as the user's behavior towards the recommendations, *e.g.,* tendency to request contents through the recommendations list. However, this approach can be less efficient in some scenarios: for example, when the content catalog is big, the caching concerns a large pool of users with diverse tastes, and the cache capacity is limited, then only contents with large aggregate popularity will be cached. These cached contents might feature in the recommendations list of only some users, without even being the top related contents. In this case, users might not request these contents, and the caching policy turns out to be inefficient.

- **Joint optimization of caching and recommendations.** In this approach, the decisions are taken in a way that caching allocation accounts for the recommendations and recommendations account for the caching allocation. In fact, the discussion on the other two approaches above revealed an important co-dependency of caching and recommendations: cache-friendly recommendations depend on caching allocation and risk their efficiency in practice, while recommendation-aware caching can be problematic when the users' tastes (and thus, the recommendations for these users) are diverging. Therefore, the joint approach seeks to take these decisions simultaneously. In optimization theory parlance, this means that we want to solve a problem where the control variables are caching and recommendations. This makes this approach more challenging, not only in terms of modeling, but also in terms of finding solutions. However, these challenges come with a great reward once overcome, since the joint approach is optimal when compared to the previous approaches. In fact, in Chapter 2, we provide an example where we illustrate that caching without taking into account the user recommendations and biasing recommendations later (to favor cached contents) is suboptimal.

1.4.1 The Benefits

The benefits of interweaving caching and recommendation decisions are multifold. Most importantly, **these benefits concern all the involved parties: the network, the users, and the CP**. The network here refers to the CDN (which can eventually be managed by the CP itself) and/or the ISP. As explained earlier, when a user clicks/selects one of the cache-friendly recommendations, this will lead to a cache hit at the cache nearby. This means that the cache can serve this content without fetching it from a server deep in the network, and without generating additional traffic at the backbone network. Therefore, extra retrieval costs will be avoided.

Regarding the benefits from the user's perspective, when her request is served by the cache nearby, the user can enjoy high QoS and QoE. This could translate to low start-up delays, high bitrate, etc. Moreover, when the cache-friendly recommendations do not deviate irreparably from her tastes, she can enjoy recommendations of high quality. In

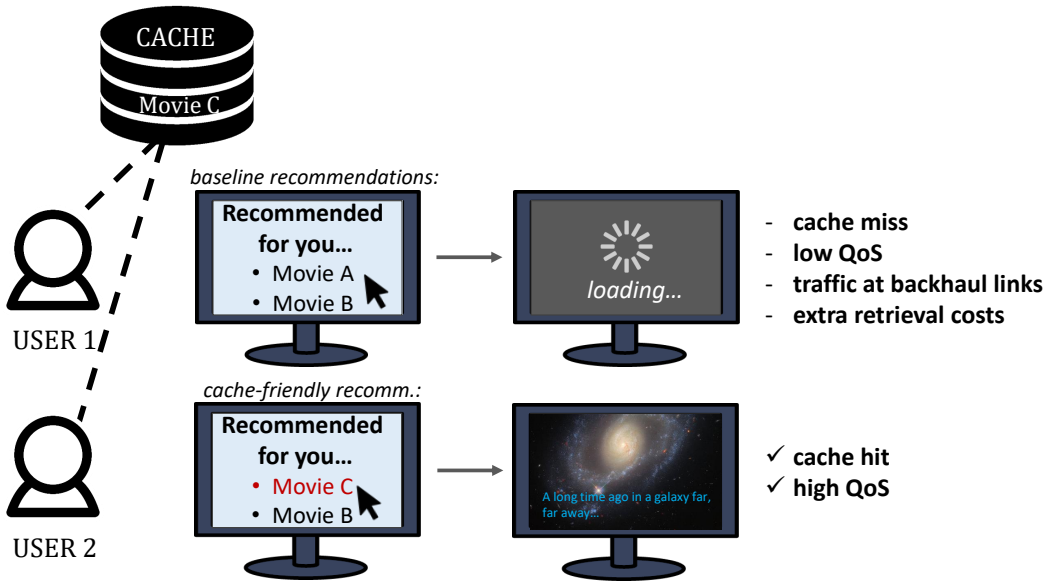


Figure 1.3: Potential benefits of interweaving caching and recommendation decisions.

fact, experiments have shown that the overall satisfaction of a user from an OTT service is greatly affected by both the streaming quality of the requested content and her interest in the content itself [60].

Furthermore, through a co-design of caching and recommendations, the CP can see its revenues significantly increasing due to user satisfaction. This is equivalent to user engagement, and thus, more ad impressions and/or less user churn. As we will see in Chapter 3, the CP could also explicitly incorporate financial factors when deciding on cache-friendly recommendations. The benefits of our approach are illustrated in Fig. 1.3, where User 1 receives the baseline recommendations (that were issued independently of the caching allocation), and User 2 receives cache-friendly recommendations.

Finally, the main challenges of interweaving caching and recommendations come from the challenges faced by caching systems and recommender systems, namely:

- limited cache capacity (combined with ever-growing content catalogs);
- recommendations quality, *i.e.,* offering recommendations to the users that are closely related to their tastes.

1.4.2 Related Work

To the best of our knowledge, the idea of co-designing caching and recommendations first appeared less than a decade ago following the increasing success of OTT streaming services. Nevertheless, it recently gained momentum with related works appearing in top-tier venues. In this section, we will review some of these works and discuss their contributions in the field. Additional references that are more specific to each chapter of the thesis will be provided at the end of each chapter.

In an early work in this direction [61], the authors propose heuristic algorithms for recommendations in peer-to-peer networks that take into account both service cost and user preferences. The authors in [62] study how content delivery costs can be reduced through proactive caching, and how a further reduction can be achieved by shaping the content requests. They propose to shape the requests, and thus, decrease the uncertainty of content requests by modifying the ratings (or "valuations") of the contents that appear to the users. Interestingly, [63] takes into account the impact of the positions of the contents in the recommendations list. In particular, the authors propose a reordering of the videos appearing in YouTube's related videos section by "pushing" the cached items to the top of the list. Adopting a similar "position-aware" approach, [64] presents a method of designing a cache-friendly recommendation policy where cached items are placed at the top of the recommendations list. The authors then evaluate this policy in the experimental testbed that they built, and provide useful insights. For example, they observe that the users tend to select the top recommendations, even if these are favoring cached items.

Another direction is to study sequential requests that are generated through recommendations. For example, such sequences are observed when a user watches several YouTube videos by clicking on the items of the related videos section. The works [65] and [66] study the problem of cache-friendly recommendations in the framework of sequential requests. More specifically, [65] formulates the problem of minimizing the fetching costs as a Markov chain, and proposes a heuristic solution. A similar probabilistic model to describe the users' behavior towards recommendations is adopted in [66]. The authors formulate the problem of devising cache-friendly recommendations as a Markov Decision Process, where the viewing session can be of arbitrary length.

As we explained earlier, the joint caching and recommendations problem can be considered one of the most interesting and challenging ones when compared to the approaches described earlier. It is thus not surprising that this topic has been tackled by a plethora of works, *e.g.,* [67–72]. The majority of these works make proactive caching decisions. More specifically, a decomposition algorithm for the joint problem is proposed in [67]. The authors formulate the problem as a cache hit rate maximization problem subject to a preference distortion threshold (that controls how the nudged recommendations deviate from the user's tastes). Their policy first decides on caching, accounting for the impact of recommendations, and then adjusts the recommendations in order to favor cached items. Similarly, [68] proposes a decomposition heuristic for the joint caching and recommendations problem. The work in [69] formulates the joint problem in cache-aided device-to-device networks. With the objective of minimizing the offloading probability, it proposes a heuristic that decomposes the problem with respect to the caching and recommendation decisions. The authors in [70] formulate the joint problem in the somewhat different context of prefetching content over a time-varying channel, aiming at maximizing the throughput of the base station. They propose online and offline policies for prefetching and recommendations. The authors in [71, 72] adopt a different approach than the aforementioned works. Instead of assuming a known model for user behavior (and specifically how she accepts the recommendations) these works employ machine learning techniques to learn this behavior. Specifically, [71] formulates

the problem of deciding on caching and recommendations with the goal of maximizing the offloading probability at the base station. They employ a learning technique in order to learn the acceptance probability of the recommendations by the user. Similar in spirit, [72] presents the problem of jointly deciding on prefetching (in the user's device) and on the recommendations the user will receive with the objective of increasing the profit of the mobile network operator. They learn the users' behavior towards the recommendations through a reinforcement learning framework. From some of these works, it is already clear that it is more beneficial to jointly decide on caching and recommendations than just devising cache-friendly recommendations. However, since all of these proposed policies are based on heuristics, they do not provide performance guarantees.

Considering now different setups, [73] introduces the concept of "soft cache hits" that allows the user to choose an alternative cached content if the initially requested one is not locally cached. Through this framework, the authors devise approximate recommendation-aware caching policies for a network of caches that maximize the soft cache hit rate. Finally, in the different setting of broadcast coded transmissions, [74] studies the problem of making recommendations bandwidth-aware.

1.5 Thesis Contributions and Outline

We will now summarize the topics studied in each chapter. For every topic, we provide a brief description of the main motivations and a list of technical contributions.

1.5.1 The Joint Problem (Chapter 2)

In Chapter 2, we study the joint caching and recommendations problem. In the previous section, we discussed in detail the benefits of this approach when compared to cache-friendly recommendations or recommendation-aware caching policies. The approach we adopt is based on the following observations concerning the related work on this topic:

- The joint problem has been mainly addressed by heuristics in the literature. Even though the proposed algorithms are evaluated in numerical evaluations showcasing the benefits of the joint approach, no performance guarantees are provided.
- Most of the related work focuses on optimizing network-related metrics (such as cache hit rate) or on minimizing incurred delivery costs. Even though these models usually impose some kind of user-centric constraints on the cache-friendly recommendations, their focus is not the user's satisfaction.

Motivated by the above, we model the joint problem focusing on the user's overall experience. This is based on the fact that users are interested in both the content itself and the streaming quality in which this is delivered (as shown in real-world experiments [60]). For this reason, we define the Metric of Streaming Experience (MoSE) as a weighted sum of streaming quality and recommendations quality:

$$MoSE = Streaming\ Quality + \beta_u \cdot Recommendations\ Quality,$$

where β_u is the tuning parameter specific to each user. The value of β_u quantifies the importance of the recommendations quality compared to the streaming quality. The streaming quality component depends on the caching decisions and any metric of streaming quality the CP chooses to estimate, while the recommendations quality depends on the recommendation decision and the relevance of the recommended contents. This definition is generic enough to capture a variety of scenarios and measurements. We then formulate the problem of maximizing the aggregate MoSE over all users. The main contributions of this chapter are listed below.

Contributions of Chapter 2:

1. We introduce the metric of streaming experience (MoSE) for a recommendation-driven content application. We provide a variety of example functions/values for the components of our metric MoSE. We then formulate the problem of optimally deciding on caching and recommendations towards maximizing the users' MoSE.

2. Although we prove that the formulated problem is NP-hard, we provide a polynomial-time algorithm that approximates the optimal objective function value within a constant factor. To the best of our knowledge, this is the first algorithm with approximation guarantees proposed for the joint problem.

3. We show that the special case of the presented problem for a single cache can be transformed into an Integer Linear Program (ILP). This is particularly useful for scenarios of small size since exact algorithms could be applied.

4. Our numerical evaluations reveal a near-optimal performance and significant gains of our proposed policy over a variety of baseline schemes and existing algorithms for the joint problem. We also implemented distributed versions of our policy, and we show that important speedups can be achieved. Our evaluations were conducted on both real and synthetic datasets, and using realistic values for the problem parameters.

The work presented in Chapter 2 has been published in [75] and [76]. The details of these publications are provided below.

[75]: Tsigkari, Dimitra, and Thrasyvoulos Spyropoulos. "User-centric optimization of caching and recommendations in edge cache networks." In 2020 IEEE 21st International Symposium on" A World of Wireless, Mobile and Multimedia Networks" (WoWMoM), pp. 244-253. IEEE, 2020.

[76]: Tsigkari, Dimitra, and Thrasyvoulos Spyropoulos. "An approximation algorithm for joint caching and recommendations in cache networks." IEEE Transactions on Network and Service Management (2022), in press.

1.5.2 Recommendations as a Means of Cooperation (Chapter 3)

In Chapter 3, we study the cache-friendly recommendations problem from a network-economic point of view. We also extend our model to the joint caching and recommenda-

tions problem. Our approach is motivated by the following observations:

- Recommendations have become a powerful tool affecting all key stakeholders in the OTT and content distribution ecosystem. In other words, recommendations have a big impact in today's (multi-)billion-dollar industries that are involved in streaming services.

- The majority of the related work assumes that the same entity controls both caching and recommendation decisions. However, this is only the case for two CPs today: Netflix and YouTube. When these decisions are made independently by the CDN and CP respectively, a possible co-design of caching and recommendations could create tensions between the two entities. In particular, the cache-friendly recommendations of the CDN may deviate from the users' interests, and thus, negatively affect the CP's revenues, while the CP's recommendations might induce costly data transfers for the CDN.

Having this in mind, we propose a novel cooperation mechanism that allows the two entities to jointly design the recommendation policy. The core of our proposal is the following simple and practical idea: the CDN charges lower content delivery fees to the CP when the latter agrees to tune its recommendations towards cached contents. This discount will balance the CP's expected viewing gains with the CDN's induced savings on retrieval costs. Devising these cooperative recommendations is a new and highly non-trivial problem whose nature and complexity cannot be properly handled by existing approaches for cache-friendly recommendations. To the best of our knowledge, this is the first work proposing the cooperation of the CP and CDN on the grounds of recommendations.

From the observations listed above, it is clear that such a cooperation could take place only when incentives are provided to both entities, and when they can reach a fair agreement. For this reason, we employ tools from cooperative game theory, and we formulate the problem of cooperative recommendations as a Nash bargaining solution. This ensures a Pareto optimal solution and a fair split of the cooperation's gains. In summary, the contributions of this chapter are listed below.

Contributions of Chapter 3:

1. We provide a solid network-economic framework that takes into account the financial aspects of recommendations and the arising tradeoffs between the main actors. This framework is motivated by real-world business cases regarding the two entities' decision mechanisms and revenue models. We then identify and model the new problem of misaligned incentives among the CP and CDN regarding the recommendations offered to users.

2. We formulate a rigorous bargaining problem for addressing the trade-off between recommendation-induced revenues for the CP and retrieval costs for the CDN in streaming services. The problem's solution will allow them to devise the cooperative recommendations while fairly splitting the incurred gains.

3. We propose the CCR algorithm for the scenario where the two entities share the necessary information regarding their cost/revenue functions with a third party that solves the bargaining problem in a centralized fashion. We also propose the DCR algorithm for the scenario where the CP and CDN have undisclosed private information. This leads to a distributed bargaining solution where the CP and CDN solve their own problem instances while being oblivious to each other's private information.

4. We discuss how the presented framework can be extended to cooperative caching policies and analyze its difficulty. This problem of cooperative recommendations and caching turns out to be hard to solve, but has the potential to further increase the cooperation's gains.

5. Through a number of numerical evaluations using a real dataset and realistic system parameters, we verify the efficiency and operation of the bargaining framework, and explore the impact of key system parameters on the equilibrium properties. This provides rich insights on the potential economic benefits of our proposal and market design guidelines.

The work presented in Chapter 3 has been published/submitted in [77] and [78]. The details of these publications/submissions are provided below.

[77]: Tsigkari, Dimitra, George Iosifidis, and Thrasyvoulos Spyropoulos. "Split the cash from cache-friendly recommendations." In 2021 IEEE Global Communications Conference (GLOBECOM), pp. 1-6. IEEE, 2021.

[78]: Tsigkari, Dimitra, George Iosifidis, and Thrasyvoulos Spyropoulos. "Quid Pro Quo in Streaming Services: Algorithms for Cooperative Recommendations." *Submitted and under review* (2022).

Finally, in Chapter 4, we conclude by giving some directions for future work. These directions not only address the potential limitations of our contributions, but also answer to some questions that naturally arise (and have not been studied yet in related work). We briefly discuss the challenges and possible solution methods for these proposals.

Chapter 2

An Approximation Algorithm for the Joint Caching and Recommendations Problem

2.1 Introduction

On platforms of streaming services, such as YouTube, Netflix, and Disney+, the streaming quality (SQ) of the delivered content plays a significant role in the overall user's experience on the service. This quality can be characterized, for example, by metrics of QoS or QoE. Furthermore, the SQ is closely related to the user engagement to the service. It has been shown that, on platforms of video streaming services, low bitrate can lead to an increase in the abandonment rate [5]. At the same time, the recommendations that appear on the user's interface have a strong impact on content requests. The goal of these recommendations is to suggest contents based on the user's interests and, therefore, the user's engagement with the service is closely related to the recommendations quality (RQ) [51].

Caching mechanisms within the Content Delivery Networks (CDNs) that store contents in caches close to the user can ensure a better delivery in terms of SQ [11] while alleviating the backhaul link traffic. Making the right caching decisions is crucial in today's and, more importantly, in future architectures, as was discussed in Sec. 1.2. This is exactly where the influence of the recommendations on users' requests could come into the picture. At first glance, content caching and recommendation systems seem to be independent, since they are usually handled by two different entities: the CP and a 3rd party CDN (like Akamai). However, caching and recommendation decisions can already be handled jointly in today's architectures for CPs who own an in-house CDN solution (see Sec. 1.2.1). The trend towards end-to-end network slicing in future wireless networks further supports such an approach. In this setting, CPs will own their own virtual network (slice) including communication, storage, and CPU resources at the base stations of the Radio Access Network (RAN).

In this chapter, we formulate and analytically study the problem of jointly optimizing

both caching and recommendation decisions in a generic network of caches. In particular, we aim to decide on : (i) what content to store at each cache, and (ii) what content to recommend to each user, based on their location in the caching network and their predicted preferences. Based on the impact of the RQ and SQ on the user's engagement and satisfaction, we adopt a user-centric approach. We define the metric of streaming experience (MoSE) that captures the fundamental tradeoff between the SQ and RQ. We formulate this joint optimization problem with the objective of maximizing this metric. We prove that it can be approximated up to a constant factor. To the best of our knowledge, this is the first polynomial algorithm to achieve a constant approximation ratio for the joint problem.

2.2 Problem Setup

2.2.1 Caching Network

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

Definition 2.1 (Caching variables)**.** We let x_{ij} be the binary variable, where $x_{ij} = 1$ when the content i is cached in cache j, and $x_{ij} = 0$ otherwise. We denote the corresponding matrix by $X = \{x_{ij}\}_{i,j}$.

We consider a set $\mathcal{U}$ of users, each of which has access to a subset of caches. We denote this set by $\mathcal{C}(u)$ for user $u \in \mathcal{U}$. A request for content i by user u is served by one of the caches belonging to $\mathcal{C}(u)$ where the requested content is stored, *i.e.*, by one of the caches of the set $\{j : j \in \mathcal{C}(u) \text{ and } x_{ij} = 1\}$. The access to a cache could be over multiple links (as in hierarchical caching or in Information Centric Networking (ICN)) or direct (*e.g.*, wireless connectivity to a nearby small cell [28]). For the purposes of our analysis, such networks can be represented as a generic bipartite graph between users and (associated) caches, as shown in Fig. 2.1. Specifically, every edge of this graph has a weight s_{uj}, which denotes the cache-specific streaming quality[2] that can be supported between user u and cache j. This quality can be related to estimations on the QoS or QoE, *i.e.,* delays, rebufferings, rate switching, etc. Moreover, it may differ from cache to cache, or may depend on channel quality, number of hops, scheduling policy, congestion level, etc. Finally, there is a large cache C_0 that fits all the contents, *i.e.*, $x_{i0} = 1$ for all $i \in \mathcal{K}$, and is accessible by all users, *i.e.*, $C_0 \in \mathcal{C}(u)$, for all $u \in \mathcal{U}$. This could be a large cache deep(er) in the network. For this reason and w.l.o.g., we let $s_{u0} < s_{uj}$, for all j and u, as is commonly assumed (*e.g.*, in [28, 41]).

[1] In fact, according to estimations of the size of Netflix catalog [79] and the size of Open Connect appliances [80], the cache capacity of the appliances varies from 0.1% to 2.3% of the entire catalog.

[2] This assumption is realistic. At Netflix, for example, information on routes, network proximity to the users, etc. are gathered by the Open Connect caches and are sent regularly to the Netflix cloud [20].

In our model, the caches are filled or updated during off-peak hours and, therefore, the cache allocation is static for the time period between two cache updates (*e.g.,* a day or a time window of a few hours). In this context, if a requested content is not cached in any of the caches, the content is served by the large cache C_0. Although CDN caches traditionally have been operated with the use of dynamic caching policies (where the cache is typically updated upon a cache miss), it has been argued that such policies perform poorly in the scenario of small caches (under a non-stationary request model) [35]. Hence, in such setups, similar models like ours are much more common in related works, *e.g.*, [28, 41]. More importantly, there is a trend towards static caching model even in today's architectures: Netflix, for example, is updating the Open Connect caches every night during off-peak hours [24]. This further supports our model. Therefore, in what follows, all the problem parameters are considered to be known for the time period between two cache updates. The presented caching setup is generic and could capture a variety of caching networks, such as femto-caching setups [28], hierarchical CDN networks [81], etc.

2.2.2 Recommendations and User Model

A list of N_u recommended contents appears to the user $u \in \mathcal{U}$. This number may vary from user to user depending on the device used, as is the case in Netflix [51], for example. As discussed in Sec. 1.3.1, the recommendations are personalized and might depend on various factors such as user ratings (*e.g.*, via collaborative filtering), past user behavior, viewing times, etc. [55]. State-of-the-art RSs usually assign a relevance or utility (or "ranking") to each content for each user u [55, 82]. We denote by $r_{ui} \in [0, 1]$ these relevances. Typically, the CP would select the N_u items with the highest r_{ui} to feature the recommendations list of user u. In our work, the recommendation decisions (*i.e.,* deciding which contents will appear to the user's recommendations list) are made not only based on the relevances r_{ui} but also on the caching decisions. Therefore, our model uses the relevances r_{ui} (that are derived from a typical RS) as input for our problem.

Definition 2.2 (Recommendation variables)**.** We let $y_{ui} \in \{0, 1\}$ denote the binary variable for content i being recommended to user u ($y_{ui} = 1$) or not ($y_{ui} = 0$). We denote by Y the matrix of y_{ui}. Then, the equations $\sum_{i \in \mathcal{K}} y_{ui} = N_u$, for all $u \in \mathcal{U}$, capture the fact that N_u contents are recommended.

Motivated by the discussion in Sections 1.4 and 2.1, we assume that both caching and recommendation decisions are made by the same entity (e.g., Netflix).

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

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

Essentially, α_u captures the percentage of time a user u tends to follow the recommendations. For example, it is estimated, on average, that $\alpha_u = 0.8$ on Netflix [51], but it can of course differ among users. Assuming prior knowledge of the user's disposition to follow the recommendations is common in related works (*e.g.*, [67]) and also in other works on recommendation systems (*e.g.*, [83]). In practice, α_u might change over longer time intervals both because of intrinsic changes to user behavior or due to decreasing/increasing trust in the recommendations. Such changes could be addressed by dynamic or stochastic models. In this work, we assume that our optimization happens at a smaller time scale, for which the parameter α_u is roughly constant (but it can be recalibrated at longer intervals).

Furthermore, the assumption that each recommended content will be clicked with equal probability $1/N_u$ is also common in related works, and might hold in scenarios where the recommended items are "unknown" to the user, and hence she cannot evaluate their utility, before requesting them.

As for the p_{ui}, they capture the probability of user u requesting the content i outside of recommendations (*e.g.*, through the search bar). This could be an arbitrary distribution over the catalog (*e.g.*, with probability mass only on content the user already "knows"). Alternatively, given the relevances r_{ui}, a reasonable choice could also be the normalized values:

$$p_{ui} = r_{ui} / \sum_{k \in \mathcal{K}} r_{uk}. \tag{2.1}$$

2.2.3 Example

To better elucidate our model thus far, we present a small-scale example and Fig. 2.1 that illustrates the variables and the parameters defined above. We consider a network of $C = 2$ caches of capacity 2, and a large cache C_0 containing the entire catalog that consists of $|\mathcal{K}| = 9$ equal-sized contents. As shown, cache 1 contains contents 1 and 4 (*i.e.*, $x_{11}, x_{14} = 1$), while $x_{1j} = 0$ for any other j. There are 3 users present in the network. An edge between a user u and a cache j means that user u can fetch a content from cache j. For example, for user 1, we have that $\mathcal{C}(1) = \{0, 1\}$. Note that such an edge might actually correspond to a path of multiple physical links. The corresponding cache-specific stream. quality is indicated as an edge weight. In this example, a single recommendation ($N_u = 1$) appears to every user (illustrated by a dashed-line arrow). For example, the content 4 is recommended to user 1 (*i.e.*, $y_{14} = 1$). If user 1 requests it, then it can be streamed from cache 1 at streaming quality s_{11}. However, if user 1 requests, say, the content 2, this will be fetched from cache C_0 at a (lower) quality s_{10}. Lastly, arrows from users to recommendations display the probabilities α_u. We note that, for the convenience of the reader, we provide a summary of the notation used in this chapter in Table 2.1.

