Reinforcement learning and optimization for energy efficient 5G slicing with Quality of Service guarantees

Maxime Elkael

INSTITUT POLYTECHNIQUE DE PARIS

NNT : 2023IPPAS015

Thèse de doctorat

Reinforcement Learning and Optimization for an Energy and Resource Efficient 5G slicing

Thèse de doctorat de l'Institut Polytechnique de Paris
préparée à Télécom SudParis

École doctorale n°626
Spécialité de doctorat : Mathématiques et Informatique

Thèse présentée et soutenue à Evry, le 19 décembre 2023 par

Maxime Elkael

Composition du Jury :

Tristan Cazenave Professeur, Université Paris-Dauphine	Rapporteur
Adlen Ksentini Professeur, Eurecom	Rapporteur
Emmanuel Hyon Maitre de Conférence, Université Paris-Nanterre	Examinateur
Nancy Perrot Cheffe de Projet, Orange Labs	Examinatrice
Stefano Secci Professeur, Centre National des Arts et Métiers	Président
Véronique Vèque Professeure, Université Paris-Saclay	Examinatrice
Hind Castel-Taleb Professeure, Télécom SudParis	Directrice de thèse
Andrea Araldo Maitre de Conférence, Télécom SudParis	Co-Directeur de thèse
Badii Jouaber Professeur, Télécom SudParis	Co-Directeur de t

Résumé

Cette thèse traite des problèmes d'allocation des ressources dans les réseaux 5G, en utilisant le network slicing. Le network slicing est un corpus de techniques basées sur la virtualisation et la softwarisation du réseau qui permettent à l'opérateur de fournir différentes quantités de ressources à différents clients. Notre objectif est d'améliorer l'efficacité énergétique et la consommation de ressources des réseaux 5G, tout en respectant des contraintes de Qualité de Service. Pour ce faire, nous formulons et résolvons des problèmes d'optimisation dans les différents domaines du réseau : nous nous intéressons d'abord au placement dans le réseau coeur. Pour résoudre le problème, une nouvelle approche combinant la recherche Monte Carlo et la recherche par voisinage est formulée. Nous montrons qu'il surpasse les approches de l'état de l'art pour le problème de placement du réseau coeur. Ensuite, nous mettons l'accent sur l'efficacité énergétique en proposant un framework holistique pour l'allocation de ressources efficaces en énergie dans les réseaux 5G partagés entre les opérateurs de réseaux physiques (PNO) et les opérateurs de réseaux mobiles virtuels (MVNO). Ce framework tient compte à la fois du placement des composants logiciels, du routage des demandes des utilisateurs et du dimensionnement des ressources, tout en respectant les accords de niveau de service (SLA) basés sur des contraintes de latence et de fiabilité. Grâce à la génération de colonnes, nous obtenons des solutions exactes, démontrant des économies d'énergie remarquables allant jusqu'à 50% dans des réseaux réels, par rapport aux algorithmes de placement ou de minimisation des ressources existants. Enfin, nous abordons le problème de l'optimisation de l'énergie dans les réseaux Integrated Access and Backhaul (IAB), un élément clé des déploiements denses de la 5G. S'appuyant sur le framework du réseau d'accès ouvert Open RAN (O-RAN), notre modèle minimise les noeuds IAB actifs tout en garantissant une capacité minimale pour l'équipement de l'utilisateur (UE). Formulé comme un programme non linéaire binaire et résolu à l'aide du solveur Gurobi, cette approche réduit la consommation d'énergie du RAN de 47%, tout en maintenant la qualité de service pour les UEs. Dans l'ensemble, cette thèse contribue à faire progresser le slicing et l'optimisation énergétique du réseau 5G, en fournissant de nouveaux algorithmes pour des parties différentes et complémentaires du réseau 5G.

Abstract

This thesis addresses resource allocation problems in 5G networks, leveraging network slicing. Network slicing is the set of techniques based on virtualization and network softwarization which allows the network operator to provide different amounts of resources to different tenants. Our objective is to improve the energy-efficiency and resource consumption of 5G networks, while guaranteeing Quality of Service constraints. To do so, we formulate and solve optimization problems at the different domains of the network: we are first concerned with the placement of slices in the core network. To solve the problem, a new approach combining Monte Carlo Search and Neighborhood Search is formulated. We show it beats state-of-the-art approaches for the core network placement problem. Then, we shift the focus to energy efficiency by proposing a holistic framework for energy-efficient resource allocation in 5G networks shared between Physical Network Operators (PNOs) and Mobile Virtual Network Operators (MVNOs). This framework jointly considers software component placement, user request routing, and resource dimensioning while meeting Service Level Agreements (SLAs) based on latency and reliability constraints. Through Column Generation, we obtain exact solutions, demonstrating remarkable energy savings of up to 50% in real networks compared to existing placement or resource minimization algorithms. Finally, we delve into the realm of energy optimization in Integrated Access and Backhaul (IAB) networks, a key component of dense 5G deployments. Leveraging the Open Radio Access Network (O-RAN) framework, our model minimizes active IAB nodes while ensuring a minimum capacity for User Equipment (UE). Formulated as a binary nonlinear program and solved using the Gurobi solver, this approach reduces RAN energy consumption by 47%, while maintaining Quality-Of-Service for UEs. Overall, this thesis contributes to advancing 5G network slicing, virtualization, and energy optimization, providing novel algorithms for different and complementary parts of the 5G network.

Contents

1 Introduction **13**
- 1.1 Motivation . . . 13
- 1.2 Contributions . . . 15
  - 1.2.1 Virtual network embedding for Core Network Slice Placement . . . 15
  - 1.2.2 Edge slice placement . . . 15
  - 1.2.3 Integrated Access and Backhaul . . . 16
- 1.3 Thesis organization . . . 17

2 Background **19**
- 2.1 Introduction . . . 19
- 2.2 Infrastructure of the Network . . . 19
  - 2.2.1 Core Network . . . 20
  - 2.2.2 Radio Access Network . . . 23
  - 2.2.3 Transport Network . . . 25
- 2.3 New Service and Networking paradigms . . . 26
  - 2.3.1 Mobile Edge Computing . . . 26
  - 2.3.2 Network Slicing . . . 28
  - 2.3.3 Integrated Access and Backhaul . . . 29
- 2.4 TopologyZoo Dataset . . . 30
- 2.5 Conclusion . . . 31