2.2.4 Metric of Streaming Experience (MoSE)

In the context of media streaming platforms, the user's entertainment and contentment with the provided services are affected by the quality of the recommendations she receives,

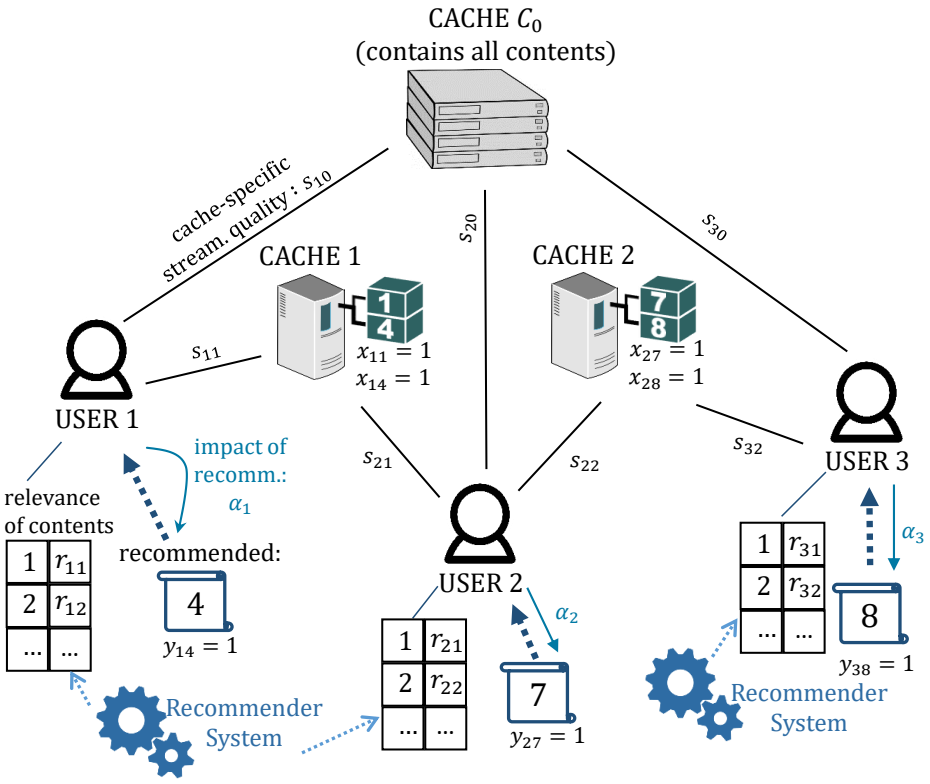


Figure 2.1: Illustration of the variables and parameters considered for the joint caching and recommendations problem in a network of caches. Detailed description of this example can be found in Section 2.2.3.

i.e., if they are tailored to her tastes or not. On the other hand, it has been observed that low SQ (*e.g.*, low bitrates, rebufferings, etc.) greatly affects user experience and, most importantly (for CPs), retention/abandonement rates [5]. In fact, experiments on video streaming services have shown that the user's overall experience depends on both the streaming quality and the user's interest in a content [60]. Moreover, some recent experimental evidence suggests that users might be willing to tradeoff (some) content relevance for (better) QoS [64]. In this direction, *we define the metric of streaming experience as a twofold quantity*: one part relates to the recommendation quality; the second part relates to the streaming quality.

Definition 2.3 (Recommendations Quality - RQ)**.** The recommendations quality, as perceived by user u, is equal to $\sum_{i \in \mathcal{K}} y_{ui} \varphi(r_{ui})$, where φ is any non-decreasing function.

The function φ represents the impact of a recommended content's relevance r_{ui} in the user's perceived RQ. It could be a linear function, or, more commonly, a concave function (*e.g.*, $\log(r_{ui})$) to capture diminishing returns beyond a minimum content relevance. Moreover, we can demand a minimum quality $r_{\min}$ for any recommendation[3]

[3]This quantity can serve as an additional safeguard to support the earlier assumptions: *e.g.*, if any

Table 2.1: Notation Summary for Chapter 2

Notation	Description
$\mathcal{K}$	catalog of contents
$\mathcal{U}$	set of users in the network
C_0	large cache containing the entire catalog
C	number of caches in the network (C_0 is excluded)
$\mathcal{C}_j$	capacity of cache j, $j = 0, \cdots, C$
$\mathcal{C}(u)$	set of caches that user u communicates with
r_{ui}	relevance (or utility) of content i for user u
s_{uj} ($s_{u(j)}$)	cache-specific streaming quality between user u and cache j, $s_{u(j)}$ for ordered qualities (see Def. 2.4)
σ_i	size of content i
N_u	number of recommended contents for user u
α_u	probability that user u follows the recommendations
p_{ui}	probability that user u requests content i while not following the recommendations
φ	function of r_{ui} that captures the impact of r_{ui} in the perceived recommendations quality
x_{ij}	caching variable, $x_{ij} = 1$ when content i is cached in cache j, and $x_{ij} = 0$ otherwise
y_{ui}	recommendation variable, $y_{ui} = 1$ when content i is recommended to user u, and $y_{ui} = 0$ otherwise

if we define φ as follows:

$$\varphi(r_{ui}) = \begin{cases} \log(r_{ui}) & \text{if } r_{ui} \geq r_{\min}, \\ -\infty & \text{otherwise.} \end{cases} \tag{2.2}$$

Regarding the SQ, this depends on which cache the requested content is streamed from. A content i requested by user u will be fetched by the "best" *connected* cache that stores it, as in [28].

Definition 2.4 (Ordered cache-specific qualities)**.** If $\mathcal{C}(u)$ is the set of caches that user u has access to, we let $s_{u(1)} = \max\{s_{uj}, j \in \mathcal{C}(u)\}$ denote the maximum (cache-specific) quality for user u. Similarly, $s_{u(2)}$ denotes the second highest quality for u, and so forth[4].

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

recommended content's relevance is above r_{min}, then, α_u, the user's trust in the recommendations, will not be compromised.

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

Lemma 2.1 ((Expected) Streaming Quality- SQ)**.** *For a given cache allocation X, a content $i \in \mathcal{K}$ will be streamed to user u (upon request) in the quality:*

$$s_u(X,i) := \sum_{j=1}^{|\mathcal{C}(u)|} \left[s_{u(j)} x_{i(j)} \prod_{l=1}^{j-1} (1 - x_{i(l)}) \right], \tag{2.3}$$

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

$$\overline{s}_u = \alpha_u \sum_{i \in \mathcal{K}} \frac{y_{ui}}{N_u} s_u(X,i) + (1 - \alpha_u) \sum_{i \in \mathcal{K}} p_{ui} s_u(X,i). \tag{2.4}$$

Proof. For a requested content $i \in \mathcal{K}$, $\prod_{l=1}^{j-1}(1 - x_{i(l)}) x_{i(j)}$ captures the fact that i will be retrieved by the cache (j) (*i.e.*, the cache with the j-th highest quality) for lack of any other cache with higher quality in $\mathcal{C}(u)$ where the content is cached (*i.e.*, $x_{i(l)} = 0, l < j$). Then, this request will be served in the cache-specific quality $s_{u(j)}$. Of course, if i is not cached in any cache, it will be retrieved from C_0 which is ranked last, resulting in low streaming quality. Essentially, $s_u(X,i)$ is the highest cache-specific quality associated to content i for user u among all the locations where i is cached. Finally, given that, upon request, the content i will be streamed in the quality $s_u(X,i)$, and given the probabilities of such a request to happen (through recommendations or not), the formula of the expected streaming quality, $\overline{s}_u$, easily follows. □

Remark 2.1. When estimating[6] the SQ, s_{uj}, the cache-specific streaming quality, can be chosen to be a function of QoS-related or QoE-related estimations. For example, s_{uj} could be the estimated bitrate between user u and cache j, or it could also include factors related to rebuffering probabilities, jitter, delays, etc., as is commonly considered in works related to streaming experience [84]. Our framework is defined in such a way that is flexible enough to optimize whatever such values(s) the CP deems appropriate or is able to estimate. Alternatively, s_{uj} could be equal to:

$$s_{uj} = \begin{cases} 1 & \text{if } j \in \mathcal{C}(u) \setminus C_0, \\ 0 & \text{otherwise.} \end{cases} \tag{2.5}$$

In that case, $\sum_u \overline{s}_u$ estimates the expected cache hits for the (small) caches: upon a request, it counts 1 if the content is cached. In Table 2.2, we provide a variety of examples functions/values for s_{uj} that exist already in the literature on multimedia streaming services.

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

[6] Since our focus is proactive caching, we consider the estimated or average SQ. The estimation could, for example, take into account the DASH algorithm that is in place. An interesting direction for future work would be to consider dynamic policies where the SQ is time-varying or even unknown.

Definition 2.5 (MoSE function)**.** The metric of streaming experience for user $u \in \mathcal{U}$ as a function of the caching and recommendation variables is defined as $\overline{s}_u + \beta_u \sum_{i \in \mathcal{K}} y_{ui} \varphi(r_{ui})$, where $\overline{s}_u$ is given by (2.4) and $\beta_u > 0$ is a tuning parameter. Then the aggregate MoSE over all users is equal to:

$$f(X, Y) := \sum_{u \in \mathcal{U}} \left[\overline{s}_u + \beta_u \sum_{i \in \mathcal{K}} y_{ui} \varphi(r_{ui}) \right]. \tag{2.6}$$

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

We stress here that the MoSE was defined in such a generic way that can be adjusted according to the needs of the CP. Particularly, the CP can choose the quantities s_{uj} and the function φ based on available measurements in network, user behavior, etc. We provide a detailed example set of choices (derived mostly from related work) for the different MoSE components, as shown in Table 2.2.

Finally, metrics similar to MoSE are also employed in other recommendation-driven applications. This is, for example, the case of Google Search. The algorithm employed by Google Search, that provides links after a search, can be seen as an RS: the results are personalized (based on previous search history, browser cookies, etc. [85]) and they are ranked according to their relevance or other criteria (*e.g.,* ads, sponsored links). In order to improve user experience, Google has incorporated the loading speed of each webpage as a factor in this ranking [86]. This implies that webpages that are relevant to the search terms, but are observed to be slow to load, will be ranked further down in the list of search results.

2.2.5 Joint Caching and Recommendations (Toy Example)

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

To better understand the tradeoffs involved, we present a toy example depicted in Fig. 2.2, and two "naive" policies:

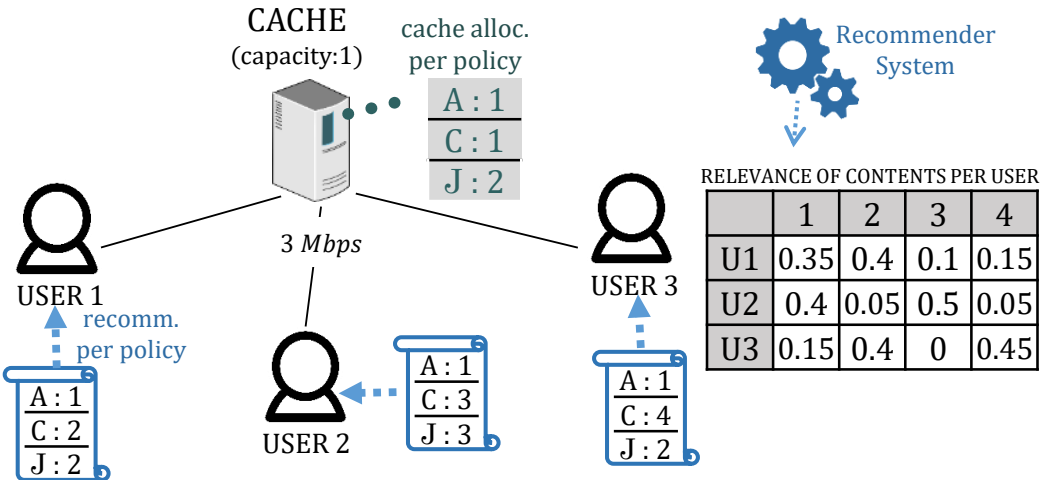
Policy C, for "Conservative". This policy caches the $\mathcal{C}_j$ most popular contents (for the users connected to the cache j); it then recommends to each user u the N_u contents that are the most relevant (*i.e.,* highest r_{ui}), regardless of whether they are cached or not. This policy captures today's status quo.

Policy A, for "Aggressive". This policy has the same caching policy as policy C, but

Table 2.2: Example functions/values for the MoSE components

Component	Example functions/values	Comments
s_{uj} in SQ (as in Lemma 2.1)	• QoS metrics	*e.g.*, related to bitrate, start-up delay, packet loss, etc.
	• estimated QoE as a function ψ of QoS (ψ can be linear, logarithmic, exponential etc.)	as in [87] (exponential QoS-QoE relation, IQX hypoth.) or as in [88] (logarithmic QoS-QoE relation)
	• estimated multidimensional QoE	as in [9] or as in [84]
	• cache hits (as defined in (2.5))	requires no network-related measurements/estimations
$\beta_u > 0$ (weight factor)	• small β_u	β_u represents the sensitivity of user u to the SQ and RQ, small/large $\beta_u \rightarrow$ priority to the SQ/RQ
	• large β_u	
function φ in RQ (as in $\sum_{i \in \mathcal{K}} y_{ui} \varphi(r_{ui})$)	• logarithmic function	it captures the diminishing returns property (as often observed in human perception [89])
	• as defined in (2.2)	it captures minimum QR thresholds per user

it recommends only cached contents (the most relevant to the user among them). It is closer to cache-friendly recommendation policies like the one in [73].



	1	2	3	4
U1	0.35	0.4	0.1	0.15
U2	0.4	0.05	0.5	0.05
U3	0.15	0.4	0	0.45

Figure 2.2: Toy example of Sec. 2.2.5. On the left: illustration of the network together with the caching and recommendation decisions made by policies A, C, and J. On the right: the matrix of content relevances/utilities per user.

Note that both policies make the caching and recommendation decisions *separately*. In this example, we will attempt to show the benefits of a policy that makes jointly these decisions. Referring to Fig. 2.2, suppose we have a catalog of 4 equal-sized contents and 3 users, all connected to the large cache C_0 (not shown in the figure, for simplicity) that contains all files and a smaller cache C_1 of capacity 1. We measure the SQ through the estimated bitrate. All users can download a content from C_1 or C_0 with bitrate 3 Mbps or 2 Mbps respectively. We further assume that $N_u = 1$ and $\alpha_u = 1$ for all users. We depict the relevances r_{ui} on the right side.

Both policies would cache the item with the highest aggregate relevance/utility, *i.e.*, content 1. Policy A would then recommend this item to *all* users. Policy C would instead recommend the most related item per user, namely contents 2, 3 and 4 respectively. It is easy to see that policy C would lead to better RQ, while policy A would lead to better SQ. However, we would like to know which policy is optimal with respect to maximizing the aggregate MoSE (as expressed in (2.6)).

A better option would be to cache content 2, observing that this would then facilitate the recommendation decisions. More precisely, it allows one to recommend content 2 to both users 1 and 3, achieving cache hits for them with maximum or close to maximum RQ. Instead, for user 2, the content 3 is recommended (with relevance $r_{23} = 0.5$), since content 2 would seriously degrade the user's RQ ($r_{22} = 0.05$ only). This policy which we refer to as "J" for Joint in Fig. 2.2, outperforms both A and C in this example in terms of MoSE (for φ being the logarithmic function and for most values of $\beta > 0$).

In this example, it is easy to guess how to outperform the policies A and C (or even find the optimal one). However, this task becomes significantly harder for bigger

scenarios (when considering overlapping cache topologies, large content catalogs, multiple recommendations per user, etc.). To this end, in the next section, we formulate and analyse this problem, and propose an algorithm with approximation guarantees.

2.3 Problem Formulation and Analysis

The optimization problem we are targeting is the following:

MoSE problem.

$$\begin{aligned} \underset{X,Y}{\text{maximize}} \quad & f(X,Y) \\ \text{subject to} \quad & \sum_{i\in\mathcal{K}} \sigma_i x_{ij} \leq \mathcal{C}_j \text{ for every } j = 1,\dots,C; && (2.7)\\ & \sum_{i\in\mathcal{K}} y_{ui} = N_u \text{ for every } u \in \mathcal{U}; && (2.8)\\ & x_{ij}, y_{ui} \in \{0,1\}, && (2.9) \end{aligned}$$

where, according to (2.3), (2.4) and (2.6), $f(X,Y)$ is equal to

$$\sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{K}} \left[\alpha_u \frac{y_{ui}}{N_u} s_u(X,i) + (1-\alpha_u)\, p_{ui}\, s_u(X,i) + \beta_u y_{ui}\varphi(r_{ui})\right]$$

and $s_u(X,i) := \sum_{j=1}^{|\mathcal{C}(u)|} \left[s_{u(j)}\, x_{i(j)} \prod_{l=1}^{j-1}(1-x_{i(l)})\right]$. The constraints in (2.7) are the capacity constraints for every cache. In the case of equal-sized contents, (2.7) suggests that no more than $\mathcal{C}_j$ items can fit in cache j, and the constraints in (2.8) suggest that each user receives N_u recommendations. Finally, as expressed in (2.9), the problem's variables, x_{ij} and y_{ui}, are binary.

Lemma 2.2. *The MoSE problem is NP-hard.*

Proof. An instance of the MoSE problem (when the recommendation variables are fixed) is the well-known femto-caching problem [28]. The authors in [28] proved that this problem is NP-hard through a reduction to the set cover problem. □

2.3.1 Intuition on Joint Optimization

As we saw in Lemma 2.2, even just the caching part (*i.e.*, maximizing in variable X) of the MoSE problem is hard to solve. For this simpler problem, the authors in [28] propose algorithms with approximation guarantees by exploiting submodularity properties of the objective. However, these algorithms do not account for the recommendation part of the MoSE problem (variable Y) and, therefore, the approximation guarantees do not extend to the joint problem.

One could be tempted to extend the methodology in [28] by using both sets of variables X and Y as the ground set. However, the authors of [67] prove that a subcase of the MoSE problem (when $\beta_u = 0$) is not submodular in X and Y.

Furthermore, the authors of [73] consider problem variants where the caching decision is "recommendation-aware". They show that this problem is hard even for one cache, but manage to retrieve submodularity properties and use the methodology of [28] to derive algorithms with approximation guaranties. However, their objective and problem setup do not contain recommendation variables (among other things, the recommender's actions are fixed, and the caching policy simply knows what the recommender would do). It is thus significantly different than the MoSE problem. Finally, a brief qualitative comparison of these works is shown in Table 2.3.

Table 2.3: State-of-the-art works on caching and/or recommendations

Related Works	Variables		How many caches?	Approx. guarantees
	Caching	Recomm.		
[28]	✓	✗	Network	✓
[73]	✓	✗*	Network	✓
[67]	✓	✓	Single cache	✗
This work	✓	✓	Network	✓

*In [73], although the problem formulation does not contain any recommendation variable, the caching variable is "recommendation-aware".

This discussion raises the question of whether the MoSE problem can be efficiently approximated and how. In the next section, we prove that this is indeed the case. By first considering something akin to a primal decomposition [90] of the original problem (rather than handling variables X and Y at the same time as the ground set), we show that:

(i) for the problem on variables Y, *i.e.*, fixing X ("inner" problem), the global maximizer can be found efficiently;

(ii) the problem on variables X ("outer" problem), given the global maximizer of Y (for any X), is in fact submodular.

This property will allow us to devise an algorithm for the joint problem that is polynomial in the problem size and, somewhat surprisingly, retains the approximation guarantees of the much simpler "caching-only" problems considered in [28] and [73].

2.3.2 Towards Efficient Algorithms

The key to our methodology is the following lemma.

Lemma 2.3. *The MoSE problem is equivalent to the problem:*

Outer problem.

$$\begin{aligned} \underset{X}{\text{maximize}} \quad & f^*(X) := f(X, \underset{Y}{\operatorname{argmax}} f(X, Y)) \\ \text{subject to} \quad & (2.7), (2.8), \text{ and } (2.9). \end{aligned} \tag{2.10}$$

The equivalence of the two problems follows straightforwardly from the well known identity [91]:

$$\max_{X,Y} f(X,Y) = \max_{X}(\max_{Y} f(X,Y)). \tag{2.11}$$

Inner problem and algorithm

The first step would be to find a closed-form expression for f^* for any cache allocation, *i.e.*, matrix X. Hence, given X, the problem of choosing the recommendation policy, *i.e.*, matrix Y, is the problem of finding $f^*(X)$, as defined in (2.10). We formulate this problem:

Inner problem.

$$\begin{aligned} \underset{Y}{\text{maximize}} \quad & f(X,Y) \\ \text{subject to} \quad & (2.8) \text{ and } y_{ui} \in \{0,1\}. \end{aligned}$$

The following lemma states that the inner problem can be decoupled into $|\mathcal{U}|$ problems.

Lemma 2.4. *If* $F_u^*(X) := \max_{Y} \left(\bar{s}_u + \beta_u \sum_{i\in\mathcal{K}} y_{ui}\varphi(r_{ui})\right)$*, for any* u *and any placement* X*, then* $f^*(X) = \sum_{u\in\mathcal{U}} F_u^*(X)$*.*

Proof. Given a cache placement X, it is easy to see that the recommendation decisions (variable Y) for a user do not interfere with the decisions for the other users. Moreover, the constraints in (2.8) are decoupled for every user. □

By (2.3) in Lemma 2.1, we can write $F_u^*(X)$ as follows.

$$\begin{aligned} F_u^*(X) &= \max_{Y}\left(\sum_{i\in\mathcal{K}} y_{ui}\left(\frac{\alpha_u}{N_u}s_u(X,i) + \beta_u\varphi(r_{ui})\right)\right) \\ &+ (1-\alpha_u)\sum_{i\in\mathcal{K}} s_u(X,i)p_{ui}. \end{aligned} \tag{2.12}$$

Next, we introduce the notion of V-value, which is the coefficient of y_{ui} in (2.12).

Definition 2.6 (V-value and ordered V-values)**.** We define, as V-value of a content $i \in \mathcal{K}$ for user $u \in \mathcal{U}$ and for a given cache allocation X, the quantity

$$V_{ui}(X) := \frac{\alpha_u}{N_u}s_u(X,i) + \beta_u\varphi(r_{ui}), \tag{2.13}$$

where $s_u(X,i)$ is defined in (2.3). Similar to Def. 2.4, we define the ordered V_{ui} (sorted in decreasing order) as the ordered sequence $\{V_{u[k]}\}_{k\in\mathcal{K}}$[7].

[7]We do not use the same notation as in Def. 2.4 because the ordering here is done with respect to the V-value and not the quality s_{uj}. In general, $V_{u(k)}(X) \neq V_{u[k]}(X)$, for all $u \in \mathcal{U}$ and $k = 1, \ldots |K|$.

Lemma 2.5. *For a given cache allocation X, we consider the matrix Y' such that $y'_{u[k]} = 1$ for $k = 1, \ldots, N_u$, and $y'_{u[k]} = 0$ otherwise, where $[k]$ is the content index associated to the k-th highest V-value for the user $u \in \mathcal{U}$. Then*

$$F_u^*(X) = \sum_{k=1}^{N_u} V_{u[k]}(X) + (1-\alpha_u) \sum_{i \in \mathcal{K}} \big(s_u(X,i) p_{ui}\big), \tag{2.14}$$

$$\text{and} \qquad f^*(X) = f(X, Y') = \sum_{u \in \mathcal{U}} F_u^*(X). \tag{2.15}$$

In words, the optimal solution for the inner problem is to recommend to every user u the N_u contents with the highest V-value associated to the cache placement X. Note that this solution depends on the solution of the outer (caching) problem and, hence, the inner problem needs to be solved as part of solving the outer problem, as shown in (2.11).

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

Based on Lemma 2.5, the algorithm that finds the solution for the Inner Problem is summarized in Algorithm 1.

Algorithm 1: **Inner algorithm (subroutine)**

Input : $\mathcal{U}$, $\mathcal{K}$, N_u, X, $\{\beta_u\}$, φ, $\{\alpha_u\}$, $\{r_{ui}\}$, $\{s_{uj}\}$

1 Start with empty matrix Y
2 **for** *every user* $u \in \mathcal{U}$ **do**
3 **for** *every content* $i \in \mathcal{K}$ **do**
4 Calculate V_{ui};
5 Store $\{V_{u[k]}\}_{k=1}^{N_u}$ (in decreasing order). ;
6 **end**
7 Set $y_{u[k]} = 1$ for $k = 1, \cdots, N_u$;
8 **end**
9 **Return** Y

Complexity of the inner algorithm

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

Outer problem and submodularity

We proved that the optimal Y can be found efficiently for the inner problem, given any cache allocation X. We want now to solve the outer problem (defined in Lemma

2.3, (2.14), (2.15)), *i.e.,* with respect to the caching variables X. While often caching problems can fit into the category of knapsack/general assignment problems, this is not the case for the outer problem. We note that the "profit" or gain of storing a content into a cache is not a constant and depends on the solution of the inner problem. Nevertheless, we will now prove some interesting properties of the outer problem that will lead us to an algorithm for the MoSE problem.

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

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

Essentially, P_X consists of the pairs (content, cache) of all the cached contents. Since, by definition, the large cache C_0 contains the entire catalog (*i.e.*, $x_{i0} = 1$, for all $i \in \mathcal{K}$), X is a $|\mathcal{K}| \times (C+1)$ matrix. In other words, P_X belongs to the set $\mathcal{P} := P(\mathcal{K} \times \{1, \ldots, C+1\})$, where $P(\mathcal{K} \times \{1, \ldots, C+1\})$ is the powerset of $\mathcal{K} \times \{1, \ldots, C+1\}$. Inversely, given a placement P, we can define the corresponding matrix X_P such that x_{ij} is equal to 1, for every pair (i,j) in P, and 0 otherwise. Hence, from now on, X and P will be used interchangeably to denote the content allocation across the network of caches. We also define the subsets of a placement P representing the storage of the cache m: $P^{(m)} := \{(i,m) \in P\}$. We can thus extend F_u^*, f^*, s_u and V_{ui} to the ground set $\mathcal{P}$.

Lemma 2.6. *The set function F_u^* is monotone increasing for all $u \in \mathcal{U}$.*

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

$$s_u(P,i) \leq s_u(Q,i), \text{ for all } i \in \mathcal{K}. \tag{2.16}$$

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

Next, by Definition 2.6, the following inequalities are true

$$V_{ui}(P) \leq V_{ui}(Q), \tag{2.17}$$
$$V_{u[k]}(P) \leq V_{u[k]}(Q), \text{ for all } i,k \in \mathcal{K}. \tag{2.18}$$

Finally, it follows by (2.12) that $F_u^*(P) \leq F_u^*(Q)$. □

Next, we define the marginal gain of F_u^* and we state an immediate consequence of Lemma 2.6.

Corollary 2.1 (Marginal gain)**.** *For a cache placement P, and a pair (i,j) such that $(i,j) \notin P$, we denote by*

$$\Delta F_u^*(P,(i,j)) := F_u^*(P') - F_u^*(P),$$

where $P' := P \cup \{(i,j)\}$, the marginal gain of F_u^* at P with respect to (i,j). Then, $\Delta F_u^*(P,(i,j)) \geq 0$.

Lemma 2.7. *The set function F_u^* is submodular*[8] *for all $u \in \mathcal{U}$.*

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

$$\Delta F_u^*(A,(i,j)) \geq \Delta F_u^*(B,(i,j)). \tag{2.19}$$

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

In order to prove Lemma 2.7, we need a few intermediate results. *All the results and proofs given here are true for any $u \in \mathcal{U}$, so, for simplicity, the index u will be omitted throughout the two following lemmas and the proof of Lemma 7.*

Lemma 2.8. *Given P, a placement of cached items, and a pair $(i,j) \in (\mathcal{K} \times \{1,\dots,C\})$ such that $(i,j) \notin P$, we write $P' := P \cup (i,j)$. Then, $s(P',i) = max\{s_j, s(P,i)\}$.*

Proof. By (2.3), after adding (i,j) to the placement, if s_j is greater than $s(P,i)$, then i will be retrieved from cache j, *i.e.*, $s(P',i) = s_j$. On the other hand, if cache j does not offer a better quality for the user than before, the quality associated to i will stay the same, *i.e.*, $s(P',i) = s(P,i)$. □

Lemma 2.9. *Given P, a placement of cached items, and a pair $(i,j) \in (\mathcal{K} \times \{1,\dots,C\})$ such that $(i,j) \notin P$, the following statements are true:*

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

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

In the second case, the marginal gain is equal to

$$\begin{aligned} \Delta F^*(P,(i,j)) &= (1-\alpha)\, p_i\, (s_j - s(P,i)) \\ &+ \begin{cases} 0, & \text{if } V_i(P') \leq V_{[N]}(P), \\ V_i(P') - M, & \text{if } V_i(P') > V_{[N]}(P), \end{cases} \end{aligned}$$

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

Essentially, Lemma 2.9 states that adding (i,j) to P will lead to a positive marginal gain of F^* if and only if the cache j can provide a higher (cache-spec.) quality for the user than any other cache where i was already cached (in P).

Proof. By Lemma 2.5, $\Delta F^*(P,(i,j)) > 0$ if and only if

$$\sum_{k=1}^{N} \big(V_{[k]}(P') - V_{[k]}(P)\big) > 0 \tag{2.20}$$

$$\text{or} \quad \sum_{k \in \mathcal{K}} (s(P',k) - s(P,k)) > 0. \tag{2.21}$$

[8] For definition, see [92].

Given that the only difference between P and P' is the content i in cache j, the qualities of the contents other than i remain the same as the addition of (i, j) does not affect them. As a result, the inequality in (2.21) is true if and only if $s(P', i) > s(P, i)$. By Lemma 2.8, $s(P', i) = s_j$ and, therefore, (2.21) is equivalent to $s_j > s(P, i)$. Next, when the inequality (2.20) holds, $V_i(P') > V_i(P)$ because, otherwise, $V_{[k]}(P') = V_{[k]}(P)$, for every $k \in \mathcal{K}$. By (2.13), the inequality $V_i(P') > V_i(P)$ is equivalent to $s(P', i) = s_j > s(P, i)$. Hence, we proved that the inequality in (2.20) implies $s_j > s(P, i)$, which is equivalent to (2.21). Therefore, statement (b) holds. Since $\Delta F^*(P, (i, j)) \geq 0$ (by Corollary 2.1), it follows that $\Delta F^*(P, (i, j)) = 0$ if and only if $s_j \leq s(P, i)$.

Next, we calculate $\Delta F^*(P, (i, j))$ when $s_j > s(P, i)$. First, note that, by (2.14), the expression $F^*(P') - F^*(P)$ consists of two summands. The summand with coefficient $(1 - \alpha)\, p_i$ is equal to $\sum_{k \in \mathcal{K}} (s(P', k) - s(P, k)) = s_j - s(P, i)$. In order to calculate the other summand, we compare $V_i(P')$ with the V-values of the recommended items in P *before* adding (i, j), *i.e.*, the values $V_{[k]}(P)$ for $k = 1, \ldots, N$.

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

On the other hand, if $V_i(P') > V_{[N]}(P)$, content i will definitely feature in the recommendations list after adding it in j, which implies (2.20). We consider two subcases:

- i was already among the recommendations in P even before caching it in j, *i.e.*, $V_i(P) \geq V_{[N]}(P)$. In this case, since the streaming quality is better at j, a part of the marginal gain will come from the difference in V-value of i before and after adding (i, j). This means that $\sum_{k=1}^{N} \left(V_{[k]}(P') - V_{[k]}(P)\right) = V_i(P') - V_i(P)$.

- i was not recommended before caching it in j, *i.e.*, $V_i(P) < V_{[N]}(P)$. Since $V_i(P') > V_{[N]}(P) > V_i(P)$, content i gets into the recommendations list by replacing the N-th recommendation. Hence, a part of the marginal gain will come from the difference of the new V-value of i and the V-value of the $[N]$-th item in P, *i.e.*, $\sum_{k=1}^{N} \left(V_{[k]}(P') - V_{[k]}(P)\right) = V_i(P') - V_{[N]}(P)$.

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

We can now prove Lemma 2.7.

Proof of Lemma 2.7. For two placements A and B such that $A \subseteq B \subseteq \mathcal{P}$ and a pair $(i, j) \in \mathcal{P} \setminus B$, we need to prove (2.19). As before, A' and B' are the sets $A \cup (i, j)$ and $B \cup (i, j)$ respectively. Since $A \subseteq B$, eq. (2.16) (Lemma 2.6) implies that

$$s(A, i) \leq s(B, i). \tag{2.22}$$

In line with Lemma 2.9, we examine the following cases: *i)* $\Delta F^*(A, (i, j)) = 0$; *ii)* $\Delta F^*(A, (i, j)) > 0$. The first case is equivalent to $s_j \leq s(A, i)$, by Lemma 2.9.

Then, by (2.22), $s_j \leq s(B,i)$. We invoke once again Lemma 2.9 and we get that $\Delta F^*(B,(i,j)) = \Delta F^*(A,(i,j)) = 0$.

Concerning the second case, it is equivalent to $s_j > s(A,i)$ and $\Delta F^*(A,(i,j))$ is given by the formula in Lemma 2.9. We consider three subcases:

- $s_j \leq s(B,i)$;
- $s_j > s(B,i)$ and $V_i(B') \leq V_{[N]}(B)$;
- $s_j > s(B,i)$ and $V_i(B') > V_{[N]}(B)$.

In the first subcase, $\Delta F^*(B,(i,j)) = 0$, by Lemma 2.9 and, therefore, $\Delta F^*(A,(i,j)) > \Delta F^*(B,(i,j)) = 0$.

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

$$s_j = s(A',i) = s(B',i). \tag{2.23}$$

If $V_i(B') \leq V_{[N]}(B)$, Lemma 2.9, (2.22) and (2.23) imply that

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

If $V_i(B') > V_{[N]}(B)$, by (2.22), (2.23) and (2.18), it follows that

$$V_i(A') = V_i(B') > V_{[N]}(B) \geq V_{[N]}(A). \tag{2.24}$$

Combining this with (2.22) and Lemma 2.9, in order to prove $\Delta F^*(A,(i,j)) \geq \Delta F^*(B,(i,j))$, we only need to prove that

$$\max\{V_i(A), V_{[N]}(A)\} \leq \max\{V_i(B), V_{[N]}(B)\}. \tag{2.25}$$

It follows by (2.22) that $V_i(A) \leq V_i(B)$, and therefore $V_i(A) \leq \max\{V_i(B), V_{[N]}(B)\}$. Moreover, $V_{[N]}(A) \leq \max\{V_i(B), V_{[N]}(B)\}$, by (2.24). We then obtain (2.25) and this concludes the proof. □

Lemma 2.10. *The set function f^*, as defined in (2.10), is monotone increasing and submodular.*

Proof. By Lemma 2.4, $f^*(X) = \sum_{u \in \mathcal{U}} F_u^*(X)$. It is easy to prove that monotonicity and submodularity are preserved under non-negative linear combinations. Therefore, the result is an immediate consequence of Lemmas 2.6 and 2.7. □

2.3.3 MoSE Algorithms and Guarantees

We managed to prove through the decomposition in (2.11) that $f^*(X)$ is submodular. The theory on submodularity optimization suggests that different greedy algorithm variants give constant approximations for the outer problem, and thus for the MoSE problem. In fact, the factor of approximation depends on the constraints in (2.7).

The case of equal-sized contents

We define a greedy algorithm that we call the *MoSE algorithm*. This algorithm starts with a placement P consisting of empty caches (except for the large cache that contains the entire catalog) and greedily fills one by one all the available shots. In every round of selection, it calculates the marginal gain of f^* at P with respect to at most $C \cdot |\mathcal{K}|$ elements, *i.e.*, pairs (content, cache), by solving the Inner Algorithm (as subroutine). It then selects and adds to P the element that maximizes the marginal gain (ties broken arbitrarily), before the next selection round begins. The algorithm is summarized in Algorithm 2.

Algorithm 2: **MoSE algorithm (equal-sized contents)**

Input: C, $\{\mathcal{C}_j\}$,$\mathcal{U}$, $\mathcal{K}$, $\{N_u\}$, $\{s_{uj}\}$,$\{r_{ui}\}$, $\{\beta_u\}$, $\{\alpha_u\}$

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

Outer algorithm:

while *caches are not full, i.e.,* $|P^{(j)}| < \mathcal{C}_j$ *for all* j, **do**

  for *every (not full) cache* $j = 1, \ldots, C$, **do**

    for *every content* $i \in \mathcal{K}$ *s.t.* $(i, j) \notin P^{(j)}$, **do**

      Estimate $\Delta f^*(P, (i, j))$ by calling **Inner Algorithm(X)**; Store $\max \Delta f^*(P, (i, j))$.

    end

  end

  $(\eta, \theta) := \operatorname{argmax}_{(i,j)} \Delta f^*(P, (i, j))$.

  Add (η, θ) to P, *i.e.*, $P^{(\theta)} \leftarrow P^{(\theta)} \cup (\eta, \theta)$.

end

Return $X^* \leftrightarrow P, Y^* = f^*(X^*)$

Theorem 2.1 (Homogeneous sizes)**.** If we let OPT denote the optimal objective function value of the MoSE problem with equal-sized contents, and (X^*, Y^*) denote the feasible solution given by the MoSE algorithm, then

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

Proof. Since the constraints in (2.7) are matroid constraints, as in [28], the theory on submodular maximization [92] suggests that a 1/2-approximation is achievable by the above greedy algorithm. □

We should note here that a better than 1/2-approximation can be achieved in some special cases or at the cost of a larger running time, see for example [93] or [94].

The general case of contents of heterogeneous sizes

The fundamental difference between the two cases is the capacity constraints. The constraints in (2.7) in the general case are knapsack constraints. However, the MoSE

algorithm is oblivious of the content's size. The following algorithm is an adaptation of the MoSE algorithm that takes size into account. More precisely, in every round of selection, it adds to the cache the element (content, cache) that maximizes the ratio of marginal gain to the content's size, while satisfying the constraints in (2.7). It is summarized in Algorithm 3.

Algorithm 3: **s-MoSE algorithm (size-aware)**

Input : Same as in MoSE alg. and $\{\sigma_i\}$
Start with $P = \cup_{j=1}^{C} P^{(j)}$, where $P^{(j)} = \emptyset$, for all j;
Outer algorithm:
while *caches are not full, i.e.,* $\sum_{k \in P^{(j)}} \sigma_k < \mathcal{C}_j$, **do**
  for *every (not full) cache* $j = 1, \ldots, C$, **do**
    for *every content* $i \in \mathcal{K}$ *such that* $(i,j) \notin P^{(j)}$ *and* $\sigma_i \leq \mathcal{C}_j - \sum_{k \in P^{(j)}} \sigma_k$, **do**
      Estimate $\delta f^*(P,(i,j)) := \frac{\Delta f^*(P,(i,j))}{\sigma_i}$
      by calling **Inner Algorithm**(X);
      Store $\max_{(i,j)} \delta f^*(P,(i,j))$.
    end
  end
  $(\eta, \theta) :=$ argmax$_{(i,j)} \delta f^*(P,(i,j))$. Add it to P.
end
Return $X^* \leftrightarrow P, Y^* = f^*(X^*)$

Theorem 2.2 (Heterogeneous sizes)**.** If we let OPT_s denote the optimal objective function value of the MoSE problem in the general case (contents of heterogeneous sizes), and (X^*, Y^*), (X_s, Y_s) denote the feasible solutions given by the MoSE and s-MoSE algorithms respectively, then

$$\max\{f(X^*, Y^*), f(X_s, Y_s)\} \geq \frac{1 - 1/e}{2} OPT_s.$$

Proof. In the case of variable-sized contents, both MoSE and s-MoSE algorithms can perform arbitrarily badly [95]. According to the result in [95], it suffices to choose the maximum objective function value achieved by the two algorithms in order to achieve a $\frac{1-1/e}{2}$-approximation. □

It is worth mentioning that, in [96], the author describes an algorithm with better approximation ratio but at the cost of higher computational complexity. We elaborate on the complexity of the proposed algorithms in the next section.

Complexity, implementation speed-ups and distributed techniques

It is easy to see that the complexity of both the MoSE and the s-MoSE algorithms is the same. The algorithms need to run at most $\sum_{j=1}^{j=C} |\mathcal{C}_j|$ times in order to fill all caches. At each iteration, they evaluate the marginal gain of at most $C \cdot |\mathcal{K}|$ pairs

(content, cache). For every evaluation, they call the *Inner Algorithm* of complexity $O(|\mathcal{U}| \cdot |\mathcal{K}|)$ and complete $|\mathcal{U}|$ calculations that concern the non-recommendation part of the objective function. Therefore, the total complexity of the MoSE and s-MoSE algorithms is $O(|\mathcal{U}| \cdot |\mathcal{K}|^2 \cdot C \cdot \sum_{j=1}^{j=C} |\mathcal{C}_j|)$.

Implementation-wise, there is a way to speed up both algorithms by using the so-called lazy evaluations method [95]. This method takes advantage of the monotonicity and submodularity of the objective function in order to avoid unnecessary calculations in the selection process of the caching placement. More precisely, this method is based on the observation that the marginal value $\Delta f^*(P_k, (e, j))$ is bounded above by $\Delta f^*(P_{k-1}, (e, j))$ for all $e \in \mathcal{K}$, where $P_{k-1} \subset P_k$. Therefore, at each iteration, one could keep a sorted list of the marginal value of every content, and if $\Delta f^*(P_k, (e, j)) \geq \Delta f^*(P_{k-1}, (e', j))$, for all $e' \neq e$, then there is no need to calculate $\Delta f^*(P_k, (e', j))$ for $e' \neq e$, and the element e will be added in cache j. Furthermore, recent works propose methods for further acceleration, *e.g.,* a randomized greedy algorithm in [97].

Finally, we note that there is a technique suggested in the literature for distributed/multi-processor implementations of greedy algorithms of submodular maximization [98], such as the MoSE algorithm. More precisely, for a set of m processors/nodes, this technique starts by partitioning the ground set, *i.e.,* the catalog of contents, into m subsets. Then each processor solves *in parallel* the MoSE problem only on one of the subsets by applying our proposed policy. This leads to m solutions (caching allocation and users' recommendations) $DS_1, \ldots, DS_m$. Next, we define a new subset of the catalog by merging the contents for which the caching variable was equal to 1 in at least one of the previous solutions. We then run on this subset the MoSE algorithm which gives the solution MS. Finally, among the solutions $DS_1, \ldots, DS_m$ and MS, the one with the largest value of the objective function is selected. We note that, under some conditions, this implementation offers approximation guarantees that depend on the guarantees of the centralized algorithm and on the number m. For more details on the algorithm and these approximation guarantees we refer the reader to [98]. We will call *m-DMoSE* the algorithm described above, where m is the number of processors.

2.3.4 The Single-Cache Case

We study now the case where $C = 1$, *i.e.*, apart from the large cache C_0, there is only one cache. We prove that, in this case, the MoSE problem can be transformed into an Integer Linear Program (ILP) problem and, thus, common optimization methods can be applied to find the optimal solution for small problem's instances. This will be useful in the next section since it will allow us to compare the performance of our algorithm with the optimal joint policy.

We introduce the variable $\{z_{ui}\}_{i,u}$ such that $z_{ui} = x_i y_{ui}$. The objective of the MoSE problem in (2.7) becomes

$$g(X, Y, Z) = \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{K}} \Big[\frac{\alpha_u}{N} \left((s_{u1} - s_{u0})\, z_{ui} + s_{u0} y_{ui} \right) + (1 - \alpha_u)\, p_{ui} \left((s_{u1} - s_{u0}) x_i + s_{u0} \right) + \beta_u y_{ui} \varphi(r_{ui}) \Big]. \tag{2.26}$$

Therefore, the MoSE problem for $C = 1$ is equivalent to:

Z Problem.

$$\begin{aligned} \underset{X,Y,Z}{\text{maximize}} \quad & g(X,Y,Z) && (2.27)\\ \text{subject to} \quad & (2.7), (2.8), \\ & z_{ui} = x_i y_{ui}; && (2.28)\\ & x_{ij}, y_{ui}, z_{ui} \in \{0,1\}. && (2.29) \end{aligned}$$

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

Notice that, although $g(X,Y,Z)$ is linear in the variables X, Y and Z, the constraints (2.28) are nonlinear. However, we will prove that these constraints can be replaced by the following inequalities:

$$\begin{aligned} z_{ui} &\leq x_i, && (2.30)\\ z_{ui} &\leq y_{ui}, \text{ for all } u \in \mathcal{U}, i \in \mathcal{K}. && (2.31) \end{aligned}$$

Lemma 2.11. *The MoSE problem for $C = 1$ is equivalent to the following ILP problem:*

MoSE ILP problem.

$$\begin{aligned} \underset{X,Y,Z}{\text{maximize}} \quad & g(X,Y,Z) \\ \text{subject to} \quad & (2.7), (2.8), (2.29) - (2.31). \end{aligned}$$

Proof. It suffices to prove that a solution for the Z problem is also a solution for the MoSE ILP problem and the inverse. Let us assume that $(\bar{X}, \bar{Y}, \bar{Z})$ is a solution for the Z problem. Since $\bar{X}$ and $\bar{Y}$ are binary variables, the expression $\bar{z}_{ui} = \bar{x}_i \bar{y}_{ui}$ implies the inequalities (2.30) and (2.31). Hence, $(\bar{X}, \bar{Y}, \bar{Z})$ is also a solution for the MoSE ILP problem.

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

2.4 Performance Evaluation

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

2.4.1 Scenario 1

As a first step, we compare the performance of the MoSE algorithm with its distributed implementations 2-DMoSE and 4-DMoSE (*i.e.,* in 2 and 4 processors, see Sec. 13) and with the optimal policy (oracle). We consider a scenario with a single cache and the large cache C_0 that contains the entire catalog. As shown in Sec. 2.3.4, the MoSE problem for $C = 1$ can be transformed into an ILP problem. Therefore, in order to find the optimal policy (oracle), we use the standard MATLAB solver which employs methods such as branch-and-bound, cutting-plane method or exhaustive search.

We consider 20 users connected to the cache and a catalog of 200 unit-sized contents. We assume that the cache can fit 15 contents and every user receives $N = 2$ recommendations. The small size of the scenario is necessary to be able to calculate the optimal objective value. We will consider much larger scenarios subsequently. Moreover, the impact of the recommendations is determined by α_u, whose values follow a uniform distribution between 0.7 and 0.9 (in line with the statistics gathered on Netflix [51]). In this scenario, we consider a synthetic dataset for the relevances r_{ui} and the popularities p_{ui}. We chose p_{ui} such that the aggregate content popularities over all users, *i.e.*, $\sum_u p_{ui}$, follow a Zipf distribution (with parameter 0.6). Then, r_{ui} are chosen randomly in $[0, 1]$ such that their normalized value, *i.e.*, $r_{ui}/\sum_k r_{uk}$, are equal to p_{ui}, for every $i \in \mathcal{K}$, as in (2.1).

In this scenario, we measure the SQ as cache hits with the values s_{uj} as in (2.5), and the RQ (Def. 2.3) by considering $\varphi(r_{ui}) = \log(r_{ui})$. For a variety of values of $\beta_u = \beta > 0$, we queried the oracle and we calculated the MoSE given by the proposed algorithm and its distributed implementations. For some of the values β, Table 2.4 shows the approximation ratio achieved and Table 2.5 shows the average execution time per instance of the problem.

As we saw in Sec. 2.3.3, the ratio $f(X^*, Y^*)/OPT$ cannot be lower than 1/2. We observe that, in practice, the achieved ratio is much higher than 1/2, as is also observed for other submodular problems, *e.g.*, in [43]. In fact, among all the different values of β we considered (30 in total), the lowest observed approximation ratio was equal to 0.9757. Moreover, the approximation ratios achieved by the distributed algorithms 2-DMoSE and 4-DMoSE are also close to 1. However, as expected by the discussion in Sec. 13, they do not perform as well as the (centralized) MoSE algorithm for some instances of the problem (*e.g.,* for $\beta = 3.2$).