3 Core Network Slice Placement: Virtual Network Embedding Problem **33**
- 3.1 Introduction . . . 33
  - 3.1.1 Summary of chapter contributions . . . 34
- 3.2 Pre-existing methods for VNE . . . 35
  - 3.2.1 Mathematical programming . . . 36
  - 3.2.2 Graph neural networks . . . 36
  - 3.2.3 Heuristics and meta-heuristic techniques . . . 36
  - 3.2.4 Reinforcement learning approaches . . . 37
- 3.3 VNE model . . . 38
  - 3.3.1 Graph theoretic notation . . . 39

3.3.2 Problem constraints . . . 40
3.3.3 Online VNE description . . . 40
3.4 VNE as a Markov-Decision Process (MDP) . . . 41
3.4.1 MDP description . . . 41
3.4.1.1 Elements of the MDP . . . 42
3.4.2 System's evolution . . . 43
3.4.2.1 Reward Function . . . 43
3.4.2.2 Objective function . . . 44
3.4.3 Characteristics of the MDP and implications on resolution . . . 45
3.5 Contributions for online VNE . . . 46
3.5.1 NRPA and NEPA . . . 46
3.5.1.1 Review of Nested Rollout Policy Adaptation (NRPA) . . . 46
3.5.1.2 Virtual links placement . . . 49
3.5.1.3 Heuristic weight initialization . . . 50
3.5.1.4 Neighborhood Enhanced Policy Adaptation (NEPA) . . . 54
3.5.1.5 Main steps of NEPA . . . 55
3.5.1.6 Theoretical analysis . . . 57
3.5.1.7 Dimensionality reduction and pre-treatment 60
3.5.2 Numerical Results . . . 61
3.5.2.1 Compared methods . . . 61
3.5.2.2 Results on synthetic physical topologies . . 63
3.5.2.3 Real Topologies . . . 65
3.5.2.4 Specific case: Perfectly solvable scenarios . . 69
3.6 Conclusion and remarks . . . 71

4 Edge Network Slice Placement and Dimensionning 75
4.1 Introduction . . . 75
4.2 Background . . . 78
4.2.1 Other pre-existing methods and models for edge placement . . . 78
4.2.2 Background on used method: column generation . . . 79
4.2.2.1 Introduction . . . 79
4.2.2.2 Solving Linear Programs with the Simplex algorithm . . . 80
4.2.2.3 Solving Linear Programs (LPs) with CG . . 82
4.2.2.4 Branch-and-Bound . . . 82
4.2.2.5 Branch-and-Price . . . 83
4.2.3 Background on Shortest path problems . . . 84
4.2.3.1 Multi-objective SP problem (MOSPP) . . . 84
4.2.3.2 Resource-constrained SP (RCSP) . . . 85

4.2.3.3 SP with forbidden paths (SPFPP) 85
4.2.4 Background on proving NP-Hardness of a problem . . 86
4.3 Contribution 1: Multi-objective multi-constrained shortest path problem with forbidden paths 88
4.3.1 Considered problem 88
4.3.2 Algorithm 89
4.3.3 Computational experiments 90
4.4 Contribution 2: Edge Service placement 93
4.4.1 Graph model and assumptions 93
4.4.2 Integer Programming Problem formulation 97
4.4.3 Proposed Solution 99
4.4.3.1 Simplification to a MILP 100
4.4.3.2 Making the MILP more tractable 101
4.4.3.3 Column Generation-based solution 101
4.4.3.4 Branching rule 103
4.4.4 Performance Evaluation and Comparison 104
4.4.5 Proofs 110
4.5 Contribution 3: Understanding what makes energy minimization harder 119
4.5.1 Energy Minimization Flow Problem (EMF) 119
4.5.2 Proof of NP-Hardness 119
4.6 Conclusion 122

5 Radio Access Network: Integrated Access and Backhaul Routing 125
5.1 Introduction 125
5.2 Background on IAB networks 126
5.3 Contribution 127
5.3.1 System Model and Optimization 127
5.3.1.1 Graph formulation 127
5.3.1.2 Optimization problem 128
5.3.2 Performance Evaluation Setup 132
5.3.2.1 Placement of gNBs and UEs 132
5.3.2.2 Access and Backhaul channel models 133
5.3.2.3 Time-varying UE density model 134
5.3.3 Results 134
5.3.3.1 Energy Consumption 135
5.3.3.2 Capacity 136
5.3.3.3 Runtime 139
5.4 Conclusions 139

6 General Conclusion and Perspectives **141**
6.1 Conclusion 141
6.2 Future Directions 142

A Large version of figures **147**

B Comparison of results for NEPA with and without the Alternative Reward Function Based on Degrees **153**

C Ablation study of NEPA components **157**

D Statistics on network topologies **161**

Acknowledgement

Completing this thesis would have been impossible without the expertise and support of my advisors, Hind, Andrea and Badii. Thank you to all of you for your invaluable and challenging insights as well as your availability. Thank you also for having enabled me to grow and expand my views during the PhD journey. I hope we will keep going with those fruitful and friendly collaborations in the next years. I also want to thank David and Massinissa from Davidson Consulting, who have also been key to deciding directions and making this thesis possible. I would particularly like to thank Massinissa warmly, as I always enjoyed discussing research matters and directions with him.

I would also like to dedicate a very warm acknowledgement to the members of the jury: Tristan Cazenave, Adlen Ksentini, Emmanuel Hyon, Nancy Perot, Stefano Secci and Véronique Vèque for accepting to judge my work on this very special day. I would particularly like to thank Tristan and Stefano, who also accepted to take part to my mid-term defense almost two years ago. Your insights have been key in orienting the second half of my PhD work. I also want to dedicate a special thank to Emmanuel. It was a real pleasure to give my first lessons under your supervision. Thank you for being so friendly, I also really enjoyed our research discussions and wish we will be able to keep going with them in the future.

My PhD would not have been the same without the team at WinesLab who hosted me for 6 months this year. Many thanks to my advisor Salvo, as well as Tommaso. I am also immensely grateful to Andrea, without whom this research visit would not have been possible. This is especially important for me as the people at WinesLab will enable me to keep going with energy-efficiency research in the near future.

This acknowledgment would also not be the same without all the friends I made along the way. Many thanks to all the very nice folks I met both in Palaiseau, Evry and Boston. I am not listing all of you for fear of forgetting someone. It was always nice to hang out with you in the office and struggle together with our papers.

Finally, many thanks to my family: my mom, my dad, my sister and Helena for all the support, love and care you have always provided me. I also cannot forget the support provided by Hélyette, Patrick and Emilie.

Chapter 1

Introduction

1.1 Motivation

The rapid proliferation of mobile devices and the growing demand for high-data-rate applications with stringent Quality-Of-Service (QoS) have lead to the need for a new mobile network standard, the fifth-generation (5G) mobile networks. Because of their promise for higher throughput and lower delay, combined with their programmbility, these networks are seen as enablers for a wide range of new applications, such as augmented reality, autonomous vehicles, and the Internet of Things (IoT). In particular, paradigms such as network slicing, mobile edge computing, and Integrated Access and Backhaul are seen as key technologies to tailor the network to those varied usecases. Network slicing is the possibility of instantiating and scaling customized virtualized networks (deemed slices) on demand. Edge computing is the idea of placing computational resources at the edge, directly next to the users. Integrated Access and Backhaul is the use of millimeter waves for backhauling in the Radio Access Network (RAN), instead of traditional optics fiber

However, the unprecedented surge in data traffic, coupled with the diverse requirements of the afforementionned novel applications, presents significant challenges to mobile network operators and service providers: those different services have different Quality-of-Service requirements that sometimes contradict one another. For example, serving autonomous vehicle networks requires very stringent delays (in the range of 7ms) and reliability (in the magnitude of 99.999%) [1], while video streaming services are less concerned with those metrics but require higher throughput. This can create a conflict because, for example, serving high throughput to those services might mean using a lot of the network bandwidth, hence increasing the delay of the other applications if they are not properly isolated. On the other hand, another key challenge is to make the network consume less

energy. This is known to represent 60% [2] of the total OPEX of network operators, and furthermore, cellular networks are known to represent up to 3.5% of the global carbon footprint [3].

In this context, the new programmability of 5G networks is a big opportunity for adressing both challenges: traditional cellular networks were primarily designed with the goal of providing ubiquitous coverage and high data rates. However, this approach often leads to inefficiencies in energy utilization, as network resources are allocated uniformly across all users and services, regardless of their specific needs. As 5G networks embrace a more flexible and adaptive infrastructure, there arises a unique opportunity to optimize energy consumption. This same adaptability is also key in satisfying the variety of services 5G has to support: instead of a one-size-fits-all networking paradigm, we can allocate the right amount of resources at the right place.

This shift also comes at the cost of complexity: since traditional networks didn't rely on customizability too much, it was sufficient to use the same simple policies or heuristics in all parts of the network. This is not possible if we want to tailor the resource usage to the varied, dynamic usecases we described. Hence, a key question is how to design algorithms which take all those aspects into account, in all parts of the network. Furthermore, the solutions should be both fast and optimal or near-optimal. For this reason, it is required to come up with efficient algorithms. Namely, in this thesis, most of the problems will be modeled within the framework of graph theory [4], and in particular most problems come down to flavors of multicommodity flow problems [5, 6], *e.g.* the problem of routing a set of goods (in our case, data traffic) over a network. To solve these problems, we shall focus on Reinforcement Learning (RL) approaches, as they have been shown in the last few years to be able to generalize well and provide fast solutions to hard problems. Then, we also resort to Integer Linear Programming (ILP) based solutions, since off-the-shelf mature solvers are available to solve such problems to optimality, even if they are theoretically intractable.

Due to the complexity and heterogeneity of the network, combined with the variety of the usecases and the different constraints they present, it is not easy to design a single end-to-end solution to the problems. In the core network, the main issue is purely resource usage. At the edge, instead, meeting low delays is equally important, since if the edge experiences high delay, the benefits of edge computing are negated. On the Radio Acess Network, the optimization is much more confined to what happens at a single base-station level than to the bigger picture of a full graph. For

those reasons, in this thesis, we make the choice of designing optimization algorithms by progressively getting closer to the user: we start by designing optimization for the core network, then go down the edge network, including base stations. Finally, we get even closer to users by designing radio backhauling techniques for the RAN.

1.2 Contributions

We divide our contributions in three parts, corresponding to one chapter each.

1.2.1 Virtual network embedding for Core Network Slice Placement

In Chapter 3, we solve the core network slice placement problem, *i.e.*, we have slices arriving over time which ask for resources in order to serve their users. In particular, this chapter is interested in the core network part of these slices, *i.e.* a set of Virtualized Network Functions, which consume a certain amount of CPU, along with links between these functions, which need a certain amount of bandwidth. In particular, our goal in that chapter is to maximize the number of slices we manage to place. We first model the problem as a graph placement problem, which we then formulate as a Markov Decision Process (MDP). We then solve this MDP using the Nested Rollout Policy Adaptation (NRPA) algorithm, which is an existing Monte Carlo Search algorithm. We observe that running neighborhood search on all the solutions found by NRPA would be too costly. Hence, we study the structure of NRPA in order to run Neighborhood Search only on the promising solutions, which enables us to get the most out of Neighborhood Search without sacrificing speed. We call the resulting algorithm Neighborhood Enhanced Policy Adaptation (NEPA). This method enables us to increase the acceptation probability of core network slices of up to around 30% compared to the state-of-the-art.

1.2.2 Edge slice placement

In Chapter 4, we focus on the placement of slices at the Edge. The problem is modeled as a non-linear integer program where we try to minimize the energy consumption and respect quality-of-service constraints (end-to-end delay, including processing delay on the computational nodes, which are modeled using queuing theory). Then, we linearize the problem and then leverage column-generation, which is a specialized technique for solving ILPs with a large number of variables. This technique consists in solving

the problem by iteratively adding variables that are promising. To do so, it is required to solve an optimization problem called the pricing problem. In our case, we prove that the pricing problem is equivalent to a bi-objective shortest paths problem with forbidden paths. Hence, we propose a new algorithm to solve this problem. Using those tools, we obtain an exact algorithm for the edge slice placement problem. We show that our method enables us to divide by up to almost two the amount of energy consumed by the network compared to using classic approaches which optimize for the amount of bandwidth used. However, these gains come at the cost of runtime, as our approach is inherently slow. For this reason, we theoretically investigate the hardness of the energy optimization problem. We show in particular that a simplified version of the classic bandwidth-optimization problem, which is polynomial, becomes NP-hard if considering energy optimization. This partly explains why our algorithm is slower.