Observation 2.1. Our numerical results validate the theoretical approximation guarantees of our policy and also suggest a much better approximation ratio in practice.

Regarding Table 2.5, we see that the average execution time of the MoSE algorithm is of much lesser magnitude than the oracle's one. We note that we implemented MoSE with the lazy evaluations technique which avoids unnecessary calculations (see Sec. 13). When MoSE is implemented in 2 and 4 processors, the execution time decreases significantly, and we observe a 2x speedup.

Observation 2.2. Implementing the MoSE algorithm leads to significant savings in execution time when compared with the oracle. These savings can be further pronounced in the case of a distributed/multi-processor implementation.

Table 2.4: Approximation ratio $(f(X^*, Y^*)/OPT)$

Parameter β	0.01	1	1.7	3.2
Approx. ratio for MoSE*	1	0.9977	0.9979	1
Approx. ratio for 2-DMoSE	0.9998	0.9935	0.9979	0.9818
Approx. ratio for 4-DMoSE	0.9998	0.9750	0.9979	0.8836

*theoretical lower bound: 0.5 (see Theorem 2.1)

Table 2.5: Execution Time (AVG) Per Policy

MoSE	**2-DMoSE**	**4-DMoSE**	**Oracle**
0.0221 sec.	0.0147 sec.	0.0111 sec.	120.195 sec.

Next, we investigate if this close-to-optimal performance is reflected in the SQ-RQ tradeoffs. At the same time, we will compare these tradeoffs with the ones achieved by a proposed heuristic in the literature for a similar problem [67].

Cache-aware recommendations (CAwR). CAwR [67] makes caching and recommendation decisions at every cache independently. It decomposes the problem into the caching and recommendation steps. First, given the content preference distribution for every user (equivalent to the content popularity distribution p_{ui} or content relevances r_{ui} of our model) and the weight every user gives to recommendations (the α_u of our model), the aggregate request probability of every content is calculated. Then, the N items with the highest probability are cached. Note that, in the case of variable-sized contents, the cache allocation decisions are made by solving a $0-1$ knapsack problem, where the "value" of every content is the aforementioned probability and the "weight" is its size. Then, in the recommendation step, the recommendations are made partially by cached contents and by non-cached contents that are of high utility for the particular user. The balance between cached and non-cached contents is determined by a so-called distortion parameter $r_d \in [0, 1)$, which is similar to the parameter β of our model.

Figure 2.3 depicts the SQ-RQ tradeoffs given by the oracle, our policy, and CAwR as points in the plane. We obtained these tradeoffs for 30 different values of $\beta_u = \beta$ in the range $[0.01, 70]$ and the distortion parameter r_d. The RQ values (x-axis) are normalized with respect to the two "extreme" policies A and C (defined in Section 2.2.5). For example, RQ $= 50\%$ implies that the RQ value lies in the middle of the interval $[R_A, R_C]$, where R_A and R_C are the RQ values achieved by policies A and C respectively. Moreover, since s_{uj} are as in (2.5), the normalized SQ values (y-axis) give the cache hit rate.

We remind the reader that each of these points corresponds to a different objective tradeoff, between SQ and RQ, that a CP might have, *i.e.*, these curves could also be interpreted as Pareto curves. As we discussed in Sec. 2.2.4, β captures the weight we attach to the RQ compared to the SQ. For a small value of β (on the left), caching decisions are made based on the aggregate (over all users) interest in contents and recommendations concern mostly cached items. This leads to high SQ/cache hits but compromised RQ. As β increases, we trade off a better RQ for a worse SQ. In fact, a better RQ would imply recommendations to each user that are close to her tastes and it

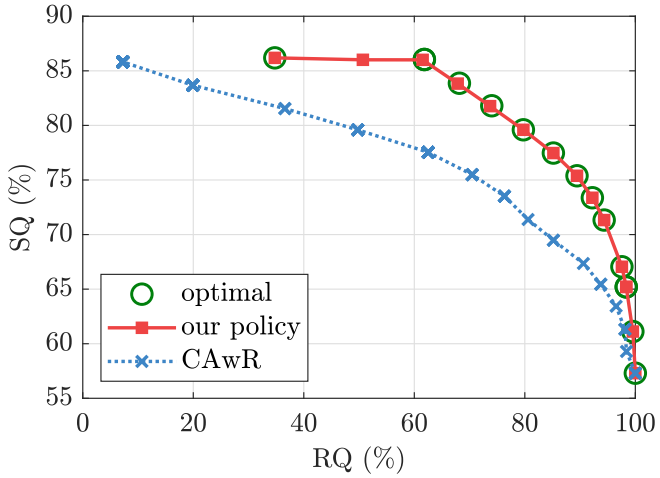


Figure 2.3: Scenario 1, SQ-RQ tradeoff points for some values of the parameters β and r_d.

is β that determines how close. As these tastes can differ from one user to the other and the cache capacity is limited, caches cannot store all the different recommended contents and this leads to decreased SQ/cache hits.

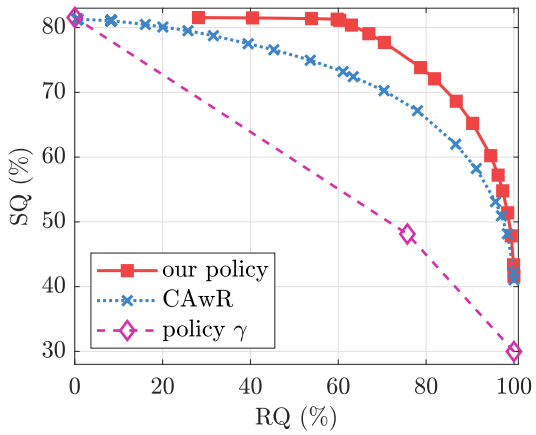
Observation 2.3. Our policy's tradeoff curve almost coincides with the optimal. Furthermore, it dominates the tradeoff curve of CAwR, *i.e.,* our policy outperforms CAwR in terms of at least SQ or RQ (or both).

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

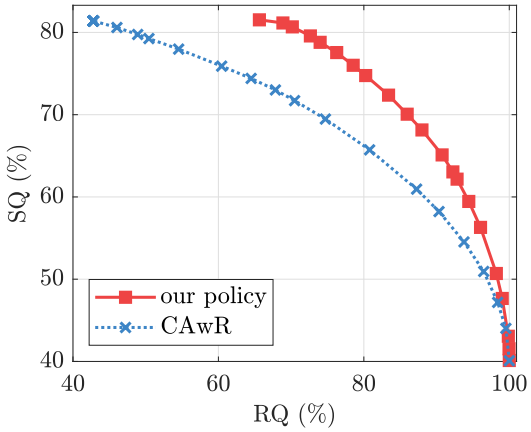
2.4.2 Scenario 2

We proceed with simulating larger scenarios. For this, we consider a single cache with 100 or 200 connected users and a catalog consisting of 6000 or 10000 contents[9]. We consider realistic values (according to footnote 1, p. 20) for cache capacity varying from 1% to 2.3% of the entire catalog. The probabilities α_u are chosen randomly in $[0.7, 0.9]$, in line with the statistics gathered on Netflix [51], and N varying from 2 to 10. For these experiments, we use a real dataset for the matrix of relevances r_{ui}:

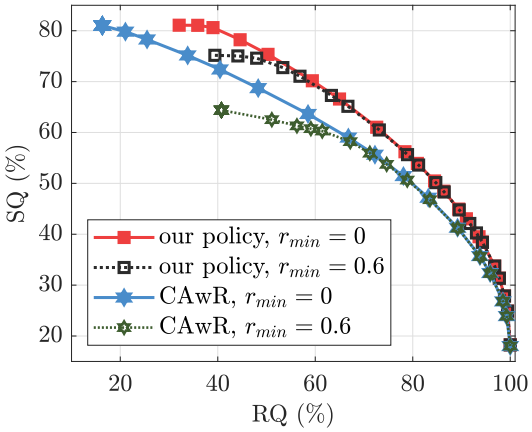
[9]Note that according to [99], the total number of titles (movies and TV shows) available on Netflix in the USA is equal to 5848.



(a) Equal-sized contents



(b) Variable-sized contents



(c) Equal-sized contents, $r_{min} = 0$ and 0.6

Figure 2.4: Scenario 2, SQ-RQ tradeoff points. Comparison of our policy with a policy proposed in the literature (CAwR) and the baseline policy γ.

MovieLens dataset. The MovieLens dataset [100] is a collection of 5-star movie ratings collected on MovieLens, an online movie recommendation service. This dataset has also been used in related works on caching and recommendations, *e.g.*, in [73]. Here, we used a variety of subsets of the total 20000263 ratings available in the original dataset. It is commonly assumed that the relevance of a content for a user is the *predicted* rating of this user for the content [82]. Therefore, we interpret the rating as the content relevance. Since the range of ratings is $0.5 - 5$ with 0.5 increments, we map every rating r to a random number in the interval $(r/5 - 0.1, r/5]$. As is common, this matrix is quite sparse. To obtain the missing ratings, we perform matrix completion through the TFOCS software [101]. TFOCS performs nuclear norm minimization in order to find the missing entries of a low-rank matrix.

Equal-sized contents

We assume that the contents are of unit size. We will show the performance improvement achieved by our policy over a baseline scheme, policy γ, and the earlier introduced CAwR. To begin, we define policy γ:

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

As before, we measure the SQ as cache hits and the RQ as $\sum_i \log(r_{ui})$. In Fig. 2.4(a), we plot the tradeoffs achieved by policy γ, CAwR, and our policy for different values of the parameters γ, r_d, and β respectively. In this instance, $N = 2$, which results in 3 possible objective values for policy γ. In fact, one point corresponds to recommending 2 cached items, the next one to recommending one among the cached items and the one of the highest relevance, and the last point to recommending the 2 first contents ranked in terms of relevance.

Observation 2.4. The SQ-RQ tradeoff curve of our policy dominates that of CAwR and that of the baseline policy γ in large, realistic scenarios, driven by real datasets.

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

Contents with heterogeneous sizes

So far, we have considered scenarios with equal-sized content (*e.g.*, chunks), as is often assumed in related work [28]. Here, we turn our attention to a scenario with contents of heterogeneous size, as analyzed in Section 12. The sizes of the contents were chosen in $\{1, 15\}$ and, according to the findings of [102] on YouTube videos, 90% of the contents have a size of at most 2 size units, while only 0.1% have a size over 10 size units. We adjust the cache capacity to 2.3% of the total size of the catalog $\sum_{i \in \mathcal{K}} s_i$. Figure 2.4(b) depicts the tradeoffs achieved by the two policies. In this context, our policy runs both the MoSE and s-MoSE algorithms and selects the maximum achieved objective function value between the two, as explained in Section 2.3.3. As expected, the difference between the tradeoff curves is similar to the one in Fig. 2.4(a). More specifically, we observe a relative gain of up to 63% in RQ and up to 15% in SQ of our policy with respect to CAwR.

Observation 2.5. Heterogeneous content sizes do not have an impact on the performance gains of our policy which, in this context, still outperforms existing schemes.

RQ-related constraints

While the previous results are promising, one might argue that the proposed policy could still recommend some rather unrelated contents, *i.e.,* contents of relevance r_{ui} close to 0, in favor of a higher objective value, or worse, that some users might receive much better recommendations, *i.e.,* tailored to their tastes, than others. For this reason, we will evaluate the performance of our policy and that of existing schemes when additional constraints on RQ are added to the problem. In particular, we measure RQ by considering φ as in (2.2). This leads to recommendations of contents whose relevance per user is at least r_{min}. Since the recommendation decisions are made by solving the "inner problem " (as explained in Section 2.3.2), the caching decisions also take into account this constraint. Subsequently, we adjust CAwR such that, at the recommendation step, the contents with $r_{ui} < r_{min}$ cannot be recommended to the user.

Figure 2.4(c) demonstrates that, for values of β close to 0 and values of r_d close to 1, the performance in SQ for both policies naturally drops when $r_{min} = 0.6$ in comparison to the performance when $r_{min} = 0$. This is because fewer contents can be recommended per user and these can largely differ from one user to the next. Therefore, only a few of them can be cached due to the limited cache capacity, and less cache hits will occur. In fact, in the dataset used for this experiment, on average, for every user, only 3% of the catalog is of relevance/utility greater than or equal to 0.6.

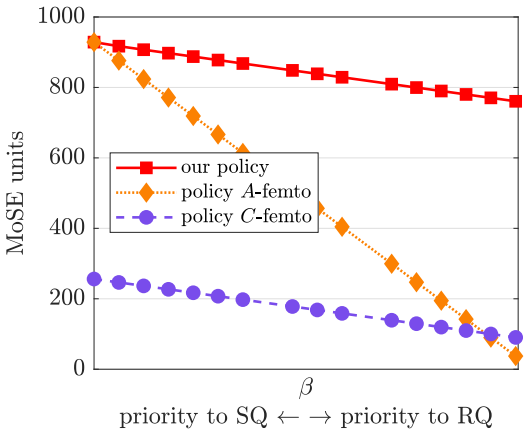
Observation 2.6. Our policy does not choose to radically compromise RQ, leading to similar performance tradeoffs even when additional strict constraints on RQ are imposed.

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

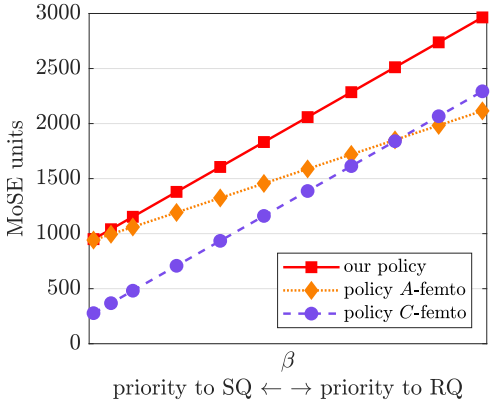
2.4.3 Scenario 3

So far, we studied scenarios with a single cache in order to be able to compare the performance of the proposed policy with the related work. We remind the reader that the approximation guarantees of our policy hold for arbitrary networks of caches where users might have access to more than one cache. The algorithm proposed in [28] makes caching decisions taking into account such coverage overlaps. However, this problem setup does not contain recommendations. In this scenario, we evaluate the performance, in terms of MoSE, of our policy and some non-joint policies whose caching decisions are made according to [28].

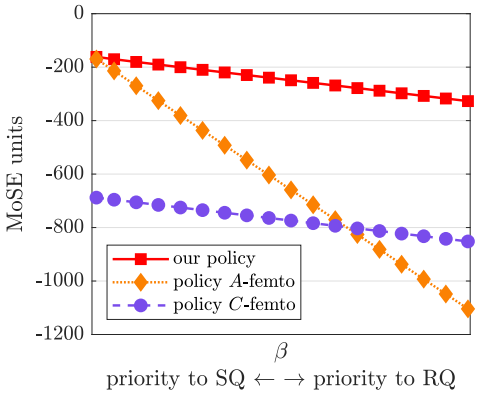
We consider a cellular network in a square area of 500 m^2 with 9 small-cell BS (helpers) and a macro-cell base station (the large cache of our scenario). A total of 100 users are placed in the area according to a homogeneous Poisson point process (in line with the related works [28], [73]), while helpers are placed in a grid. Helpers' communication ranges are set to 200 m, which results in an average of 3.5 helpers per user. In this scenario, we will measure the SQ as a function of the estimated bitrate. More precisely, we assume that $s_{uj} = \psi(b_{uj})$, where b_{uj} are the estimated bitrate that can be supported



(a) ψ linear, φ logarithmic



(b) ψ and φ linear



(c) ψ and φ logarithmic

Figure 2.5: Scenario 3, MoSE versus β for different types of SQ ($s_{uj} = \psi(b_{uj})$) and RQ ($\varphi(r_{ui})$) values/functions. Comparison of our policy with the A-femto and C-femto policies.

between user u and cache j and ψ is an increasing function of b_{uj}. Without loss of generality, we assume that the rate from the large cache (or macro-cell cache) C_0 is 0.5 Mbps, while the b_{uj} values for edge caches are chosen randomly between 2 and 15 Mbps[10]. In fact, the required Internet connection speed on YouTube [103] is 0.5 Mbps, and the recommended speed to watch a video in $4K$ is 20 Mbps.

We consider a subset of 6000 unit-sized contents of the Movielens dataset and α_u and φ, as in Scenario 2. We set the helper's capacity to 1.5% of the catalog size and $N = 5$. We will compare the performance of our policy, in terms of the MoSE, with two policies that are based on the algorithm proposed in [28]:

[10]As we are interested in capturing both wired (CDN) and wireless (femto-caching) scenarios, the physical layer details are beyond the scope of this analysis.

A-femto and C-femto policies. They generalize the policies A and C described in Section 2.2.5 in a network of caches. They both make the caching decisions based on the femto-caching policy proposed in [28] that takes into account the fact that users have access to multiple caches in the network. Then, the recommendations part of the policies A and C is applied.

For ψ being the identity function, *i.e.,* $s_{uj} = b_{uj}$ and for different values of $\beta > 0$, the achieved MoSE of our policy, the A-femto, and the C-femto policies are shown in Fig. 2.5(a). We observe that, for β close to 0, *i.e.*, priority is given to the SQ, the performance of the A-femto policy and our policy coincide. This is because both policies make the same caching and recommendation decisions, *i.e.*, cache and recommend the most popular items. The MoSE achieved by the C-femto policy is lower since, although it provides the best RQ, the recommended items are not necessarily among the cached ones and thus, they need to be retrieved from the large cache at the cost of lower SQ. In fact, this is illustrated in small scale in the toy example in Sec. 2.2.5. As β increases, the priority moves towards RQ, and hence, the performance of the A-femto policy starts to worsen until it is dominated by the one of the C-femto policy. Our policy continues to perform better than both of them as a result of caching and recommendation orchestration. Furthermore, the performance gap between our policy and the C-femto policy remains constant.

Observation 2.7. The performance gains of the proposed policy over non-joint policies are prominent in generic networks of caches as well.

In the cases studied above, we have considered the SQ and RQ functions (*i.e.*, ψ and φ respectively) being the identity or the function in (2.5) and the logarithmic function respectively. One might wonder how these choices affect the performance of our policy. For this reason, we explore their impact here. We ran on the same dataset as above the experiment for: *i)* both ψ and φ being linear functions (Fig. 2.5(b)), and *ii)* both ψ and φ being logarithmic functions (Fig. 2.5(c)). We remind the reader that the rationale for the logarithmic function has been elaborated in Sec. 2.2.4. We notice that the relative improvement in performance of the proposed policy in comparison to the A-femto and C-femto policies are similar for the different choices of functions considered (Fig. 2.5(a)-(c)). We note that a variety of coefficients of these functions have been considered in every case. Furthermore, we observe that even though the range of the y-axis varies in Fig. 2.5 (a)-(c), the relative gains are similar and the preceding analysis on the relative performance of the three policies holds in every case.

Observation 2.8. The performance improvements are consistent for different choices of SQ and RQ functions.

2.5 Related Work

Hierarchical caching. Optimization of hierarchical caching (*e.g.*, CDNs or ICNs) has been widely explored both in the context of wired [81] and wireless networks [28]. Various aspects of this problem have been explored such as caching for different video streaming

qualities [41] etc. See, for example, a recent survey on caching in [27]. Nevertheless, these works are oblivious to the impact of the recommendations, beyond the simple (usually IRM) popularity model used as input.

Caching-recommendation interplay. We refer the reader to Sec. 1.4.2 for an overview of the related work on this topic. However, we note that [67] and [68] tackle the joint problem in a problem setup that is closer to ours. The heuristics proposed in these works decompose the joint problem into caching and recommendations subproblems.

Joint optimization theory. Submodularity-based proofs for caching-related problems have flourished since the seminal paper of [28], where the focus is on one set of variables (caching). Joint optimization in a discrete setup has been widely considered in many caching-related and other networking problems. For example, routing or user association variables (subject to capacity constraints) are treated jointly with caching in [104]. It is shown that these problems can be mapped to variants of the facility location problem, and related approximation algorithms can be adapted to the problem. However, facility location algorithms are not applicable to the joint caching and recommendations problem at hand. Moreover, we note that iterative methods based on Benders decomposition have also been considered for jointly optimizing the economic costs of caching policies [48]. The decomposition and submodularity method we use is similar in spirit to the methods in [105] and [43]. While the former studies quite a different problem than ours, the latter proposes an approximation algorithm for the joint caching and routing problem in cache networks.

2.6 Conclusion

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

Chapter 3

Models for Cooperative Recommendations

3.1 Introduction

Recommender systems (RSs) permeate today's streaming services, and are substantially affecting the content requests issued by their subscribers. Indeed, by proposing contents that are relevant to their users' interests, Content Providers (CPs) can increase the viewing activity in their platforms, reduce user churn, and eventually boost their revenues [51]. Therefore, it is not surprising that CPs comprehend the business value of these systems and invest research and financial resources to improve their accuracy.

At the same time, recommendations can be leveraged by content caching networks to steer user requests towards nearby-cached contents. These caching networks are either today's traditional Content Delivery Networks (CDNs) or edge cache providers in future wireless architectures (we will use, hereafter in this chapter, the term CDN to imply any such caching network provider). The terms of cache/network-friendly recommendations capture exactly this idea: recommendations aiming to reduce the CDNs' routing expenses without deviating irreparably from the users' viewing preferences. This idea may not only reduce the operating and retrieval costs of CDNs, but can also improve the service quality for the users by achieving smaller viewing start-up delays and/or higher bitrates of the streamed content (as was shown in Chapter 2).

Clearly, RSs have already become a powerful tool affecting all key stakeholders in the content distribution ecosystem. As their influence further increases, it is imperative to ensure they will foster synergies instead of creating misaligned incentives. Specifically, a hitherto unexplored aspect in this context is the tension between CPs and CDNs when it comes to recommendations: the cache-friendly recommendations of CDNs may deviate from the users' interests, and thus, negatively affect the CPs' revenues; while the CPs' recommendations might induce costly data transfers for the CDNs. This problem is more pronounced in the case where Over-The-Top (OTT) CPs lease CDN infrastructure to deliver their services, but appears also in content streaming platforms with self-owned caching infrastructure.

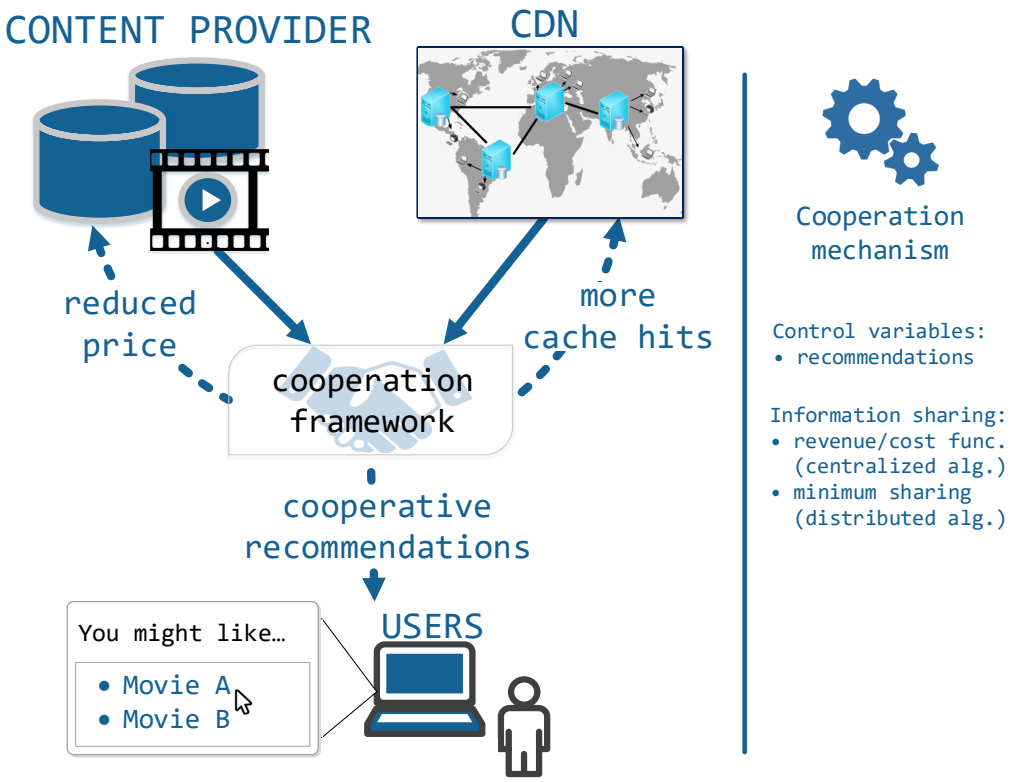


Figure 3.1: The CP and CDN cooperate by agreeing on the recommendations the users receive. The incentives of the cooperation are provided by the reduced price the CP is charged for the content delivery and the resulting increase in the number of cache hits that leads to lower retrieval costs for the CDN.