1.2.3 Integrated Access and Backhaul

In Chapter 5, we design an algorithm for energy-efficient Integrated Access and Backhaul (IAB) topologies. The idea of IAB is to leverage millimeter wave-based radios for backhauling: we focus on cases where there are several base stations in one zone, but a single one has an optical fiber connected to a remote core network. Hence, we have to find a topology to connect all the users, to the core network, all through the radio spectrum. Furthermore, we seek a topology which minimizes the energy consumed by the network, which, in our case, depends on the amount of nodes that are turned on. We also focus on finding a tree topology, as it is one of the specific cases defined by 3GPP. We then formulate this problem as a non-linear mixed integer problem. Since the formulation cannot be solved using off-the-shelf solvers due to the non-linearity, we prove that there exists an equivalent linear reformulation. The proof is based on the observation that since we are looking for a tree topology, minimizing the number of nodes is equivalent to minimizing its number of edges. We then evaluate our algorithm based on a real dataset giving us a 3D scan of an area of Milan, which we feed to an existing approach for placing IAB nodes. Once the nodes are placed, we also feed the dataset to a raytracing channel model in order to obtain realistic bandwidth capacities. The algorithm is then compared to several simple baselines, and we show that our algorithm is the only one which ensures the energy consumption is low while making sure the users are served a certain minimum amount of bandwidth. We reduce the energy consumption by up to 47%.

1.3 Thesis organization

The thesis is organized as follows:

- In the present Chapter, we introduce the thesis
- In Chapter 2 we review the architecture of the 5G network as well as the main networking paradigms which support it.
- In Chapter 3 we focus on the placement of slices in the Core Network. We model the problem as the Virtual Network Embedding (VNE) problem, where the goal is to place Virtual Networks (*i.e.* slices) on a physical network infrastructure, so as to minimize the resource utilization. We leverage RL, in particular of the Monte-Carlo-Tree-Search type to solve the problem. This chapter led to two publications [7] [8].

  - *ELKAEL, Maxime, AIT ABA, Massinissa, ARALDO, Andrea, CASTEL-TALEB, Hind, JOUABER Badii.* ***Monkey business: Reinforcement learning meets neighborhood search for virtual network embedding.*** *Computer Networks, 2022, vol. 216, p. 109204.*
  - *ELKAEL, Maxime, CASTEL-TALEB, Hind, JOUABER, Badii, ARALDO, Andrea, AIT ABA, Massinissa,* ***Improved monte carlo tree search for virtual network embedding.*** *In : 2021 IEEE 46th Conference on Local Computer Networks (LCN). IEEE, 2021. p. 605-612.*

- In Chapter 4 we focus on the placement of slices at the Edge. The problem is modeled as a non-linear integer program where we try to minimize the energy consumption and respect quality-of-service constraints. We linearize the problem and then leverage column-generation, which is a specialized technique for solving ILPs with a large number of variables. This chapter led to one publication [9], and two currently in preparation

  – *ELKAEL Maxime, ARALDO, Andrea, D'ORO, Salvatore, CASTEL-TALEB, Hind, AIT-ABA, Massinissa, JOUABER, Badii,* ***Joint placement, routing and dimensioning at the network edge for energy minimization.*** *In Globecom 2023.*

  – *ELKAEL Maxime, ARALDO Andrea, CASTEL-TALEB, Hind, JOUABER, Badii,* ***An Exact Algorithm to Solve Multi-objective, Multi-Constrained Shortest Path Problems with Forbidden Paths, in preparation***

  – *ELKAEL Maxime, ARALDO, Andrea, D'ORO, Salvatore, CASTEL-TALEB, Hind, AIT-ABA, Massinissa, JOUABER, Badii,* ***Joint placement, routing and dimensioning at the network edge for energy minimization: extended journal version, in preparation***

- In Chapter 5, we design an algorithm for energy-efficient radio-backhauling topologies. The solution is based on ILP and this chapter led to one publication [10] along with one journal extension that currently is in preparation.

- *GEMMI, Gabriele, ELKAEL, Maxime, POLESE, Michele, MACCARI, Leonardo, CASTEL-TALEB, Hind, MELODIA, Tommaso,* ***Joint Routing and Energy Optimization for Integrated Access and Backhaul networks.*** *In Globecom 2023.*
- *GEMMI, Gabriele, ELKAEL, Maxime, POLESE, Michele, MACCARI, Leonardo, CASTEL-TALEB, Hind, MELODIA, Tommaso,* ***Joint Routing and Energy Optimization for Integrated Access and Backhaul networks: extended journal version, in preparation***

- Finally, in the last Chapter we give our Concluding remarks and Future Directions.

Chapter 2

Background

2.1 Introduction

The currently developping 5G network is expected to go beyond 4G by providing performances (in terms of throughput, delay and reliability) an order of magnitude higher than that of LTE. These new characteristics will enable new usecases such as autonoumous vehicles, drone swarms, Virtual/Augmented reality or connected industry. However, this comes with challenges: each usecase has different requirements in terms of quality-of-service, particularly delay and reliability: for example, it is not acceptable to lose too many packets in an autonomous vehicles network, as it would put safety at risk.

Hence, these new networks come with the challenge of making the differents usecases work in isolation, while satisfying all of their requirements. For this reason, the 5G network comes with a new architecture which is more flexible than LTE, making it more suitable for supporting the variety of usecases.

Hence, in this section, we describe the 5G Network's components as well as the key concepts we shall use throughout the thesis, basing ourselves on the current litterature as well as on the architecture standardized by 3GPP. We shall detail the different parts and the main components of the network, as we will later design optimization algorithms around them. This section is organized as follows: section 2.2 details the architecture of the network, the section 2.3.2 explains the concept of network slicing, and finally, 2.3.3 details what the Integrated Access and Backhaul network is.

2.2 Infrastructure of the Network

We start with a description of the 5G network considered in the thesis. We shall divide our explanation in three parts. First, we describe the core

network, which is the part of the network that is dedicated to all the management aspects of the network, *e.g.* authentication, security, billing, routing, and making gateways to other networks (mainly, internet). We then interst ourselves to the Radio Access Network (RAN), which ensures the connection with User Equipments (UEs) *e.g.* phones and the core network, by the means electromagnetic waves. We shall then describe the features of the transport network, which connects the core and the RAN. Finally, we shall describe Mobile Edge Computing (MEC), as it is envisionned as a key architecture for meeting the reliability and delay requirements standardized in 5G.

2.2.1 Core Network

In order to highligh the features of the 5G core network, let us start by going through those of the 4G/LTE network, which is called the Evolved Packet Core (EPC). In 4G, the EPC (represented in Figure 2.1) is comprised of 4 main elements which we call network functions:

- The Home Subscriber Server (HSS) is a database providing the subscriber details to the other network functions, *e.g.* it contains the user profiles and keeps track of the security authorization, authentication details and location of the users.

- The Mobile Management Entity (MME) is tasked with handling authentication and authorization processes for user devices attempting to access the network. It manages user context, tracks the location of devices through their attach and detach procedures, and facilitates the seamless handover of devices between different base stations. The MME also coordinates the establishment and termination of user sessions, working in conjunction with other network elements like the Serving Gateway (SGW) and Packet Data Network Gateway (PGW). Additionally, the MME plays a critical role in mobility management by maintaining information about active user sessions and their respective locations, ensuring efficient routing and optimized resource allocation. Through these functions, the 4G MME contributes significantly to the overall reliability, security, and performance of LTE networks.

- The Serving Gateway (SGW) functions as the interface between the radio access network and the core network. It handles tasks such as packet routing and forwarding. Additionally, the SGW performs tasks related to charging, such as collecting and reporting usage information for billing purposes.

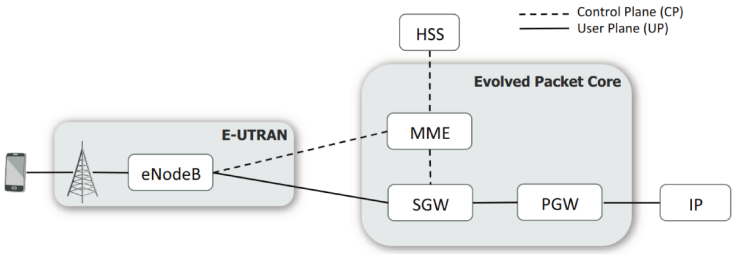


Figure 2.1: Evolved Packet Core Architecture, figure from [11]

- The Packet Data Network Gateway (PGW) assigns IP addresses to user devices, manages Quality of Service (QoS) policies, and performs deep packet inspection to enforce security measures like firewalling and filtering. The PGW also makes the connection between the 4G network and other networks, *e.g.*, mainly, the internet

At the commercial level, the 4G is typically implemented in proprietary hardware. It can also be done in software, but in any case, most of the work is done by the MME. This leads to software implementations such as SRSRan [12] or OpenAirInterface [13] where all parts of the core are a single, monolithic unit. The direct consequence is that even if implemented in software, the 4G core network is not adaptable to the different services and requirements we mentionned earlier, as all services have to be served in a very similar way by the same piece of software. This problem is even more pregnant with hardware implementations, as they are black boxes which the operator cannot customize

This is the key aspects adressed in the design of the 5G core network. Namely, instead of being a monolithic software (where most of the work happens in the MME), the core network is broken up in more network functions, each with a tinier scope. As shown in Figure 2.2, the architecture is that of a logical bus, *e.g.* all control plane functions can communicate with one another, and elements of the core can be included or excluded depending on the usecase at hand. Let us now desribe the different services involved.

- The Access and Mobility Management Function (AMF) performs the initial device registration, authentication, session establishment, and mobility management as devices move. It interacts with UEs and other network functions to establish and maintain connections, manage handovers, and ensure mobility transitions.

- The Session Management Function (SMF) is responsible for establishing, maintaining, and terminating Packet Data Unit (PDU) sessions

for UEs. Establishing a PDU session means to setup the path from the UE and the core.

- The Network Repository Function (NRF) serves as a central repository for network function and service information. It maintains a registry of available network functions, their capabilities, and their associated service profiles. This information is used by other network functions to dynamically locate and interact with the required services as per the specific requirements of UEs and applications. Having such a repository allows network operators to adapt to changing service demands and optimize network performance by switching components and optimization algorithms when needed. Its key role is also to maintain an up-to-date, accurate representation of the network topology and available services.

- The Network Exposure Function (NEF) enables controlled and secure exposure of network capabilities and services to authorized external entities. Facilitates the interaction between third-party applications, services, and the 5G network, acting as a gateway for external entities to access network functions, service information, and subscriber data, while maintaining strict authorization and privacy controls.

- The Authentication Server Function (AUSF) manages user authentication and security functions. It is responsible for verifying the identity of UEs during the initial network attachment and subsequent connection establishment processes. The AUSF also has the role of deriving, distributing and managing cryptographic keys.

- The Policy Control Function (PCF) manages network-wide policies for quality of service (QoS), traffic prioritization, and resource allocation. It translates policy decisions (or intents) into actionable instructions that influence data flow and network behavior. The PCF dynamically adapts policies to optimize network performance based on factors like network congestion, user preferences, and service requirement.

- The Unified Data Management (UDM) is responsible for managing and storing subscriber-related data and profiles. It is the 5G equivalent of the HSS.

- The Application Function (AF) is responsible for interacting with applications and services on the application layer. Its main role is to manage and enforce application-specific policies and quality of service (QoS) parameters. It receives information about the application's

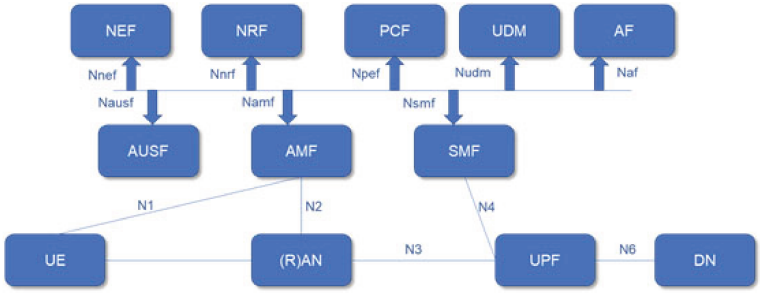


Figure 2.2: 5G Core Architecture, figure from [14]

requirements, such as QoS expectations and data priority, and communicates these requirements to the other network functions (mainly the PCF). The AF acts as an interface between applications and the network, influencing how network resources are allocated and utilized to support the needs of various applications.