The goal of this chapter is to investigate this new problem by: 1) understanding and modeling the root causes of the CP's and CDN's potential conflicts when it comes to recommendations; 2) proposing a cooperation framework to enable their agreement; and 3) designing algorithms for realizing this coordination based on the information the two entities want to disclose. The core of our proposal is the following simple and practical idea: the CDN charges lower content delivery fees to the CP when the latter agrees to tune its recommendations towards cached contents. This discount will balance the CP's expected viewing gains with the CDN's induced savings on retrieval costs (see Fig. 3.1).

Devising these *cooperative recommendations* is a new and highly non-trivial problem whose nature and complexity cannot be properly handled by existing approaches for cache-friendly recommendations. Moreover, we explore how this problem can be tackled in conjunction with the CDN's caching policy. Even though adding the caching as a variable makes the problem harder, this extended problem has the potential to further increase the cooperation gains.

3.2 Problem Setup

3.2.1 Recommendations, Content Requests, and Caching

In this chapter, we present a cooperation scheme between the CP and CDN on the basis of the recommendations the former offers to its users. Following the current business models for the two entities, we model their utility functions that represent their profit from the OTT market. For the convenience of the reader, we provide a summary of the notation introduced in this chapter in Table 3.1.

Content Recommendation Model: The CP owns a content catalog $\mathcal{K}$ that is accessible to a set $\mathcal{U}$ of users through the CP's OTT service. A (personalized) list of N_u items are recommended to each user $u \in \mathcal{U}$. The recommendations are based on the predicted relevance of each content to the user's tastes, viewing history, context, etc. These relevances (sometimes also called "scores" or "rankings") are calculated by today's state-of-the-art RS (that is employed by the CP) using techniques such as collaborative filtering, deep neural networks, reinforcement learning, etc. [51], [106]. We denote by $r_{ui} \in [0, 1]$ these relevances. Typically, the CP would select the N_u items with the highest r_{ui} or the ones with the highest expected revenue to feature the recommendations list of user u [51]. In particular, the recommendation policy could depend on the revenue model of the CP (as we will see in the next section), *i.e.,* whether the revenues depend mostly on user engagement or/and advertisements (ad impressions). In this chapter, the recommendation decisions (*i.e.,* deciding which contents will appear in the user's recommendations list) are made not only based on the utilities r_{ui} but also on the cooperation terms.

Definition 3.1. Our problem considers two sets of recommendations:

- *(Input) Baseline Recommendations:* $Y^b = (y^b_{ui} \in \{0, 1\}, u \in \mathcal{U}, i \in \mathcal{K})$, where $y^b_{ui} = 1$ if content i is recommended to user u. These are decided by the CP before any cooperation and are *input parameters* for our problem. For example, these could be the top N_u most relevant contents (to each user), as mentioned above.

- *Cooperative Recommendation Variables:* $Y = (y_{ui} \in [0, 1], u \in \mathcal{U}, i \in \mathcal{K})$, which are the *probabilistic* recommendation variables optimized jointly by the CP and CDN. These are the *control variables* of our problem.

In contrast to Chapter 2, the control variables are continuous here. Using continuous variables for the cooperative recommendations allows the CP to provide some variety to the recommendations it offers to the same user from session to session.

Content Request Model: As in Chapter 2, each user u makes content requests according to the following model:

- The user follows the recommendations with probability α_u; where, w.l.o.g. each of the N_u items is considered equally likely to be requested. Hence, each recommended content is requested by the user with probability α_u/N_u.

- With probability $(1-\alpha_u)$, the user ignores the recommendations and requests a content $i \in \mathcal{K}$ of the catalog with probability p_i.

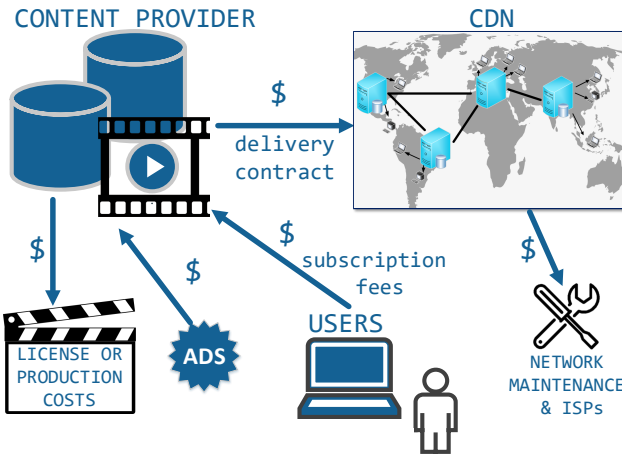


Figure 3.2: Sources of revenue and cost for the CP and the CDN as described in Sec. 3.2.2. The delivery contract is the foundation of their economic relationship.

Content Caching Model: A CP subscribes to a CDN provider through a Service Level Agreement (SLA) for the delivery of the contents to the users. The CDN manages a set of C caches with capacity $\mathcal{C}_j$, $j = 1, \ldots, C$, where $\mathcal{C}_j \ll |\mathcal{K}|$. Moreover, there is a root cache C_0 that stores all the contents. W.l.o.g., we consider equal-sized contents (divided in chunks), as is common in related works (*e.g.,* [107], [35]). The CDN optimizes the caching decisions based on performance (e.g., latency, cache hits) and cost criteria (routing costs). These decisions are described as follows:

Definition 3.2. The (input) baseline caching is denoted by $X^b = (x_i^b \in \{0,1\}, i \in \mathcal{K}, j = 1, \ldots, C)$, where $x_{ij}^b = 1$ if content i is fully stored in cache j. These are determined by the CDN before any cooperation and are input parameters.

To better focus on the mechanics of the cooperation mechanism, we will develop our framework in the context of cache-friendly recommendations, *i.e.,* assuming that the caching policy is decided at a different timescale than the recommendations and is fixed during the cooperation. We revisit caching variables, and how these could potentially also be designed jointly with recommendations later, in Sec. 3.4.

3.2.2 Revenue/Cost Models and Utility Functions

We will now discuss the various sources of revenues and costs for the CP and CDN (illustrated in Fig. 3.2) in order to define their utility functions. While these sources can, of course, be highly nuanced from scenario to scenario, we propose a model that tries to capture key elements while staying tractable.

CP revenues: When a user u requests a content i, this content is associated with an expected revenue R_{ui} that is estimated by the CP as a result of its revenue model and the associated costs related to the purchase of contents (through licensing or production). In fact, there are 4 main revenue models for streaming platforms [108]:

- ad-based (*e.g.,* YouTube (non-Premium))
- subscription-based (*e.g.,* Netflix)
- transaction-based or pay-per-view (*e.g.,* iTunes)
- hybrid model (*e.g.,* Amazon Prime Video).

The CP's expected revenue depends on the content relevances r_{ui} in a non-trivial way. For this reason, we capture this relation by a fairly generic model:

$$R_{ui} = \phi_{ui}(r_{ui}), \tag{3.1}$$

where ϕ_{ui} can be any nondecreasing function of r_{ui} that describes the impact of user's (predicted) interest in a content on the CP's revenues[1]. For example, ϕ_{ui} could be related to the probability of a user abandoning the viewing session as a function of r_{ui}.

CP costs/CDN revenues: The delivery of a requested content is done by the CDN that charges the CP on a basis of the amount of requests. We remind the reader that these charges apply to CPs without an in-house CDN, which is still the case for a large number of CPs, *e.g.,* Disney+, Hulu. We assume that the CP has to pay λ currency units *per request*[2].

CDN costs: The main source of expenditures for the CDN is the cost related to the delivery of a requested content to the user[3]. We let $\mathcal{C}(u)$ be the subset of caches that a user u has access to including the root cache (which is accessible by every user). A request for content i by user u may be served by at least one of the small caches in $\mathcal{C}(u)$ where i is stored. If the content is not cached, it will be served by the root cache C_0.

We assume that every link between user u and the caches in the set $\mathcal{C}(u)$ is characterized by a delivery (retrieval) cost (for the CDN). We let k_{uj} denote this cost per request for user u by the cache j. The value of k_{uj} can be estimated as a result of transit fees the CDN pays to transit networks or Internet Service Providers (ISPs) to retrieve the content from the origin servers of the CPs and make it available to the users. Moreover, they can include maintenance-related costs, *e.g.,* related to storage capacity, hardware, estate, energy, etc [45]. The delivery cost from the root cache C_0 to user u is k_{u0}, where $k_{u0} > k_{uj}$ for all $j = 1, \ldots, C$. The CDN serves each request through the lowest-cost cache that has the requested item, as is common in most caching setups [28, 81]. We denote the sequence of increasing user-cache costs by $k_{u(1)}, k_{u(2)}, \ldots, k_{|\mathcal{C}(u)|}$. Then, based on the caching decisions X^b, the delivery cost for content i by user u is:

$$K_{ui}(X^b) = \sum_{j=1}^{|\mathcal{C}(u)|} \left[k_{u(j)} x^b_{i(j)} \prod_{l=1}^{j-1} \left(1 - x^b_{i(l)}\right) \right]. \tag{3.2}$$

[1]These functions are built by the CPs using historical data; and are typically concave capturing diminishing returns on the relevances r_{ui}.

[2]We can extend our model to different pricing schemes with small modifications. For example, if the CDN charges the CP per content/chunk *delivered*, then λ can be multiplied by the abandonment probability.

[3]We focus here on the revenues and costs that are relevant in our model. Obviously, additional costs are incurred by both the CP and CDN, *e.g.,* fixed business costs.

Table 3.1: Notation Summary for Chapter 3

Content Requests and Recommendations

$\mathcal{K}$	catalog of contents
$\mathcal{U}$	set of users in the network
N_u	number of recommended contents for the user u
α_u	probability that user u follows the recommendations
p_i	probability that a user requests content i while *not* following the recommendations

Revenues and Costs

λ	price per chunk requested that the CP pays to the CDN
ρ	discount on the delivery price λ, $\rho \in (0, 1)$
R_{ui}	CP's revenue (expected) per view from user u for content i
r_{ui}	relevance (predicted) of content i to user u
K_{ui}	CDN's cost of delivering content i to user u

Input Parameters

y^b_{ui}	(input) baseline recommendations, before any cooperation
x^b_{ij}	(input) caching allocation, before any cooperation

Variables

y_{ui}	cooperative recommendation variable corresponding to user u and content i for the centralized solution
$\psi_{ui}, \widetilde{\psi}_{ui}$	cooperative recommendation variables for the distributed algorithm, as decided by the CP and the CDN respectively

CP's and CDN's utilities before cooperation: Based on the above problem setup and revenue models, we can now derive the total *utility* (revenues minus costs) each of the two parties enjoys before cooperating. We define the baseline (initial) utility of the CP before any cooperation as the expected revenue minus the expected price it has to pay to the CDN:

$$U^b = \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{K}} \frac{\alpha_u}{N_u} y^b_{ui} (R_{ui} - \lambda). \tag{3.3}$$

We do not account for the revenue that comes from the content requests that are not a result of recommendations, i.e., when, with probability $1 - \alpha_u$, the user does not follow any of the recommendations. It is easy to see that these requests do not affect the cooperation (whose control variables are the recommendations). Moreover, note that the definition of U^b is generic and does not depend on how the CP devises the standard recommendations (*i.e.,* the values y^b_{ui}).

Given the caching and recommendation decisions before the CP-CDN cooperation, the baseline (initial) utility of the CDN is expressed as the expected revenue (from the delivery contract) minus the expected delivery (or retrieval) costs:

$$\widetilde{U}^b = \sum_{u \in \mathcal{U}} \sum_{i \in \mathcal{K}} \frac{\alpha_u}{N_u} y^b_{ui} \big(\lambda - K_{ui}\big). \tag{3.4}$$

3.2.3 Towards Cooperative Decisions

The goal of our cooperative framework is to improve the aforementioned utilities of *both* parties[4]. As explained earlier, such improvements can result by motivating the CP to modify some of its original recommendations towards lower cost items (*e.g.,* cached ones). To ensure that the CP will not lose revenue from these modifications (we remind the reader that this revenue relates to how related the recommended contents are for users, see (3.1)) we assume the CDN offers a discount on the content delivery fees of such "lower cost" content. In particular, we let ρ denote this discount on the price λ, where $0 < \rho < 1$. The value of ρ is either set[5] by the CDN or by a regulatory authority (who acts as a mediator for their cooperation). Then, the new price the CP would have to pay to the CDN is

$$\Lambda_{ui} = \lambda[1 + \rho(y^b_{ui} - 1)]. \tag{3.5}$$

Specifically, if a content i is recommended now but it was not before the cooperation (*i.e.*, $y_{ui} > 0$ and $y^b_{ui} = 0$), then the discount ρ applies. If, on the contrary, the content continues to be recommended (even partially) as before (*i.e.*, $y_{ui} > 0$ and $y^b_{ui} = 1$), no discount applies. One could also include in the model an extra requirement that the discount is applied to newly recommended content that is also cached. Our problem formulation, to follow, is applicable to either scenario, so w.l.o.g. we will focus on the former. We note that the requests that do not come through recommendations are not subject to any discount. Then, the new utility for the CP and CDN are:

$$U = \sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{K}} \frac{\alpha_u}{N_u} y_{ui}(R_{ui} - \Lambda_{ui}), \tag{3.6}$$

$$\widetilde{U} = \sum_{u\in\mathcal{U}}\sum_{i\in\mathcal{K}} \frac{\alpha_u}{N_u} y_{ui}\big(\Lambda_{ui} - K_{ui}\big). \tag{3.7}$$

3.2.4 Toy Example

To better understand the cooperation model and the tradeoffs involved, we present a toy example depicted in Fig. 3.3. We consider a scenario with two users, a catalog of four equal-sized contents and a single cache with capacity 2. Upon request, a content is served by the cache, if it is cached there. Otherwise, it will be served by the root. For simplicity, we assume that the users will receive a single recommendation that will follow with probability 1 and that all contents are equally relevant to the users. The CP pays to the CDN \$0.5 per request (outside of any cooperation) while the CDN offers to the CP a discount of 30% on the delivery fees if they cooperate and the CP modifies its recommendations. The revenues and costs related to the contents are depicted in the table on the top right of the figure.

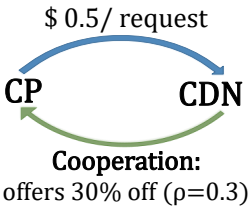
[4]We are specifically interested to maximize the gains and ensure they are "fairly" shared; we elaborate on the fairness framework in the next section.

[5]We will discuss in Sec. 3.5 how the value of ρ could be chosen in practice.

In the table on the bottom, we see the pre-cooperation (baseline) and cooperative decisions taken as well as the utilities (profits) of the two entities. In particular, outside of any cooperation, the CP would recommend the contents that will bring the highest revenue, *i.e.,* Movie A to User 1 and Movie B to User 2, while the CDN would cache some contents without knowledge of the recommendations and how they shape the requests. We assume that Movies C and D are cached based on the aggregate popularity observed in a period of time prior to the cooperation.

Therefore, the requests for the recommended contents will lead to cache misses and extra retrieval costs (for the CDN). However, if the two entities cooperate, then it would be better to recommend Movie C to both users. In that case, the CP will pay reduced delivery fees (that will compensate for the differences in the revenues R_{ui}), while the CDN will avoid the extra costs. We see that, already in this toy example, the cooperation leads to gains of at least 20% for each entity. Note that any other solution, *e.g.,* recommending movie D to both users, would result in worse gains for at least one entity and thus in an *unfair* allocation of the cooperation gains.

In this example, it is easy to guess how to find the cooperative recommendation policy that boosts both entities' profits. However, this task becomes significantly harder for bigger scenarios (large content catalogs, multiple recommendations per user, etc.). Moreover, one might wonder: would the CP and CDN be willing to exchange information on their utility functions in order to find the solution (since these functions constitute sensitive business information)? And how the cooperative recommendations can impact the users? To this end, in the next section, we formulate a cooperation mechanism while addressing these concerns.



	CP's revenue (R_{ui}) CACHED by the CDN				CDN's delivery cost (K_{ui} (X^b))
	Movie A	Movie B	Movie C	Movie D	
User 1	$1	$0.2	$0.95	$0.94	• $0.01 if cached • $0.22 if not cached
User 2	$0.1	$1	$0.95	$0.3	

For U=2 (two users), N_u=1 (single recomm.), α_u=1, and caching allocation(X^b): Movie C & Movie D

Non-cooperation (baseline scheme):	**Cooperation**: gains
Recomm. (Y^b): Movie A → User 1, B→ User 2	Recomm. (Y): Movie C → User 1 & User 2
• CP's util. (U^b): 2*(1-0.5)= $ 1	• CP's util. (U): 2*(0.95-0.35)=$ 1.2 (+ 20 %)
• CDN's util. ($\tilde{U}^b$): 2*(0.5-0.22)=$0.56	• CDN's util. ($\tilde{U}$): 2*(0.35-0.01)=$ 0.68 (+ 21 %)

Figure 3.3: Toy example presented in Sec. 3.2.4. In this example, the CP-CDN cooperation leads to financial gains of 20% and 21% respectively. These gains derive from the discount on the delivery fees (for the CP) and the fetching/retrieval savings (for the CDN).

3.3 Problem Formulation and Algorithms

The toy example above illustrated that the CP-CDN cooperation should provide adequate incentives for both entities. This means that the cooperative recommendation policy should be devised in a way that it satisfies: $U \geq U^b$ and $\widetilde{U} \geq \widetilde{U}^b$. As explained earlier, the CDN will propose a discount on the delivery fees in order to incentivize the CP to tune its recommendations towards cached contents. Given this discount, the two parties will try to benefit as follows:

- *CDN:* it increases cache hits (through cache-friendly recommendations) and thus it reduces the delivery costs (term K_{ui} in (3.7)). These cost savings will compensate the lower delivery fees (term Λ_{ui} in (3.7)).
- *CP:* it modifies the recommendations only if the cooperative ones lead to minor loses in expected revenue R_{ui}, that can be amortized by the applied fee reduction.

Moreover, both parties have the following concrete goals: 1) benefit as much as possible (hence the need for an *optimization framework*), and 2) reach a "fair" agreement (that is, the allocation of the cooperation gains should be Pareto optimal, *i.e.,* there is no other solution that would benefit one party more without deteriorating the other's gains).

Having these desired properties as guideline, we model this optimization problem as a Nash Bargaining Solution (NBS) [109, 110]. The NBS is defined as the maximization of the product of payoffs (*i.e.,* the utility gains) of the two entities subject to individual rationality constraints, or equivalently the maximization of the logarithm of this product where the constraints are implicit in the domain of the logarithms [110]. Therefore, the NBS would be

$$\max_{Y} \left[\log(U(Y) - U^b) + \log(\widetilde{U}(Y) - \widetilde{U}^b)\right], \tag{3.8}$$

where $U(Y) - U^b$ and $\widetilde{U}(Y) - \widetilde{U}^b$ represent the gains in utility of the CP and the CDN from a potential cooperation.

3.3.1 Centralized Cooperative Recommendations

We will first formulate and study the centralized problem where the two entities share their cost/revenue functions.

CCR: Centralized Cooperative Recommendations.

$$\min_{Y} \quad \left[-\log\left(\sum_{u,i} \frac{\alpha_u}{N_u} y_{ui}(R_{ui} - \Lambda_{ui}) - U^b\right) \right.$$
$$\left. -\log\left(\sum_{u,i} \frac{\alpha_u}{N_u} y_{ui}(\Lambda_{ui} - K_{ui}) - \widetilde{U}^b\right)\right] \tag{3.9}$$

$$\text{s.t.} \quad \sum_{i \in \mathcal{K}} y_{ui} = N_u, \forall u \in \mathcal{U}, \tag{3.10}$$

$$y_{ui} \in [0,1], \forall u \in \mathcal{U}, i \in \mathcal{K}, \tag{3.11}$$

where the baseline utilities U^b and $\widetilde{U}^b$ are defined in (3.3) and (3.4). The constraints in (3.10) suggest that each user receives N_u recommendations[6]. Moreover, we note that the inequalities

$$U - U^b = \sum_{u,i} \frac{\alpha_u}{N_u} y_{ui}(R_{ui} - \Lambda_{ui}) - U^b \geq 0, \tag{3.12}$$

$$\widetilde{U} - \widetilde{U}^b = \sum_{u,i} \frac{\alpha_u}{N_u} y_{ui}(\Lambda_{ui} - K_{ui}) - \widetilde{U}^b \geq 0 \tag{3.13}$$

are implicit constraints as the domain of the logarithms must be non-negative. In fact, $(U^b, \widetilde{U}^b)$ is the "disagreement point" of the cooperation [109]: if $U < U^b$ or $\widetilde{U} < \widetilde{U}^b$, there will be no feasible solution and, thus, no agreement on cooperation. Then the CP will keep its baseline recommendations Y^b while the CDN will not provide a price discount.

By formulating the optimization problem in this way, the solution uniquely satisfies the Nash's axioms [109, 110]. First, the solution is Pareto optimal. Furthermore, due to the implicit domain constraints, the payoff of every entity is no worse than the payoff it would get outside of any cooperation, *i.e.*, $(U^b, \widetilde{U}^b)$. Finally, if the positions of the two entities (in terms of utility functions and the disagreement point) are symmetric, then the solution treats them symmetrically.

The next lemma shows that the CCR problem is tractable.

Lemma 3.1. *The CCR Problem is convex.*

Proof. The objective function is convex since the logarithm is a concave function and the arguments of the logarithms are linear functions of Y. Moreover, the problem's constraints are linear. □

As a result of Lemma 3.1, standard interior-point or dual methods would efficiently give the optimal solution. We summarize below the algorithm to devise the cooperative recommendations in a centralized manner. This algorithm could be run either by a trusted third party (cooperation mediator) that collects the necessary information or by the two entities together.

The CCR Algorithm. The CP communicates the values α_u, N_u, Y^b, U^b and its utility function U. The CDN communicates the discount ρ, the value of $\widetilde{U}^b$, and its utility function $\widetilde{U}$. Then, the CCR Problem is solved through standard dual or interior-point methods. It returns Y^*, the optimal cooperative recommendation policy.

Remark 3.1. The proliferation of encrypted user-cache communication through HTTPS/TLS requests is considered an obstacle for efficient content caching within the OTT services. However, there are protocols proposed in literature that can ensure that the CDN's caches are blind to the cached contents (*e.g.*, [111]). Similar protocols could be embedded in our framework since the CDN needs only to estimate the retrieval cost of the cached items (that could be encrypted). Designing such a protocol is an interesting direction for future work.

[6] If the solution Y^* contains more than N_u positive values (due to the probabilistic model) we can easily select exactly N_u following the technique in [107] and being compatible with Y^* on expectation.

3.3.2 Distributed Cooperative Recommendations with Minimum Information Sharing

As explained earlier, in order to solve the CCR Problem the two entities need to share their utility functions. However, in the highly competitive ecosystem of streaming services and content distribution, these functions constitute sensitive information. Withholding such information might prevent the two parties from cooperating. Therefore, there is need for a cooperation mechanism that can assure privacy. Establishing such a mechanism is not trivial since fairness needs to be guaranteed, as in the centralized solution. We remind the reader that the NBS framework requires that both utility functions are taken into account in the same objective (see (3.8)).

We overcome this challenge by applying the *Alternating Direction Method of Multipliers* (ADMM) [112] to solve the problem in a distributed way. The idea behind ADMM is to *split* the problem into two subproblems, where each subproblem contains only one entity's utility function. Then, the cooperative recommendation problem is solved iteratively: each entity solves the subproblem that contains only its utility function and makes local recommendation decisions. Through coordination and after a sufficient number of iterations, the subproblems' solutions coincide. The coordination preserves the entities' private information and is carried out by a cooperation mediator, which is either a trusted third party or the two entities together. In order to define this distributed algorithm, in what follows: 1) we reformulate the CCR Problem into an equivalent problem (DCR Problem) that can be split into two subproblems, 2) based on the theory on ADMM, we propose the distributed DCR algorithm, and 3) we prove that the resulting cooperation gains converge to the ones of the centralized problem.