- Finally, the User Plane Function (UPF), serves as a gateway to the Data Network (DN), *e.g.* external networks. The UPF replaces the PGW and SGW in 4G.

With the past cellular technologies, these network functions would have been implemented in hardware. However, the new trend (adopted in 5G) is to leverage Network Function Virtualization (NFV), a concept which consiss in using virtualization technologies to decouple network functions from proprietary hardware, allowing them to be executed as software instances. This is done with the goal of providing operators with more flexibility and the ability to dynamically instantiate these functions. Hence, those network functions are typically implemented as software container or virtual machines . This enables the operator to instantiate, duplicate, scale and remove the functions as needed, depending on the load and the user profiles. This implies those functions can be instantiated on decentralized systems, such as datacenters. Those characteristics motivate the need for placement algorithms for the core network, as, depending on the usecase, some of these functions might not be necessary, or they might have to be configured in a specific way. Parameters such as the amount of bandwidth used to interconnect them, the computational resources they have to obtain or the isolation requirements would lead to varying placement choices which we study in this thesis.

2.2.2 Radio Access Network

Let us now focus on the Radio Access Network (RAN). The RAN is the part of the network which directly interacts with the UEs through electromagnetic waves. 5G Also introduces key novelties at the RAN level in

order to enable new usecases. These novelties, implemented in the Base Station (BS) are:

- Higher Frequency Bands: 5G introduces the use of higher frequency bands, including millimeter-wave (mmWave) frequencies, which enable wider bandwidths for faster data transmission. This results in significantly higher data rates and lower latency compared to previous generations.

- Massive MIMO: 5G RAN utilizes Massive Multiple-Input Multiple-Output (MIMO) technology, which involves deploying a large number of antennas at both the base station and user equipment. This technology enhances spatial multiplexing, allowing multiple data streams to be transmitted simultaneously to multiple users, improving both capacity and coverage.

- Beamforming: Beamforming technology is heavily utilized in 5G RAN. It focuses the transmission and reception of signals in specific directions. This avoids having to emit in an omni-directional way, which yields to more interferences and less spectral efficiency. This is also of key importance for mmWave, since those signals typically require to have line-of-sight between the BS and the UE.

- More flexible physical layer protocols: in 4G, parameters of the physical layer such as numerology and signaling delays are much less customizable. Changing such parameters enables the operator to adapt to the current usecase. For example, increasing the signaling delays (*e.g.* reducing the frequency at which the base-station emits control messages) might enable the base-station to get into sleep modes more often, which reduces energy consumption, at the cost of reactivity when users show up. This would typically be done in a low-usage scenario where reliability constraints are not too stringent (for example in a business neighborhood at night).

Besides those, the 5G RAN's architecture also provides a major shift from a proprietary hardware-based implementation to more open software components, similar to what the 5G core provides compared to the 4G EPC. The enabler of this change is Open RAN, it specifies standardized interfaces between the different components of the RAN. This has the same advantage as in the core: the operator can swap components in and out at will, and modify or parameterize them according to its usecases. The main elements of the Open RAN, which are shown in Figure 2.3 are:

- The Radio Unit (RU), which communicates with the UEs through its antennas

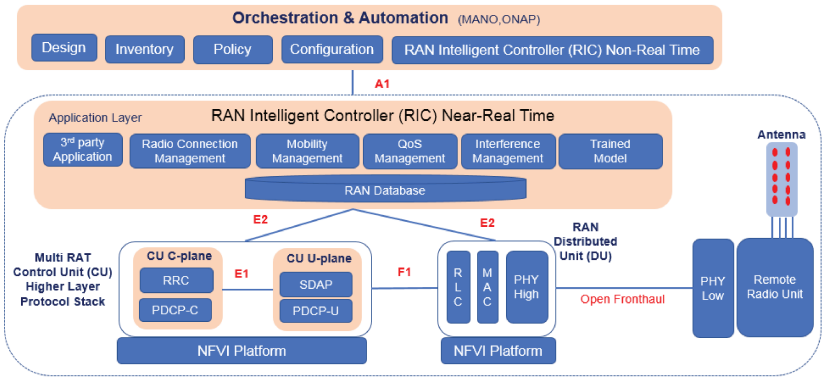


Figure 2.3: Open RAN architecture, figure from [15]

- The Distributed Unit (DU) processes the signal. It performs functions such as encoding, decoding, modulation, and demodulation. In short, it takes care of all the operations required to transform the analog signal into the digital domain. It also manages radio resource allocation and scheduling. Typically, the DU is located close to the RU, and can either be implemented in hardware or in software. Note that having a centralized DU for several RUs can be of interest since it allows to improve performances of the network (for example, having information on the waveform sent by several RUs enables the DU to do more advanced interference cancellation). This however comes with the tradeoff of using more computational power.

- The Control Unit (CU) is responsible for managing and controlling multiple Distributed Units (DUs). It handles higher-level functions such as coordination, network management, and resource allocation across different DUs. The CU is typically located in a centralized data center or cloud environment.

- The RAN intelligent controllers (which can either be real-time, near real-time or non real-time, depending on the tasks they have to manage) are software controllers which interact with the CU and DU: they recieve the state of the network (through interfaces such as the E2 interface shown in Figure 2.3) and send back controls, such as orders on how to allocate radio resources, modified RAN parameters, or new slicing policies (more on this will be explained in section 2.3.2).

2.2.3 Transport Network

We now have introduced all the components that can be hosted on servers of the network: the different components of the core, the CU, the DU and

the RICs. We shall now describe the Transport Network (TN) considered in this thesis, which is in charge of interconnecting those elements. Note it also connects the RU and the DU, although the RU is not a software element.

In a similar fashion as the rest of the network, the TN should be adaptable, manageable and parameterizable, so as to adapt to specific usecases. For this reason, in this thesis, we shall consider that the TN is based on the Software Defined Networks (SDN) [16] paradigm.

At its root, the TN is made up of switches and routers. Without SDN, such network elements used routing protocols such as OSPF [17] or IGRP [18] which are designed to work well in a variety of situation, and typically in a decentralized manner: the protocols were hardcoded in the routers, so they were easy to setup, at the cost of being parameterizable and modifiable.