Instead of the recommendation variables Y, we introduce here the local recommendation variables $\Psi = (\psi_{ui} \in [0,1])$ and $\widetilde{\Psi} = (\widetilde{\psi}_{ui} \in [0,1])$ that are the variables in the CP's and CDN's subproblems respectively. We reformulate the CCR Problem into the following equivalent problem:

DCR: Distributed Cooperative Recommendations.

$$\min_{\Psi,\widetilde{\Psi}} \quad \Bigg[-\log\left(\sum_{u,i}\frac{\alpha_u}{N_u}\psi_{ui}(R_{ui}-\Lambda_{ui}) - U^b\right) - \log\left(\sum_{u,i}\frac{\alpha_u}{N_u}\widetilde{\psi}_{ui}(\Lambda_{ui}-K_{ui}) - \widetilde{U}^b\right) \Bigg] \tag{3.14}$$

$$\text{s.t.} \quad \psi_{ui} = \widetilde{\psi}_{ui}, \forall u \in \mathcal{U}, i \in \mathcal{K}, \tag{3.15}$$

$$\sum_{i\in\mathcal{K}} \psi_{ui} = N_u, \forall u \in \mathcal{U}, \tag{3.16}$$

$$\sum_{i\in\mathcal{K}} \widetilde{\psi}_{ui} = N_u, \forall u \in \mathcal{U}, \tag{3.17}$$

$$\psi_{ui}, \widetilde{\psi}_{ui} \in [0,1], \forall u \in \mathcal{U}, i \in \mathcal{K}, \tag{3.18}$$

where ψ_{ui} and $\widetilde{\psi}_{ui}$ are the local recommendation variables as decided by the CP and the

CDN respectively. The constraints in (3.15) are the *consistency constraints* that require all local recommendation variables to agree.

The augmented Lagrangian for the DCR problem is:

$$\begin{aligned} L_q(\Psi, \widetilde{\Psi}, Z) &= -\log\left(U(\Psi) - U^b\right) - \log\left(\widetilde{U}(\widetilde{\Psi}) - \widetilde{U}\right) \\ &+ \sum_{u,i} z_{ui}(\psi_{ui} - \widetilde{\psi}_{ui}) + \frac{q}{2}||\Psi - \widetilde{\Psi}||_F^2, \end{aligned} \tag{3.19}$$

where $Z = (z_{ui})$ are the dual variables, q is the penalty parameter, and $||\cdot||_F$ the Frobenius norm. Moreover, the dual function is

$$d(Z) = \inf_{\substack{(\Psi,\widetilde{\Psi}) \\ \text{s.t. (3.16)-(3.18)}}} L_q(\Psi, \widetilde{\Psi}, Z) \tag{3.20}$$

The ADMM for the DCR Problem is described below (see also Fig. 3.4):

The DCR algorithm. The CP communicates α_u, N_u , Y^b and the value of U^b. The CDN communicates ρ and the value of $\widetilde{U}^b$. Then, at every iteration $k+1$:

- The CP solves its subproblem and communicates its local solution:

$$\Psi^{(k+1)} := \operatorname*{argmin}_{\substack{\Psi \\ \text{s.t. (3.16), (3.18)}}} L_q\left(\Psi, \widetilde{\Psi}^{(k)}, Z^{(k)}\right). \tag{3.21}$$

- The CDN solves its subproblem and communicates its local solution:

$$\widetilde{\Psi}^{(k+1)} := \operatorname*{argmin}_{\substack{\widetilde{\Psi} \\ \text{s.t. (3.17),(3.18)}}} L_q\left(\Psi^{(k+1)}, \widetilde{\Psi}, Z^{(k)}\right). \tag{3.22}$$

- The cooperation mediator updates and communicates the dual variables:

$$Z^{(k+1)} := Z^{(k)} + q\left(\Psi^{(k+1)} - \widetilde{\Psi}^{(k+1)}\right). \tag{3.23}$$

We highlight here that each entity keeps private its utility function from the other entity and the mediator. Concerning the practicalities of the DCR algorithm, its iterations will terminate according to standard residual criteria (see [112]). We note that ADMM tolerates inexact minimization for the subproblems under the condition that the relative errors are summable [113]. Moreover, when the subproblems are solved in an iterative way, the warm-start technique can speed up the process. The following lemma guarantees that the DCR algorithm converges (after a sufficient number of iterations) to the centralized objective function value and solution.

Lemma 3.2. *If p^* is the optimal value of the CCR Problem, and $DO^{(k)}$ is the DCR Problem's objective function value at iteration k, i.e., $DO^{(k)} = -\log\left(U(\Psi^{(k)}) - U^b\right) - \log\left(\widetilde{U}(\widetilde{\Psi}^{(k)}) - \widetilde{U}^b\right)$, then $DO^{(k)} \to p^*$, as $k \to \infty$. Moreover, if Y^* is the solution of the CCR Problem, then $\Psi^{(k)}, \widetilde{\Psi}^{(k)} \to Y^*$, as $k \to \infty$.*

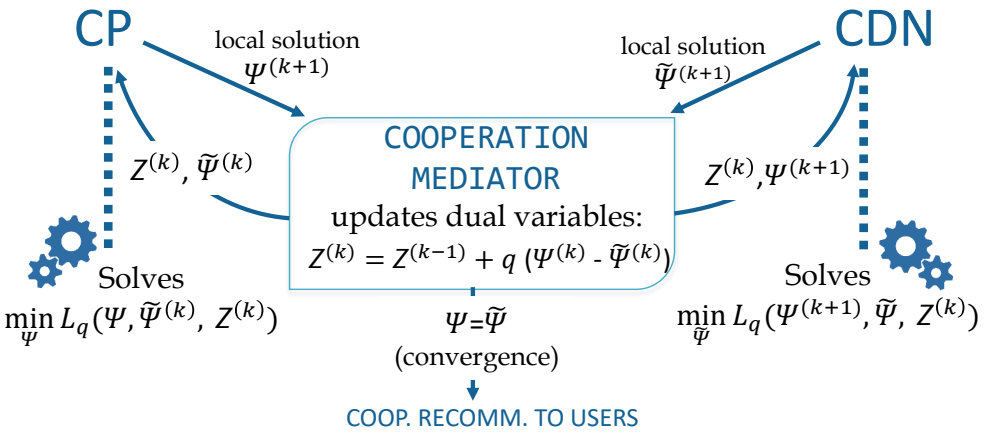


Figure 3.4: Illustration of the DCR algorithm's steps. Each entity solves its subproblem (without sharing its utility function) based on the other's local solution and the dual variables. They communicate their local solutions to the cooperation mediator that updates and communicates the dual variables.

Proof. According to the results in [112], we need to prove two conditions: 1) the extended-real-valued functions $-\log(U(\Psi) - U^b)$ and $-\log(\widetilde{U}(\widetilde{\Psi}) - \widetilde{U}^b)$ are closed, proper, and convex, and 2) the unaugmented Lagrangian L_0 has a saddle point. The two functions are indeed convex (in fact strictly convex) and closed. The corresponding extended-real-valued functions are proper since they are not identically equal to $+\infty$. We will now prove that strong duality holds. When a feasible primal solution $\Psi^* = \widetilde{\Psi}^*$ exists such that $U(\Psi^*) - U^b > 0$ and $\widetilde{U}(\Psi^*) - U^b > 0$, then strong duality holds by Slater's condition (which reduces to feasibility when the problem constraints are linear). Therefore, by feasibility and by strong duality, it follows that the unaugmented Lagrangian L_0 has a saddle point [91].

Since we proved objective convergence, then the local solutions will converge to the centralized solution Y^* since the DCR's objective function is strictly convex. This means that there is at most one global minimizer. □

Essentially, Lemma 3.2 implies that the properties of the centralized solution inherited by the NBS framework (Nash axioms, see Sec. 3.3.1) hold also for the DCR's solution. This is important since it guarantees that the cooperation gains and the fair split of these gains will not be compromised when the two entities apply the DCR algorithm (instead of the CCR).

3.3.3 Ensuring High Quality of Cooperative Recommendations

One might argue that the cooperative recommendations have the potential to degrade user's recommendations. Note that user's interest in the content is one of the factors that determine user's overall experience in OTT services, as shown in experiments [60]. However, the CP can limit a potential recommendation degradation by adding extra

constraints in the problem. For example, adding in the CCR Problem the constraints

$$\sum_i \frac{y_{ui} r_{ui}}{N_u} \geq T_u, \text{ for every user } u, \tag{3.24}$$

forces the average relevance of the cooperative recommendations to the user u to be at least equal to a threshold $T_u \in (0, 1]$. Adding these constraints does not have an impact on the problem analysis (since they are linear with respect to the variables Y). In the (distributed) DCR Problem, the same constraints (with the local variables ψ_{ui} instead of y_{ui}) can be applied when the CP solves its subproblem (with no need of communicating these constraints to the CDN).

3.4 Extension to Caching Decisions

So far, we have focused our framework on scenarios where the recommendations are the only variables that can be re-designed by the CP and CDN, in the time-scale of interest. A natural question that arises is whether also modifying the caching decisions in parallel, could yield even better profits: recommendations could concern contents that are cached in the cache that is closest to the user while they still bring high revenue to the CP. This is particularly useful in today's and future wireless architectures where caches are small while the CP's catalog is constantly growing. This is also in line with recent works proposing the joint optimization of caching and recommendation decisions, *e.g.*, [67] and the problem presented in Chapter 2. Nevertheless, none of these works either explores the financial aspects of the caching-recommendation interplay, nor is it straightforward how to include these into our problem formulation and solution methodology.

A complete treatment of this topic goes beyond the scope of this thesis, due to the additional complexity it introduces in the solution methodology, and is subject to future work. Nevertheless, we will show here how to include such variables into our model, and provide some preliminary analysis and a heuristic for this extended problem. We complement this analysis with related validation results in Sec. 3.5 that already show the proposed method can further increase the cooperation gains for both parties.

Caching Setup and Variables. In this section, we consider, for simplicity, a scenario where the CDN manages only one small cache whose capacity is $\mathcal{C}_1$. Moreover, there is a root cache C_0 that stores all the contents. We employ the prevalent continuous caching model that is valid either when coded caching is used [28] or when the files can be divided in equally-sized chunks and stored independently [107], [35], [114]. We assume that the contents are equal-sized (divided in chunks). In addition to the variables and input values that were introduced in Sec. 3.2, we define the cooperative caching variables:

Definition 3.3. The cooperative caching variables are denoted by $X = (x_i \in [0, 1], i \in \mathcal{K})$, where x_i is the portion of content i that is stored at the cache. These (together with the recommendation variables y_{ui}) constitute control variables throughout this section.

We optimize proactive caching decisions, which constitute a key element of CDN's operations today, as explained before. Therefore, as is common in related works [28, 114],

we assume that the CDN proactively stores contents in its caches and this allocation stays fixed during the period between two updates/fills and between two CP-CDN cooperation instances.

According to (3.2), the retrieval cost for the CDN is in the single-cache scenario is:

$$K_{ui}(X) = k_{u0} + x_i(k_{u1} - k_{u0}). \tag{3.25}$$

In contrast to the definition of the utility functions in Sec. 3.2, we redefine here the CDN's utility functions in order to include the profit that comes from requests out of recommendations (note that now this term contains the control variables). The CDN's baseline utility (before cooperation) and the utility for cooperation are defined as:

$$V^b = \widetilde{U}^b(X^b) + \sum_{u,i}(1-\alpha_u)p_i\left(\lambda - k_{u0} - x_i^b(k_{u1}-k_{u0})\right) \tag{3.26}$$

$$V = \sum_{u,i}\Big[\frac{\alpha_u}{N_u}y_{ui}\left(\Lambda_{ui} - k_{u0} - x_i(k_{u1}-k_{u0})\right), \\ +(1-\alpha_u)p_i\left(\lambda - k_{u0} - x_i(k_{u1}-k_{u0})\right)\Big]. \tag{3.27}$$

In the second summand of V, the delivery fees are λ since the discount does not apply to requests out of recommendations. We stress here that, for the CP, the corresponding term does not contain any of the control variables and it cancels out in the difference $U - U^b$. We can now formulate the optimization problem that can allow us to devise cooperative recommendation and caching policies in a centralized[7] manner (with information sharing between the two entities).

CCRCache: Centralized Cooperative Recommendations & Caching.

$$\min_{X,Y} \quad \Big[-\log\Big(U(Y) - U^b\Big) - \log\Big(V(X,Y) - V^b\Big)\Big] \tag{3.28}$$

$$\text{s.t.} \quad \sum_{i\in\mathcal{K}} y_{ui} = N_u, \ \forall u \in \mathcal{U}, \tag{3.29}$$

$$\sum_{i\in\mathcal{K}} x_i \le C, \tag{3.30}$$

$$x_i, y_{ui} \in [0,1], \ \forall u \in \mathcal{U}, i \in \mathcal{K}, \tag{3.31}$$

where U^b and V^b are in defined in (3.3) and (3.26). According to (3.26) and (3.27), the CDN's gain in utility is:

$$V(X,Y) - V^b = \sum_{u,i}\Big[\ \frac{\alpha_u}{N_u}\ y_{ui}\big(\Lambda_{ui} - k_{u0} - x_i(k_{u1}-k_{u0})\big) \\ -\ (1-\alpha_u)p_i x_i(k_{u1}-k_{u0})\Big] - \widetilde{U}^b. \tag{3.32}$$

The inequality in (3.30) is the cache capacity constraint and, as expressed in (3.31), the control variables are continuous. Finally, the inequalities $U(Y) - U^b \ge 0$ and $V(Y,X) - V^b \ge 0$ are implicit domain constraints.

[7] We only present the centralized problem here. In fact, the presented framework could be also implemented in a distributed way.

Lemma 3.3. *The CCRCache Problem is bi-convex.*

Proof. The objective function is bi-convex, *i.e.,* convex with respect to Y for every fixed X and convex with respect to X for every fixed Y, since the logarithm is a concave function, the utility function U is linear with respect to Y, and the function V is bilinear in X and Y (since it contains the products $y_{ui}x_i$, see (3.32)). Furthermore, all problem's constraints are linear. □

An approach to tackle a bi-convex optimization problem would be to transform it into an equivalent problem that is instead convex in (X, Y). However, such transformations leading to convex equivalent problems are the exception, rather the rule. Standard transformation "tricks" include replacing the products $y_{ui}x_i$ by new variables or discetizing one of the variables involved in the product [115]. The former option is not possible in our problem (since the variables y_{ui} and x_i appear also outside of this product), and the latter could lead to a problem with a large number of new variables. Another approach includes the GOP (global optimization) algorithm that guarantees convergence to the global optimum [116, 117]. Unfortunately, this algorithm comes at the cost of high complexity that could be prohibitive in real-world systems with vast catalogs and multiple users.

Moving away from "exact" methods that attempt to find global optima, Alternate Convex Search [118], and more recently ADMM methods [112], have been popular heuristics for bi-convex problems. Although there are problem instances whose structure permits such algorithms to (provably) converge to global optima (*e.g.*, the well-known matrix factorization problem), they (at best) guarantee convergence to stationary points. We saw in Sec. 3.3.2 how ADMM can be applied in order to provide a distributed solution. The same method can be applied for bi-convex problems since its core idea consists of splitting the main problem into subproblems. Here, the CCRCache Problem can be broken into a subproblem that contains the recommendation variables and another that contains the caching variables. In order to apply ADMM, we reformulate the CCRCache Problem into an equivalent problem by introducing new variables and adding bilinear constraints:

CCRCache′ Problem.

$$\min_{X,Y,Z} \quad \left[-\log\left(U(Y) - U^b\right) - \log\left(G(X,Y,Z) - V^b\right)\right] \tag{3.33}$$

$$\text{s.t.} \quad (3.29), (3.30),$$

$$z_{ui} = x_i y_{ui}, \ \forall u \in \mathcal{U}, i \in \mathcal{K}, \tag{3.34}$$

$$x_i, y_{ui}, z_{ui} \in [0,1], \tag{3.35}$$

where the $Z = (z_{ui} \in [0,1])$ are auxiliary variables that replace the products $x_i y_{ui}$ and $G(X, Y, Z)$ is defined as follows:

$$G(X,Y,Z) = \sum_{u,i} \Big[\quad \tfrac{\alpha_u}{N_u} \quad (y_{ui}(\Lambda_{ui} - k_{u0}) - z_{ui}(k_{u1} - k_{u0}))$$

$$- \quad (1-\alpha_u)p_i x_i (k_{u1} - k_{u0})\Big]. \tag{3.36}$$

It is important to note that the objective of the CCRCache′ Problem is convex in (X, Y, Z) while the bi-linear constraints in (3.34) couple all variables together. We describe below how ADMM [112, Sec. 9.2] can be applied in the CCRCache′ Problem. Even though ADMM for bi-convex problems has no guarantee of convergence, it is expected to have better convergence properties (faster convergence to a local or global optimum or better objective function value) than other local heuristics [112].

The CCRCache algorithm. The CP and the CDN exchange the following information: α_u, N_u, Y^b, U, U^b, ρ, G, and V^b. Then, the two entities together (or through a mediator) solve iteratively the CCRCache′ problem. At every iteration $k+1$ the following steps take place:

- Solving the (Y, Z)-subproblem:

$$\begin{aligned}\left(Y^{(k+1)}, Z^{(k+1)}\right) = \underset{\text{s.t. } (3.29),(3.35)}{\text{argmin}} \Bigg[&- \log\left(U(Y) - U^b\right) \\ - \log\left(G(X^{(k)}, Y, Z) - V^b\right) &+ \frac{q}{2}\left|\left|Z - \left(diag(X^{(k)})Y\right)^T\right|\right|_F^2\Bigg].\end{aligned} \tag{3.37}$$

- Solving the X-subproblem:

$$\begin{aligned}X^{(k+1)} \quad = \quad \underset{\text{s.t. } (3.30),(3.35)}{\text{argmin}} \Bigg[&- \log\left(G(X, Y^{(k+1)}, Z^{(k+1)}) - V^b\right) \\ + \quad \frac{q}{2}&\left|\left|Z^{(k+1)} - \left(diag(X)Y^{(k+1)}\right)^T\right|\right|_F^2\Bigg].\end{aligned} \tag{3.38}$$

- Updating the dual variables denoted by H:

$$H^{(k+1)} \quad = \quad H^{(k)} + \left(Z^{(k+1)} - \left(diag(X^{(k+1)})Y^{(k+1)}\right)^T\right). \tag{3.39}$$

Both the (Y, Z)- and X-subproblems are convex (in fact strongly convex) and can be solved efficiently through standard interior-point or dual methods. The iterations can terminate according to standard residual criteria, *i.e.,* when the differences $z_{ui} - x_i y_{yi}$ are sufficiently small. As a final remark, we stress here that we do not claim that this is necessarily the best method for this problem, and other techniques could further enhance the method's performance [119]. Our sole goal is to apply a reasonably tested method for such problems, and evaluate if the control over the caching variables can reap additional benefits (see Sec. 3.5).

3.5 Performance Evaluation

In this section, we evaluate numerically the payoffs that can be achieved through the proposed cooperation scheme. We will study two scenarios: Scenario I will focus on the evaluatation of the CCR and DCR algorithms in terms of cooperative gains and their impact on the quality of recommendations, while investigating the role of key problem parameters; Scenario II will focus on exploring the benefits of the CCRCache algorithm. First, we present the default input parameters that, unless otherwise stated, will be used across the simulations.

Catalog and Recommendations: Our scenario consists of 100 users who have access to a catalog of 6000 contents. Every user receives $N_u = 5$ recommendations and the probability of following the recommendations varies in $[0.6, 1)$, as in Netflix, where the average is equal to 0.8 [51]. For the matrix of content relevances r_{ui}, a subset of the Movielens dataset [100] containing 5-star ratings of movies was used. The ratings were mapped in the interval $[0, 1]$ and we performed matrix completion to obtain the missing ratings (as in Chapter 2). Finally, the baseline recommendations (before any cooperation), *i.e.,* the values y^b_{ui}, were set to be the ones maximizing U^b in (3.3).

Caching Topology: We consider a network of 9 caches and a root cache containing all contents. Every user has access to 2 of the caches (chosen randomly) and to the root cache. We assume that the (baseline) caching allocation, *i.e.,* X^b, as decided by the CDN, is based on a popularity distribution over the catalog as observed by every cache in a time period that precedes the cooperation. For this, we set the content popularities observed by cache j to be the normalized content utilities r_{ui} aggregated over the connected users, *i.e.*, $r_{ui}/\sum_{u\in\mathscr{C}_j} r_{ui}$, where $\mathscr{C}_j$ is the set of connected users to the cache j.

Revenues and Costs: The values of R_{ui} (CP's revenue per content), were derived through an equation that depends on the content relevances (see Sec. 3.2.2). Unless otherwise stated, this equation will be: $R_{ui} = 0.15 + 0.09 r_{ui}$ (in \$). Note that this implies that the baseline recommendations Y^b are the ones with the highest relevances r_{ui} per user[8]. Concerning the CDN's retrieval costs, they have been chosen randomly from the range $[0.0005, 0.02]$(\$) for the connected caches, while the cost for the root cache is fixed at \$0.055. Finally, the CDN charges the CP \$0.11 per request (according to [120]) for the delivery.

3.5.1 Scenario I

For the default parameters that were described above, for cache sizes varying (randomly) from $1-4\%$ of the content catalog, and for different values of the discount ρ, we evaluate the proposed cooperation in Fig. 3.5. The first subplot (top) depicts the relative gains in utility for the two entities, *i.e.,* the quantities $100\cdot(U-U^b)/U^b$ and $100\cdot(\widetilde{U}-\widetilde{U}^b)/\widetilde{U}^b$, as given by the CCR algorithm. We observe that, for low discount, the CDN benefits

[8]The equation $R_{ui} = 0.15 + 0.09 r_{ui}$ could, for example, capture an ad-based revenue model where r_{ui} can be interpreted as the user retention rate and, thus, the quantity $0.09 r_{ui}$ is the portion of ad-based revenue. It is worth noting that, in [77], we obtained similar performance results when R_{ui} is a concave function of r_{ui}.

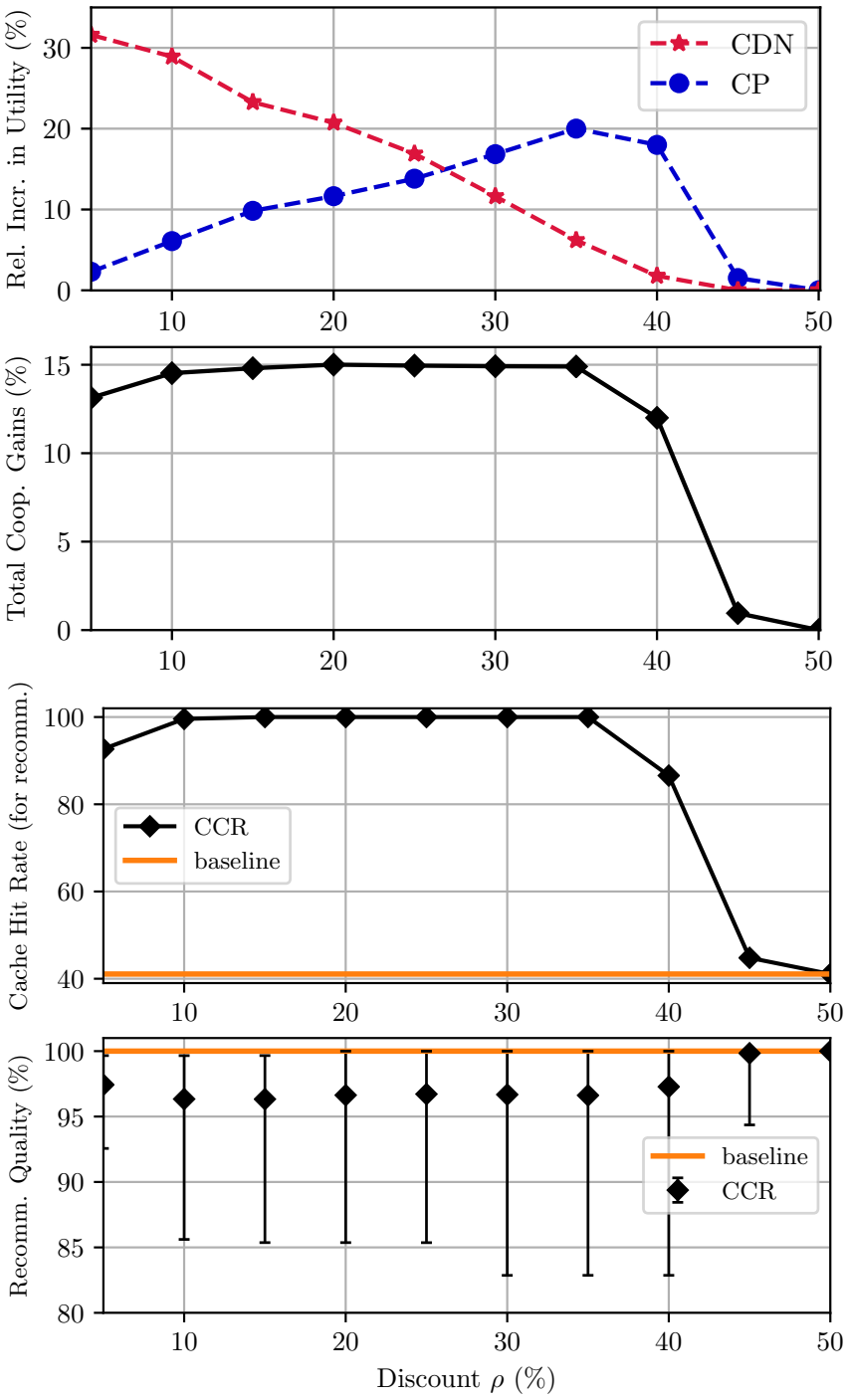


Figure 3.5: Scenario I: Relative increase in utility for the CP and the CDN, total cooperative gains, cache hit rate for recommendations, and quality of recommendations achieved through the proposed cooperation scheme (CCR algorithm) for different values of discount $\rho \in [0.05, 0.5]$.

from the cooperation more than the CP. This is because the CDN saves on routing costs without its revenue from the delivery fees decreasing significantly. On the other hand, we see that the CP benefits the most for high discount as its savings on the delivery fees become important. However, for very high discount, close to 50%, the cooperation becomes unprofitable for the CDN and, therefore, the cooperation would cease, and the recommendations would revert back to the baseline ones. It is important to highlight here that these points are Pareto optimal points. As explained in Sec. 3.3, this means there is no other solution that is better than the solution for one entity and not worse than the solution for the other entity. In the second subplot (of Fig. 3.5), we plot the total relative gains achieved from the cooperation, *i.e.,* the quantity $(U - U^b + \widetilde{U} - \widetilde{U}^b)/(U^b + \widetilde{U}^b)$.