Instead, the idea of SDN is to use a protocol such as OpenFlow [19]. Such a protocol considers that the network has a centralized controler, which can send routing tables to the SDN routers. Those routers then use the tables to determine actions depending on the incoming data. For example, the controller could tell a given router to send a packet through interface A if it comes from IP X using TCP, through interface B if it comes from IP Y, and through interface C otherwise. This simple concept enables the operator to perform advanced operations such as reserving bandwidth on a path for a link between two given elements or to prioritize some type of traffic.

2.3 New Service and Networking paradigms

We now introduce the two main networking paradigms which are seen as the enablers for hosting different services and making them use the same infrastructure. Namely, they are network slicing, and Integrated Access and Backhaul (IAB). We also introduce the new service paradigm of Mobile Edge Computing.

2.3.1 Mobile Edge Computing

We start with Mobile Edge Computing (MEC). MEC, also sometimes deemed Fog Computing, is a networking paradigm that arose after the advent of cloud computing. With cloud computing, the idea was to abstract away the servers and the hardware on which the software runs. Instead, the software can be encapsulated in containers, which are given to cloud providers. It is then possible to scale the resources allocated to containers when necessary. This scaling can either be vertical, *e.g.* by increasing the

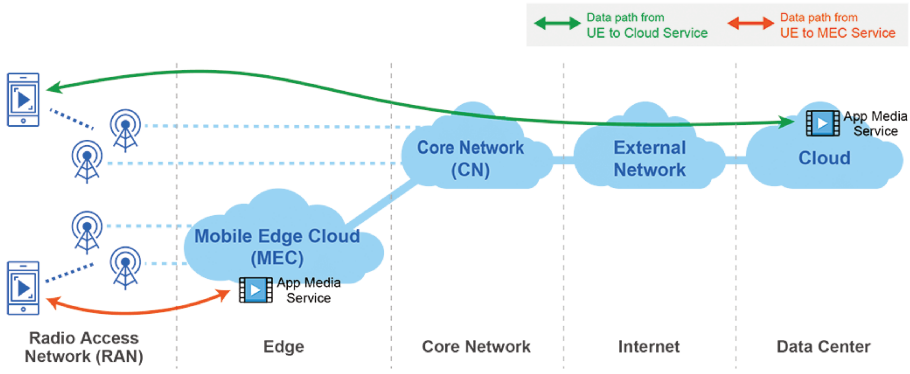


Figure 2.4: Example 5G network with MEC, figure from [20]

portion of resources of a single server allocated to the container, or horizontal *e.g.* by duplicating the container on several other servers.

The drawback of cloud computing is that in order to perform this scaling in a virtually infinite way, containers have to be placed in datacenters, which can be far away from the users. Hence, for very stringent usecases such as some of the ones we want to address, using cloud computing can induce too much delay or bandwidth consumption.

For this reason, MEC was introduced. The proposition of MEC is to place the part of the network which is the most constrained (in terms of delay) closer to the user. The benefit is that the propagation delay is lower. It also reduces the amount of bandwidth consumed because shorter paths are used. In particular, MEC is often concerned with placing applications closer to the user. For example, if a video service requires delays of less than 5ms, it is interesting to place the video encoding application along with a database of the most watched videos on a server close to the user and not in a datacenter. Note that due to the 5G architecture, this requires a container-based architecture: the application server is external to the 5G network and hence any request to it needs to go through a UPF. Hence, we have to be able to move, duplicate and scale the UPF to support MEC.

These benefits come at the cost of a complexified network: hosting services directly in the mobile network, close to the users means that the TN has to be augmented with servers. These servers can be used to host applications from third-party service providers as well as core network functions such as the UPF. It is also feasible to host the CU or the DU on those same servers. We illustrate the concept of MEC and a potential network in Figure 2.4

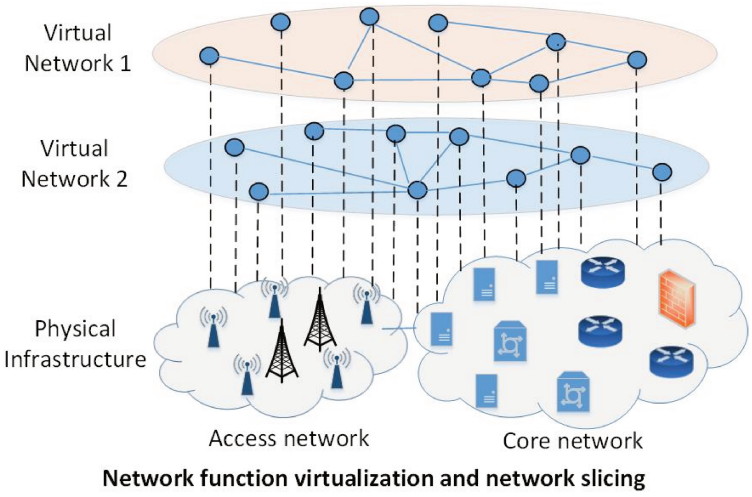


Figure 2.5: Example sliced network, figure from [21]

2.3.2 Network Slicing

We turn to the other main concept which can help support a variety of usecases, namely, network slicing (NS). NS involves the creation of multiple virtualized network instances or "slices" within a single physical network infrastructure. Each network slice is essentially a dedicated and isolated portion of the network tailored to specific requirements, applications, or user groups.

The key idea behind network slicing is to optimize the use of resources and provide customized services to different types of users or applications. Instead of having a one-size-fits-all network architecture, network slicing allows network operators to allocate resources and configure network parameters on a per-slice basis. This enables them to meet the varying performance, latency, capacity, and security demands of different services. The three 3GPP-defined classes are enhanced mobile broadband (eMMB – high speed and capacity for media usecases such as AR/VR and high definition video), massive machine-type communications (mMTC – large number of devices with low energy and data rate), and ultra-reliable low-latency communications (uRLLC – extremely high reliability, in the order of 99.999% and low delay, in the order of 1 to 5ms).

The concept of network slicing is illustrated in Figure 2.5. Let us now detail how network slicing can concretely happen in the different parts of the network:

- In the CN, the network functions are hosted either in virtual machines or in containers. One can use an orchestrator such as Kubernetes [22], in order to scale (vertically or horizontally) and duplicate those containers. In this case, the sliced resources (*e.g.* the resources shared

by the slices) are the resources of the server: CPU, memory, network card bandwidth, etc.