Observation 3.1. The proposed cooperation can lead to significant gains, up to 32% for the CDN and up to 20% for the CP in our scenario. The total cooperative gains can reach up to 15% when compared to the total baseline utilities.

It is worth noting that even gains of 3% or 6% (*i.e.,* CP's gains for $\rho = 5\%$ and 10% respectively) already correspond to very large absolute monetary sums saved (if one extrapolates to a much larger pool of users and requests, as in practice). Especially when referring to large CPs, like Netflix, that report annual profits of more than 2 billion US dollars [121].

Even though each pair of points in the top subplot corresponding to a value of ρ is Pareto optimal, we see that ρ affects the gains of each entity. Obviously, the CDN would rather offer only a small discount, while the CP would prefer the largest discount possible. One could argue that the "best" ρ is between 25% and 30%, *i.e.,* where the two lines meet, since it does not give advantage to any entity. Defining what is the "best" ρ and devising a method to find it is an interesting direction for future work. For example, one could model it as a game with alternating offers, or simply determine ρ through exhaustive search from this plot. Besides, this plot reveals the effect of possible regulatory interventions that, *e.g.,* could set bounds on such discounts in order to foster new business models, protect users' interests and so on.

In the third subplot (of Fig. 3.5), we depict the cache hit rate for the small caches generated by the cooperative recommendations, *i.e.,* the quantity $\sum_{u,i} \sum_{j \in \mathcal{C}(u) \setminus C_0} \alpha_u / N_u y_{ui} x_{ij}$, where $\mathcal{C}(u) \setminus C_0$ is the set of small caches that user u is connected to. Note that α_u / N_u is the probability the user will click on a specific recommendation. We also plot the cache hit rate of the baseline recommendations Y^b. We see that, before cooperation, only 42% of the recommendations were generating a cache hit at the CDN's caches while this percentage can go up to 100% for the cooperative recommendations. We remind the reader that, in our scenario, every user is connected to two small caches, and, therefore, we count a cache hit when the content in question is cached in at least one of the two caches. In fact, the cache hit rate could be smaller in scenarios where every user is connected to a single cache, or where the baseline caching allocation contains less popular/relevant contents. More importantly, even if the CDN can serve from its small caches a big portion of the requests that come from recommendations, there is still room for improvement: these cache hits are not necessarily at the caches closest to each user. We will elaborate on that in Sec. 3.5.2 (Scenario II). We stress here that high cache hit

rate is also beneficial for the user since it implies small start-up delays.

In the forth subplot (of Fig. 3.5), we investigate the impact of cooperative recommendations on the users' perception of the recommender. For that, we measure the recommendations quality (RQ) as defined in 2.2.4 (Def. 2.3). In particular, the RQ for user u measures the sum of relevance of the received recommendations: $\sum_i r_{ui} y_{ui}$ (for simplicity, we consider φ in Def. 2.3 to be the identity function). The forth subplot shows the aggregate RQ (summed over the users) achieved by the cooperative and the baseline recommendations. The y-axis is regularized with respect to the highest existing relevances. The errorbars show the minimum and maximum RQ observed for individual users for every instance.

Observation 3.2. As the cooperative recommendations favor cached items and significantly increase the cache hit rate, the users' aggregate RQ is barely compromised ($\geq 96\%$) in our scenario. The user's RQ is at least 83%, where 100% stands for the most relevant recommendations and the baseline here.

Next, we perform a sensitivity analysis with respect to two key problem parameters: the capacity of CDN's caches and the CP's revenues R_{ui}. For the default simulation parameters, Fig. 3.6 depicts the relative increases in utility, as obtained by the CCR algorithm, for different relative cache sizes and different values of discount ρ.

Observation 3.3. As the relative cache size decreases, we notice the highest utility gains for both the CP and the CDN.

The observation above is particularly promising for today's and future wireless architectures where base stations are equipped with caches of small capacity. As the cache size increases ($10 - 30\%$ of the catalog), the utility gains decrease. Note that when the cache capacity is large, the baseline recommendations (Y^b) are likely to be already cached. Therefore, fewer (when compared to the case of small cache capacity) recommendations need to be adjusted to favor cached items.

Next, we fix the discount at 30% and the cache size at $1 - 4\%$. In Fig. 3.7, we see how the CP's revenue values R_{ui} affect the payoffs of the cooperation. We have plotted the relative increase in utility for both entities for 5 different revenue ranges from $[0.1, 0.2)$ to $[0.1, 1)(\$)$. We observe that, for the range $[0.1, 0.2)$, the CP could have an increase of 22% of its utility. Then, as the range widens, the payoff for the CP decreases. In fact, when the CP's average revenue R_{ui} is much larger than the delivery fee, a reduction on the fee will not have a significant impact on the CP's utility. On the other hand, the CDN's payoff is not affected as much as the range changes since its utility function does not contain the parameters R_{ui}.

Observation 3.4. When the range of R_{ui} is narrow, the CP can enjoy an increase in its utility of 22%, for discount $\rho = 30\%$. As the range widens, the CP would need higher discount in order keep the gains at the same level.

In the remainder of this subsection, we will focus on the proposed distributed algorithm (DCR). For the same problem parameters as in Fig. 3.5 and the discount fixed

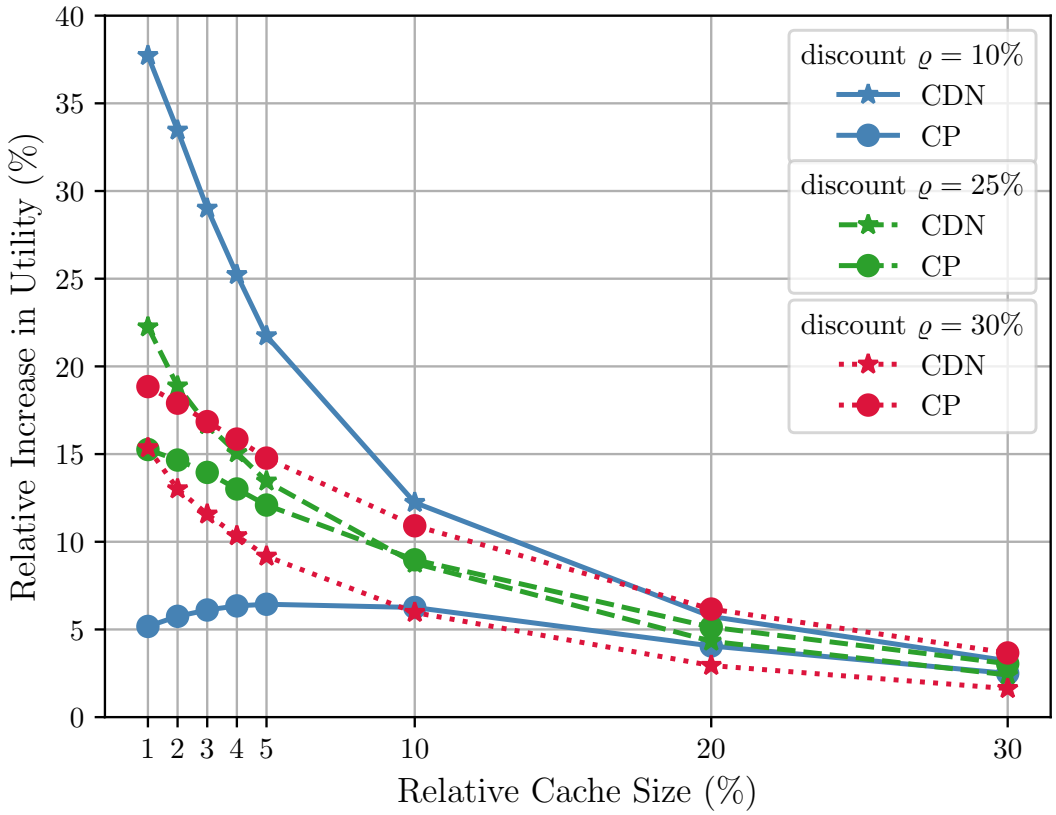


Figure 3.6: Scenario I: Relative increase in utility for different discount values ρ and for different relative cache sizes $(1 - 30\%)$.

at 30%, we will evaluate the convergence of the DCR algorithm and its impact on the cooperation payoffs. The top subplot of Fig. 3.8 depicts the primal residual obtained within 50 iterations for two different values of the penalty parameter q (see eq. (3.23) in Sec. 3.3.2). Note that the primal residual at iteration k is equal to $||\Psi^{(k)} - \widetilde{\Psi}^{(k)}||_F$ and it measures how different the CP's and CDN's local solutions are. In the bottom subplot, we plot the suboptimality gap in percentage, *i.e.*, $|DO^{(k)} - p^*|/|p^*|$, at iteration k, where p^* is the optimal objective function value that is obtained by the CCR algorithm. This gap measures how far the distributed objective value is from the centralized one and, according to Lemma 3.2, tends to zero for a sufficiently large number of iterations. Note that, as p^* is in principle unknown, only the primal and dual residuals are used as stopping criteria.

As we know from the theory on ADMM [112], the higher the penalty parameter is, the lower the primal residuals are. In fact, for $q = 0.01$, we observe a residual's value of less than $4 \cdot 10^{-3}$ and suboptimality gap of 0.14%. On the other hand, when $q = 0.003$, the residual and the suboptimality gap are equal to $6 \cdot 10^{-3}$ and 0.03% respectively. These numbers show a rather fast convergence for the size of our scenario. However, this performance can be further enhanced by applying techniques that, although do not

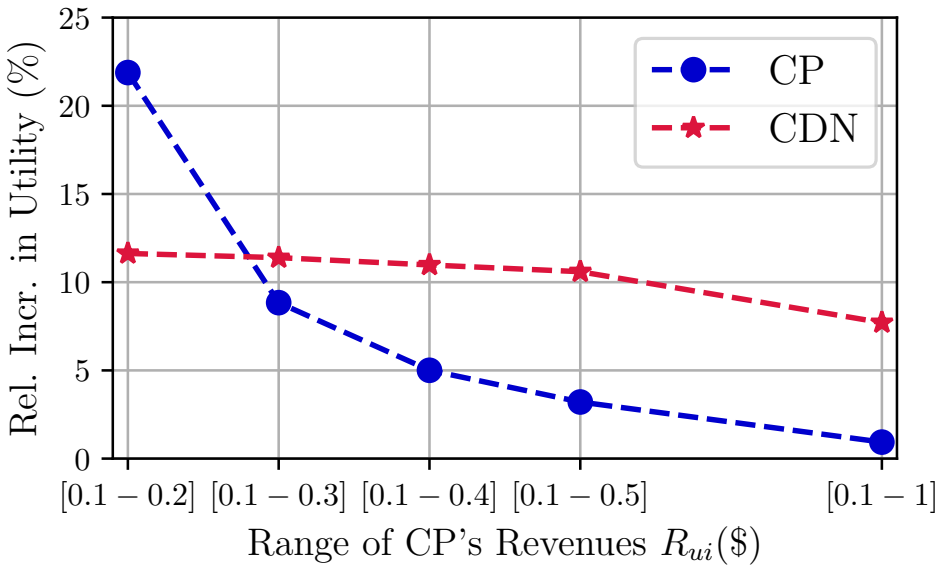


Figure 3.7: Scenario I: Relative increase in utility for different ranges of R_{ui} (CP's revenues per content) as obtained by the CCR algorithm.

guarantee faster convergence, can work well in practice (see [112] for a review on such techniques).

Observation 3.5. Within 20 iterations, the (distributed) DCR algorithm can reach a suboptimality gap of less than 0.1% when compared to the optimal objective function value achieved by the (centralized) CCR algorithm.

Table 3.2: Relative gains obtained by DCR* and CCR

	DCR $k = 2$	DCR $k = 15$	DCR $k = 30$	CCR
CP's gains (%)	17.67	16.79	16.81	16.84
CDN's gains (%)	11.65	11.63	11.63	11.63

*k stands for the number of iterations of the DCR algorithm

Finally, Table 3.2 shows the CP's and CDN's relative gains that result from the DCR algorithm (with $q = 0.003$) for different number of iterations and from the CCR algorithm.

Observation 3.6. Within only a few iterations, the relative increases in utility obtained by the the DCR algorithm approach the Pareto optimal points obtained by the CCR algorithm.

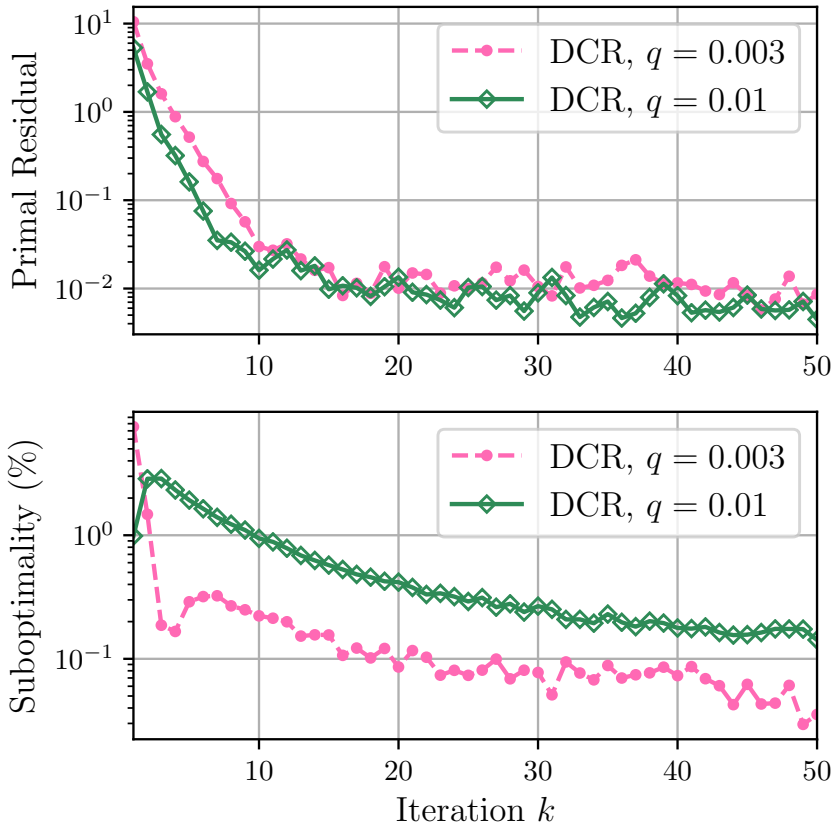


Figure 3.8: Convergence of DCR algorithm in Scenario I: Primal residual, *i.e.,* $||\Psi^{(k)} - \widetilde{\Psi}^{(k)}||_F$, and suboptimality gap, *i.e.,* $|DO^{(k)} - p^*|/|p^*|$, versus the number of iterations for two different values of the penalty parameter q.

3.5.2 Scenario II

As we saw in Fig. 3.5, the cooperative recommendations can lead to a high cache hit rate at the CDN's caches. Although this rate implies significant savings for the CDN, the cache hits do not necessarily happen at the cache that generates the lowest retrieval cost (when delivered to each user). What is more, if the CDN could cache another content, that is potentially more related to the ones in the baseline recommendations, then the CP could further increase its benefits, without actually increasing the cache hit rate, per se. For this reason, we will evaluate now the potential benefits of extending the cooperation towards caching decisions, as we discussed in Sec. 3.4, where we proposed the CCRCache algorithm.

For the default parameters that were described in the beginning of Sec. 3.5, for capacity of caches equal to 5% of the content catalog, and for different values of the discount ρ, we compare the CCR and CCRCache algorithms in Fig. 3.9. More specifically, we apply the CCR algorithm for every problem instance where only the recommendations are the cooperation variables and we apply the CCRCache algorithm for the same instance

where caching is also a cooperation variable. The top subplot depicts the relative gains in utility for the two entities, while the bottom subplot shows the objective function values obtained. We notice that CCRCache leads to larger gains (for at least the CDN) and smaller (better) objective functions values than the ones obtained by the CCR algorithm.

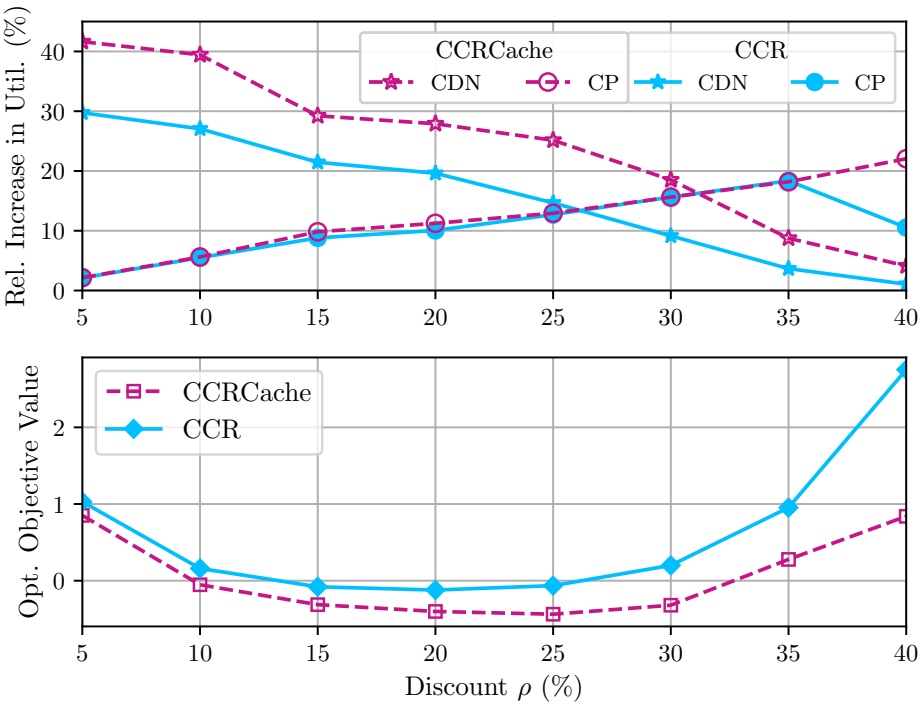


Figure 3.9: Scenario II: Relative gains in utility (for the two entities) and optimal objective function values achieved by the CCRCache and the CCR algorithms for different values of discount $\rho \in [0.05, 0.4]$.

Observation 3.7. When caching becomes a cooperation variable, the CP-CDN cooperation, through the CCRCache algorithm, can boost CDN's utility up to 42%. At the same time, the CP's utility gains are at least as high as when recommendations are the only cooperation variables (through the CCR algorithm).

3.6 Related Work

Several works in literature focus on the cache-friendly recommendations or the joint caching-recommendation paradigm. We refer the reader to Sec. 1.4.2 for an overview of the related work on this topic. We note here that, since most of the related work assumes that the same entity decides on caching and recommendations, these works do not explore the financial aspects of the recommendations from the point of view of both the CP and the CDN. More importantly, none of the existing algorithms guarantee a fair split of the financial gains that come from cache-friendly recommendations.

The theoretical framework of the NBS that we employ in this work was introduced by John Nash in 1950 in [109]. The NBS is a cooperation mechanism that has been employed, among others, in problems of spectrum access coordination [122], bandwidth allocation [123], and content caching [124]. More specifically, in [124], caches that belong to a network collaborate with each other in order to decide on the caching allocation. Moreover, in [125], the authors model a CDN-ISP collaboration as an NBS problem.

Game theory has also been employed by works that study the dynamics between CPs and edge caching providers and propose cooperations or coalitions. For example, [126] and [127] model a coalitional game between a last-mile ISP and CPs. The authors in [128] suggest that the caching network providers should give incentives to the CP in a form of a subsidy (that is paid in proportion to the savings that come from caching). Nevertheless, these works focus on the caching allocation or deployment, without exploiting the impact of recommendations on content requests.

3.7 Conclusion

In this chapter, we proposed a novel cooperation framework in which the CP and the CDN jointly decide on the recommendations in order to favor cached contents. The optimization problem of the cooperation was formulated in such a way that the cooperative recommendations lead to a fair and efficient allocation of financial gains between the two entities. We also developed a distributed algorithm when the two entities are not willing to share private information on their revenue/cost functions. Furthermore, we explored how this cooperation framework could be extended towards the CDN's caching decisions. Although this problem is harder to solve, it has the potential to further increase the cooperation gains. Our numerical evaluations show that, in realistic scenarios, the two entities can benefit of an increase in their expected net revenue of up to 37% and up to 42% when caching is a cooperation variable.

Chapter 4

Conclusions and Perspectives

Overall Conclusions

In this thesis, we studied the problem of interweaving caching and recommendation decisions in the setting of on-demand video streaming services. We approached the problem through two main perspectives. First, we presented the problem of jointly designing caching and recommendations with the objective of maximizing user satisfaction. Then, we studied the problem of cache-friendly recommendations (and extended it to the caching decisions as well) with the goal of maximizing the CP's and CDN's profits while fairly sharing the incurred financial surplus.

Our first notable contribution was the user-centric modeling of the joint problem of caching and recommendations, for which we provided an approximation algorithm. Our numerical evaluations showed that this algorithm not only approximates the optimal policy, but also outperforms policies that have been proposed in the literature. We also showed that distributed/multi-processor implementations of the proposed policy can lead to significant speedups in execution time.

Our second main contribution was the framework for a novel cooperation between the CP and the CDN on the basis of recommendations. Our game-theoretical approach together with elements from convex optimization theory allowed us to construct a cooperation scheme that leads to a fair and efficient allocation of financial gains between the two entities. Furthermore, we explored how this cooperation framework could be extended towards the CDN's caching decisions. Our numerical evaluations showed that the proposed cooperation can lead to significant gains, up to 32% for the CDN, and up to 20% for the CP.

We believe that the contributions of this thesis could serve as a stepping stone towards efficient caching and recommendation policies that are beneficial for everyone. Such policies could potentially change the landscape of streaming services for the best. Of course, further improvements regarding either our framework or alternative research approaches could be made. In what follows, we provide some ideas in this direction.

Possible Directions for Future Work

Throughout Chapters 2 and 3, we have discussed some directions for future work. In addition to that, we now provide 4 main axes for future work that we find challenging in terms of methodology, but particularly promising.

1. **Request Routing in the Joint Problem.** An interesting direction would be to incorporate request routing decision variables in the joint problem studied in Chapter 2. This direction could address possible transmission or bandwidth limitations for the caches, especially where they handle large volumes of Internet traffic. We note that the joint caching and routing problem has been already addressed in several works in the literature, *e.g.,* [43, 44], but none of these take into account the impact of recommendations on the content requests. In the joint routing, caching, and recommendations problem, one would be tempted to apply the decomposition methodology we employed in Chapter 2, and possibly extract submodularity properties. However, such an approach might not be fruitful since the authors in [44] have shown that the joint routing and caching problem is not submodular for a generic network of caches. Therefore, additional optimization methods might be needed.

2. **Dynamic Policies.** In this thesis, we studied proactive caching and recommendation decisions. This is in line with the trend towards proactive caching in today's CDNs, but also in future wireless networks. However, as we mentioned in Sec. 1.2.2, a large number of today's CDNs employ dynamic policies. Therefore, an interesting research thread would be to design dynamic policies that take into account the interplay of caching and recommendations. For example, we could design a dynamic caching policy that, upon an eviction, calculates the utility of a content as a function of the recommendations that will appear on the user's interface. Furthermore, we could adjust, on-the-fly, the recommendations as a function of what is cached at the moment where each user lands on the streaming service. This might raise computational challenges since every eviction at the cache could potentially affect all the users connected to the cache, and thus, a recalibration of recommendations might be needed.

3. **Caching over the Recommendations Graph.** In the context of sequential content requests, an interesting question is about the impact of the recommendations graph on the caching policies. This can be also extended to caching/prefetching contents at the users' devices, especially in the context of device-to-device communications in future wireless architectures. For example, in [129] and [130], the authors present a game between a user/surfer who moves over the graph of hyperlinks on the Internet and a Web browser that tries to prefetch the links before the user requests them. Extending this approach to the recommendations graphs where the users navigate could result in efficient prefetching policies that can potentially improve user experience.

4. **Extensions of the Cooperation Model.** The cooperation model presented in Chapter 3 could be extended in several directions. For example, as we discussed in

Sec. 3.5.1, one could add the discount parameter ρ as a control variable in the cooperation problem. Another direction would be to model a cooperation where more than two entities are bargaining. In particular, as we mentioned in Sec. 1.2.1, a lot of CPs (that do not own their proper CDN) employ more than one CDN for the delivery of contents to their users. We could, for example, address the problem of a bargaining game between a CP and multiple CDNs. This problem is likely to require a different modeling approach (and different tools from cooperative game theory) than our proposed scheme.

Bibliography

[1] E. Keslassy, "Netflix's Cindy Holland says subscribers watch an average of two hours a day," *Variery*, 2019. [Online]. Available: https://variety.com/2019/tv/news/netflix-cindy-holland-subscribers-watch-average-two-hours-day-1203159868/

[2] Cisco, "Cisco annual internet report (2018–2023) white paper," *Cisco: San Jose, CA, USA*, 2020.

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

[4] Sandvine. (2022) The global internet phenomena report (january 2022). [Online]. Available: https://www.sandvine.com/hubfs/Sandvine_Redesign_2019/Downloads/2022/Phenomena%20Reports/GIPR%202022/Sandvine%20GIPR%20January%202022.pdf

[5] H. Nam, K.-H. Kim, and H. Schulzrinne, "QoE matters more than QoS: Why people stop watching cat videos," in *Proc. IEEE INFOCOM*, 2016, pp. 1–9.

[6] N. Fildes, "New media brands thrive on user experience," *Financial Times*, 2019. [Online]. Available: https://www.ft.com/content/33748d9c-78ac-11e9-b0ec-7dff87b9a4a2

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

[8] M. Varela, L. Skorin-Kapov, and T. Ebrahimi, "Quality of service versus quality of experience," in *Quality of experience*. Springer, 2014, pp. 85–96.

[9] L. Skorin-Kapov and M. Varela, "A multi-dimensional view of QoE: the ARCU model," in *Proc. of MIPRO*. IEEE, 2012, pp. 662–666.

[10] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman, "BOLA: Near-optimal bitrate adaptation for online videos," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1698–1711, 2020.

[11] 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, 2019.

[12] Nielsen. (2020) Playback time: Which consumer attitudes will shape the streaming wars? [Online]. Available: https://www.nielsen.com/us/en/insights/article/2020/playback-time-which-consumer-attitudes-will-shape-the-streaming-wars/

[13] Netflix Help Center. (2020) How can I control how much data Netflix uses? (France). [Online]. Available: https://help.netflix.com/en/node/87/fr

[14] Spotify. (2022) Audio Quality. [Online]. Available: https://support.spotify.com/us/article/audio-quality/

[15] M. Hofmann and L. R. Beaumont, *Content networking: architecture, protocols, and practice*. Elsevier, 2005.

[16] Akamai. (2022) Facts & figures. [Online]. Available: https://www.akamai.com/company/facts-figures.jsp

[17] B. M. Maggs and R. K. Sitaraman, "Algorithmic nuggets in content delivery," *ACM SIGCOMM Computer Communication Review*, vol. 45, no. 3, pp. 52–66, 2015.

[18] Microsoft Azure. (2020) Content Delivery Network Pricing. [Online]. Available: https://azure.microsoft.com/en-us/pricing/details/cdn/

[19] Nokia blog. (2020) Network traffic insights in the time of COVID-19: March 23-29 update. [Online]. Available: https://www.nokia.com/blog/network-traffic-insights-time-covid-19-march-23-29-update/

[20] Netflix. Open Connect Overview. [Online]. Available: https://openconnect.netflix.com/Open-Connect-Overview.pdf

[21] Google Interconnect Help. (2022) Introduction to GGC. [Online]. Available: https://support.google.com/interconnect/answer/9058809?hl=en#

[22] Meta. (2022) The evolution of advanced caching in the facebook CDN. [Online]. Available: https://research.facebook.com/blog/2016/04/the-evolution-of-advanced-caching-in-the-facebook-cdn/

[23] Netflix Open Connect. (2022) Peering with open connect. [Online]. Available: https://openconnect.netflix.com/en/peering/

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

[25] Google Interconnect Help. (2022) Content served. [Online]. Available: https://support.google.com/interconnect/answer/7658599?hl=en&ref_topic=7659366

[26] Netflix Open Connect. (2022) Network configuration. [Online]. Available: https://openconnect.zendesk.com/hc/en-us/articles/360035533071-Network-configuration

[27] G. S. Paschos, G. Iosifidis, M. Tao, D. Towsley, and G. Caire, "The role of caching in future communication systems and networks," *IEEE J. on Selected Areas in Comm.*, vol. 36, no. 6, pp. 1111–1125, 2018.

[28] K. Shanmugam, N. Golrezaei, A. G. Dimakis, A. F. Molisch, and G. Caire, "Femtocaching: Wireless content delivery through distributed caching helpers," *IEEE Transactions on Information Theory*, vol. 59, no. 12, pp. 8402–8413, 2013.

[29] N. Golrezaei, A. F. Molisch, A. G. Dimakis, and G. Caire, "Femtocaching and device-to-device collaboration: A new architecture for wireless video distribution," *IEEE Communications Magazine*, vol. 51, no. 4, pp. 142–149, 2013.

[30] M. A. Maddah-Ali and U. Niesen, "Coding for caching: fundamental limits and practical challenges," *IEEE Communications Magazine*, vol. 54, no. 8, pp. 23–29, 2016.

[31] ——, "Fundamental limits of caching," *IEEE Transactions on information theory*, vol. 60, no. 5, pp. 2856–2867, 2014.

[32] S. Vassilaras, L. Gkatzikis, N. Liakopoulos, I. N. Stiakogiannakis, M. Qi, L. Shi, L. Liu, M. Debbah, and G. S. Paschos, "The algorithmic aspects of network slicing," *IEEE Communications Magazine*, vol. 55, no. 8, pp. 112–119, 2017.

[33] B. Shen, S.-J. Lee, and S. Basu, "Caching strategies in transcoding-enabled proxy systems for streaming media distribution networks," *IEEE Transactions on Multimedia*, vol. 6, no. 2, pp. 375–386, 2004.

[34] Google Interconnect Help. (2022) Cache-fill. [Online]. Available: https://support.google.com/interconnect/answer/7658593?hl=en

[35] 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, pp. 1–9.

[36] G. S. Paschos, A. Destounis, L. Vigneri, and G. Iosifidis, "Learning to cache with no regrets," in *Proc. IEEE INFOCOM*. IEEE, 2019, pp. 235–243.

[37] 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.

[38] J. Tadrous and A. Eryilmaz, "On optimal proactive caching for mobile networks with demand uncertainties," *IEEE/ACM Trans. on Networking*, vol. 24, no. 5, pp. 2715–2727, 2015.

[39] Netflix Tech Blog. (2016) Netflix and Fill. [Online]. Available: https://netflixtechblog.com/netflix-and-fill-c43a32b490c0

[40] ——. (2017) Content popularity for Open Connect. [Online]. Available: https://netflixtechblog.com/content-popularity-for-open-connect-b86d56f613b

[41] 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.

[42] C. Li, L. Toni, J. Zou, H. Xiong, and P. Frossard, "QoE-driven mobile edge caching placement for adaptive video streaming," *IEEE Transactions on Multimedia*, vol. 20, no. 4, pp. 965–984, 2017.

[43] M. Dehghan, B. Jiang, A. Seetharam, T. He, T. Salonidis, J. Kurose, D. Towsley, and R. Sitaraman, "On the complexity of optimal request routing and content caching in heterogeneous cache networks," *IEEE/ACM Trans. on Networking*, vol. 25, no. 3, pp. 1635–1648, 2016.

[44] B. Liu, K. Poularakis, L. Tassiulas, and T. Jiang, "Joint caching and routing in congestible networks of arbitrary topology," *IEEE Internet of Things Journal*, vol. 6, no. 6, pp. 10 105–10 118, 2019.

[45] E. Gourdin, P. Maillé, G. Simon, and B. Tuffin, "The economics of CDNs and their impact on service fairness," *IEEE Transactions on Network and Service Management*, vol. 14, no. 1, pp. 22–33, 2017.

[46] S. Elayoubi and J. Roberts, "Performance and cost effectiveness of caching in mobile access networks," in *Proceedings of International Conference on Information-Centric Networking, ICN '15, San Francisco, USA,*, 2015.

[47] J. Kwak, G. Paschos, and G. Iosifidis, "Elastic femtocaching: Scale, cache, and route," *IEEE Transactions on Wireless Communications*, vol. 20, no. 7, pp. 4174–4189, 2021.

[48] J. Krolikowski, A. Giovanidis, and M. Di Renzo, "A decomposition framework for optimal edge-cache leasing," *IEEE J. on Selected Areas in Communications*, vol. 36, no. 6, pp. 1345–1359, 2018.

[49] N. Ramachandran, "Why Netflix is experimenting with linear programming in France," *Variery*, 2019. [Online]. Available: https://variety.com/2020/digital/global/netflix-france-linear-direct-1234831123/

[50] S. Laurent, "Netflix vs. decision fatigue: How to solve the paradox of choice," *UX Collective blog*, 2019. [Online]. Available: https://uxdesign.cc/netflix-vs-decision-fatigue-how-to-solve-the-paradox-of-choice-888ca56db4b

[51] 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.

[52] R. Zhou, S. Khemmarat, and L. Gao, "The impact of YouTube recommendation system on video views," in *Proceedings of the ACM SIGCOMM Conference on Internet measurement*. ACM, 2010, pp. 404–410.

[53] F. Ricci, L. Rokach, and B. Shapira, "Introduction to recommender systems handbook," in *Recommender systems handbook*. Springer, 2011, pp. 1–35.

[54] P. Covington, J. Adams, and E. Sargin, "Deep neural networks for youtube recommendations," in *Proceedings of the 10th ACM Conference on Recommender Systems*. ACM, 2016, pp. 191–198.

[55] G. Adomavicius and Y. Kwon, "Improving aggregate recommendation diversity using ranking-based techniques," *IEEE Transactions on Knowledge and Data Engineering*, vol. 24, no. 5, pp. 896–911, 2011.

[56] X. Amatriain and J. Basilico, "Past, present, and future of recommender systems: An industry perspective," in *Proc. ACM RecSys*, 2016, pp. 211–214.

[57] H. Ma, D. Zhou, C. Liu, M. R. Lyu, and I. King, "Recommender systems with social regularization," in *Proc. ACM Web search and data mining*, 2011, pp. 287–296.

[58] Y. Zhang, Q. Zhao, Y. Zhang, D. Friedman, M. Zhang, Y. Liu, and S. Ma, "Economic recommendation with surplus maximization," in *Proc. of the 25th International Conf. on World Wide Web*, 2016, pp. 73–83.

[59] Netflix. (2021) Downloads for you takes on-the-go to the next level. [Online]. Available: https://about.netflix.com/en/news/downloads-for-you-takes-on-the-go-to-the-next-level

[60] W. Li, P. Spachos, M. Chignell, A. Leon-Garcia, L. Zucherman, and J. Jiang, "Impact of technical and content quality on overall experience of OTT video," in *IEEE Annual Consumer Comm. & Networking Conf. (CCNC)*. IEEE, 2016, pp. 930–935.

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

[62] J. Tadrous, A. Eryilmaz, and H. El Gamal, "Proactive content download and user demand shaping for data networks," *IEEE/ACM Transactions on Networking*, vol. 23, no. 6, pp. 1917–1930, 2014.

[63] D. K. Krishnappa, M. Zink, C. Griwodz, and P. Halvorsen, "Cache-centric video recommendation: an approach to improve the efficiency of YouTube caches," *ACM Trans. on Multimedia Comput., Comm., and Applications (TOMM)*, vol. 11, no. 4, p. 48, 2015.

[64] S. Kastanakis, P. Sermpezis, V. Kotronis, D. S. Menasche, and T. Spyropoulos, "Network-aware recommendations in the wild: Methodology, realistic evaluations, experiments," *IEEE Trans. Mob. Comput.*, 2020.

[65] T. Giannakas, P. Sermpezis, and T. Spyropoulos, "Show me the cache: Optimizing cache-friendly recommendations for sequential content access," in *Proc. IEEE WoWMoM 2018*, 2018, pp. 14–22.

[66] T. Giannakas, A. Giovanidis, and T. Spyropoulos, "SOBA: Session optimal mdp-based network friendly recommendations," in *Proc. IEEE INFOCOM 2021*, 2021.

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

[68] K. Qi, B. Chen, C. Yang, and S. Han, "Optimizing caching and recommendation towards user satisfaction," in *Proc. IEEE WCSP*, 2018, pp. 1–7.

[69] M.-C. Lee and Y.-W. P. Hong, "Socially-aware joint recommendation and caching policy design in wireless d2d networks," in *ICC 2021-IEEE International Conference on Communications*. IEEE, 2021, pp. 1–6.

[70] Z. Lin and W. Chen, "Joint pushing and recommendation for susceptible users with time-varying connectivity," in *Proc. IEEE GLOBECOM*, 2018, pp. 1–6.

[71] D. Liu and C. Yang, "A learning-based approach to joint content caching and recommendation at base stations," in *Proc. IEEE GLOBECOM*, 2018, pp. 1–7.

[72] ——, "A deep reinforcement learning approach to proactive content pushing and recommendation for mobile users," *IEEE Access*, vol. 7, pp. 83 120–83 136, 2019.

[73] P. Sermpezis, T. Giannakas, T. Spyropoulos, and L. Vigneri, "Soft cache hits: Improving performance through recommendation and delivery of related content," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 6, pp. 1300–1313, 2018.

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

[75] D. Tsigkari and T. Spyropoulos, "User-centric optimization of caching and recommendations in edge cache networks," in *Proc. IEEE WoWMoM*, 2020, pp. 244–253.

[76] D. Tsigkari and T. Spyropoulos, "An approximation algorithm for joint caching and recommendations in cache networks," *IEEE Transactions on Network and Service Management*, 2022.

[77] D. Tsigkari, G. Iosifidis, and T. Spyropoulos, "Split the cash from cache-friendly recommendations," in *Proc. IEEE GLOBECOM*, 2021, pp. 1–6.

[78] ——, "Quid pro quo in streaming services: Algorithms for cooperative recommendations," *Submitted and under review*, 2022.

[79] G. S. Paschos, E. Bastug, I. Land, G. Caire, and M. Debbah, "Wireless caching: Technical misconceptions and business barriers," *IEEE Communications Magazine*, vol. 54, no. 8, pp. 16–22, 2016.

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

[81] S. Borst, V. Gupta, and A. Walid, "Distributed caching algorithms for content distribution networks," in *Proc. IEEE INFOCOM*, 2010, pp. 1–9.

[82] X. Amatriain, "Building industrial-scale real-world recommender systems," in *Proc. ACM RecSys*, 2012, pp. 7–8.

[83] M. Bressan, S. Leucci, A. Panconesi, P. Raghavan, and E. Terolli, "The limits of popularity-based recommendations, and the role of social ties," in *Proc. ACM SIGKDD*, 2016, p. 745–754.

[84] H. Batteram, G. Damm, A. Mukhopadhyay, L. Philippart, R. Odysseos, and C. Urrutia-Valdés, "Delivering quality of experience in multimedia networks," *Bell Labs Tech. J.*, vol. 15, no. 1, pp. 175–193, 2010.

[85] Google Official Blog. (2009) Personalized search for everyone. [Online]. Available: https://googleblog.blogspot.com/2009/12/personalized-search-for-everyone.html

[86] Google Developers. (2018) Speed is now a landing page factor for google search and ads. [Online]. Available: https://developers.google.com/web/updates/2018/07/search-ads-speed

[87] M. Fiedler, T. Hossfeld, and P. Tran-Gia, "A generic quantitative relationship between quality of experience and quality of service," *IEEE Network*, vol. 24, no. 2, pp. 36–41, 2010.

[88] P. Reichl, S. Egger, R. Schatz, and A. D'Alconzo, "The logarithmic nature of QoE and the role of the Weber-Fechner law in QoE assessment," in *Proc. IEEE ICC*, 2010, pp. 1–5.

[89] S. Dehaene, "The neural basis of the Weber–Fechner law: a logarithmic mental number line," *Trends in cognitive sciences*, vol. 7, no. 4, pp. 145–147, 2003.

[90] S. Boyd, L. Xiao, A. Mutapcic, and J. Mattingley. (2007) Notes on decomposition methods. [Online]. Available: https://see.stanford.edu/materials/lsocoee364b/08-decomposition_notes.pdf

[91] S. Boyd and L. Vandenberghe, *Convex optimization*. Cambridge university press, 2004.

[92] M. L. Fisher, G. L. Nemhauser, and L. A. Wolsey, "An analysis of approximations for maximizing submodular set functions—II," in *Polyhedral combinatorics*. Springer, 1978, pp. 73–87.

[93] 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.

[94] 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.

[95] 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.

[96] 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.

[97] B. Mirzasoleiman, A. Badanidiyuru, A. Karbasi, J. Vondrák, and A. Krause, "Lazier than lazy greedy," in *Proc. AAAI Conf. on Artificial Intelligence*, vol. 29, no. 1, 2015.

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

[99] Flixable. (2019) Netflix Museum. [Online]. Available: https://flixable.com/netflix-museum/

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

[101] S. R. Becker, E. J. Candès, and M. C. Grant, "Templates for convex cone problems with applications to sparse signal recovery," *Mathematical Programming Computation*, vol. 3, no. 3, p. 165, Jul 2011.

[102] A. Abhari and M. Soraya, "Workload generation for YouTube," *Multimedia Tools and Applications*, vol. 46, no. 1, p. 91, 2010.

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

[104] K. Poularakis, G. Iosifidis, and L. Tassiulas, "Approximation algorithms for mobile data caching in small cell networks," *IEEE Transactions on Communications*, vol. 62, no. 10, pp. 3665–3677, 2014.

[105] E. M. Craparo, J. P. How, and E. Modiano, "Throughput optimization in mobile backbone networks," *IEEE Transactions on Mobile Computing*, vol. 10, no. 4, pp. 560–572, Apr. 2011.

[106] J. Davidson, B. Liebald, J. Liu, P. Nandy, T. Van Vleet, U. Gargi, S. Gupta, Y. He, M. Lambert, B. Livingston *et al.*, "The YouTube video recommendation system," in *Proc. ACM RecSys*, 2010.

[107] B. Blaszczyszyn and A. Giovanidis, "Optimal geographic caching in cellular networks," in *Proc. IEEE ICC*, 2015, pp. 3358–3363.

[108] Maz Systems. (2020) OTT business models. [Online]. Available: https://www.mazsystems.com/types-of-ott-business-models/

[109] J. F. Nash Jr, "The bargaining problem," *Econometrica: Journal of the econometric society*, pp. 155–162, 1950.

[110] R. B. Myerson, *Game theory*. Harvard University Press, 2013.

[111] G. Eriksson, J. Mattsson, N. Mitra, and Z. Sarker, "Blind cache: a solution to content delivery challenges in an all-encrypted web," *Ericsson review*, vol. 94, no. 1, pp. 8–19, 2017.

[112] S. Boyd, N. Parikh, and E. Chu, *Distributed optimization and statistical learning via the alternating direction method of multipliers*. Now Publishers Inc, 2011.

[113] J. Eckstein and D. P. Bertsekas, "On the Douglas—Rachford splitting method and the proximal point algorithm for maximal monotone operators," *Mathematical Programming*, vol. 55, no. 1, pp. 293–318, 1992.

[114] G. Paschos, G. Iosifidis, G. Caire *et al.*, "Cache optimization models and algorithms," *Foundations and Trends® in Communications and Information Theory*, vol. 16, no. 3–4, pp. 156–345, 2020.

[115] W. Wei, "Tutorials on advanced optimization methods," *arXiv preprint arXiv:2007.13545*, 2020.

[116] C. A. Floudas and V. Visweswaran, "A global optimization algorithm (GOP) for certain classes of nonconvex NLPs—i. theory," *Computers & chemical engineering*, vol. 14, no. 12, pp. 1397–1417, 1990.

[117] C. A. Floudas, *Deterministic global optimization: theory, methods and applications*. Springer Science & Business Media, 2013, vol. 37.

[118] R. E. Wendell and A. P. Hurter Jr, "Minimization of a non-separable objective function subject to disjoint constraints," *Operations Research*, vol. 24, no. 4, pp. 643–657, 1976.

[119] S. Diamond, R. Takapoui, and S. Boyd, "A general system for heuristic minimization of convex functions over non-convex sets," *Optimization Methods and Software*, vol. 33, no. 1, pp. 165–193, 2018.

[120] Amazon CloudFront. (2020) Pricing. [Online]. Available: https://aws.amazon.com/cloudfront/pricing/

[121] Fortune. (2021) Netflix company profile. [Online]. Available: https://fortune.com/company/netflix/fortune500/

[122] Y. Wu and W.-Z. Song, "Cooperative resource sharing and pricing for proactive dynamic spectrum access via Nash bargaining solution," *IEEE Trans. on Par. and Distr. Sys.*, vol. 25, no. 11, pp. 2804–2817, 2013.

[123] H. Yuan, X. Wei, F. Yang, J. Xiao, and S. Kwong, "Cooperative bargaining game-based multiuser bandwidth allocation for dynamic adaptive streaming over http," *IEEE Trans. on Multimedia*, vol. 20, no. 1, pp. 183–197, 2017.

[124] L. Wang, G. Tyson, J. Kangasharju, and J. Crowcroft, "Milking the cache cow with fairness in mind," *IEEE/ACM Transactions on Networking*, vol. 25, no. 5, pp. 2686–2700, 2017.

[125] W. Jiang, R. Zhang-Shen, J. Rexford, and M. Chiang, "Cooperative content distribution and traffic engineering in an ISP network," in *Proc. Conf. on measurement and modeling of comp. sys.*, 2009, pp. 239–250.

[126] V. G. Douros, S. E. Elayoubi, E. Altman, and Y. Hayel, "Caching games between content providers and internet service providers," *Performance Evaluation*, vol. 113, pp. 13–25, 2017.

[127] D. Mitra and A. Sridhar, "Consortiums of ISP-content providers formed by Nash bargaining for Internet content delivery," in *IEEE INFOCOM*, 2019, pp. 631–639.

[128] M. Ahmadi, J. Roberts, E. Leonardi, and A. Movaghar, "Cache subsidies for an optimal memory for bandwidth tradeoff in the access network," *IEEE Journal on Sel. Areas in Commun.*, vol. 38, no. 4, pp. 736–749, 2020.

[129] F. V. Fomin, F. Giroire, A. Jean-Marie, D. Mazauric, and N. Nisse, "To satisfy impatient web surfers is hard," *Theoretical Computer Science*, vol. 526, pp. 1–17, 2014.

[130] F. Giroire, I. Lamprou, D. Mazauric, N. Nisse, S. Pérennes, and R. Soares, "Connected surveillance game," *Theoretical Computer Science*, vol. 584, pp. 131–143, 2015.