- In the RAN, the resources are the radio resources. Recall each RU functions by using a certain portion of the electromagnetic spectrum for communication with the UEs. 3GPP has defined Physical Resource Blocks (PRBs) as the minimum unit of spectrum to be allocated to each slice. Each PRB represents a certain portion of the spectrum. The exact amount is specific to the parameters of the RU.
- In the TN, the sliced resources are the bandwidth of each link between two routers.

The last element needed for implementing slicing is to be able to identify to which slice each packet belongs. The 5G standard implements this with PDU sessions: upon connecting itself, a UE establishes a PDU session with the SMF. When it does so, it provides the id of the slice it connects to. Then, if this matches one of the authorized slices for the user, stored in the UDM, then the connection is established. Then, the traffic has to be treated in a differentiated manner depending on the slice id. 3GPP does not indicate specification for handling traffic at the RAN. However, one possible implementation, which was chosen for some RAN slicing platforms such as SCOPE [23], is to forward all the traffic from/to the UE in the RAN through dedicated buffers for this slice, making sure that each UE uses dedicated resources. On the rest of the transport network from the RAN to the core network (and the other way around), the same concept of buffers can be implemented on each SDN router on the way.

Finally, note that in practice, the concept of network slicing is tightly related to that of MEC: in order to reach the stringent constraints imposed on each slice, it can be necessary to leverage edge computing in order to reduce delay, provide more throughput and use less resources for the service of some of the slices (particularly the most constrained ones, *e.g.* uRLLC).

2.3.3 Integrated Access and Backhaul

Finally, we introduce the third new 5G networking paradigm for which we shall design optimization algorithms in this thesis. This concept is called Integrated Access and Backhaul (IAB). IAB was coined as a solution for deployment in extremely dense areas such as a crowded city center or a football stadium. In such places, it is required to serve an enormous amount of users at the peak of frequentation, and less so in less crowded moments.

One solution to this problem is to add more base-stations in the area. However, this can be difficult because of the cost of installing such equipments, particularly of laying new optical fiber cables. For this reason, IAB

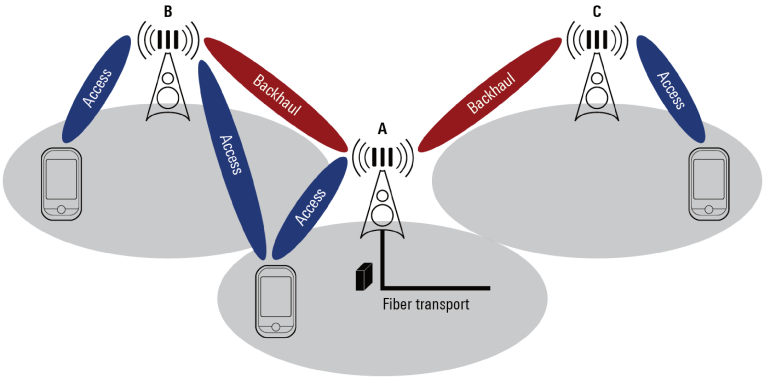


Figure 2.6: Example IAB network, figure from [24]

is a cost-efficient alternative in which only a single or a few base-stations in the area (called the IAB donors) are linked to an optical fiber. The rest of the nodes (called the other IAB nodes) do not have an optical fiber, instead, they rely on radio communication to communicate with the rest of the network and send data up to the core network. This means an IAB network is built in a similar fashion as a mesh wireless network: one can build and modify a network topology in which the IAB nodes connect to one another, with the goal of reaching the IAB donor. In this way, it is possible to connect the IAB nodes to the rest of the network (this is sometimes called "backhauling") without laying optical fibers. The concept of IAB solves the issue of handling peaks of traffic, as it makes it feasible to deploy temporary base-stations, or to deploy much cheaper permanent base-stations, which can only be turned on to handle those peaks of traffic. Furthermore, in remote areas, such as mountains, it is not always feasible to deploy optical fibers. This is another usecase for IAB.

The key technological advances on which IAB relies are millimeter waves and massive MIMO, as they make it possible to send the huge amount of data required to perform backhauling, as well as beamforming, which ensures that it is possible to emit from one BS to the other in a very directional way, avoiding interferences and improving throughput. Finally, note IAB is enabled by the 5G architecture, as it relies on the disaggregation of the 5G RAN which we presented earlier: the IAB donor comprises a CU, while the other nodes only have a DU. We illustrate the concept of IAB with a small example network in Figure 2.6.

2.4 TopologyZoo Dataset

Throughout this thesis, we shall evaluate the proposed algorithms experimentally. One particular dataset, namely, the TopologyZoo dataset [25], will be used throughout several chapters to build testing instances for the

problems/algorithms at hand. The reason we chose this specific dataset is that it is a manually curated set of real-life networks. While it is not a dataset of 5G mobile networks, it is the closest to real-life mobile networks publicly available. In particular, this dataset contains varied topologies in terms of sizes (from a few nodes to up to 850 nodes), scale (from city-wide to continent-wide networks, including regional and country-wide networks). The dataset includes topologies from a wide range of domains, such as data center networks, wireless networks, or research networks.

Topology Zoo does not have a single, centralized source. Instead, it aggregates topologies from different papers and projects. Researchers and network operators contribute their network topologies to the dataset to make it publicly available to the broader networking community. This dataset has been used in various research studies to evaluate network algorithms, protocols, and performance in different network environments. Researchers often rely on it to conduct simulations and experiments to test the scalability, robustness, and efficiency of various networking solutions.

The networks of the Topology Zoo dataset have the notable property that 80% of them are planar [26], *e.g.* they can be drawn such that no two edges cross each other. This seemingly simple property makes a lot of problems simpler in such graphs. For example, the shortest path problem [27], the graph coloring problem [28] or the Multicommodity-Flow problem [29] all become easier if the graph is planar.

2.5 Conclusion

In this chapter, we have detailed the main characteristics of the 5G network that we will keep in mind for the rest of this thesis. Mainly, we have seen that the 5G network can be split into three main domains, *e.g.* the RAN, the edge and the core, and that each of these domains are being increasingly softwarized. The stance we adopt in the remainder of the thesis is to mostly abstract away the details of the implementation of the network: while we are mindful of the overall architecture when modeling the network, our goal is to design general algorithms based on graph-theoretic models that are applicable in the current cases as well as slight variations in terms of architecture or even type of network. Hence, we do not detail how our algorithms play with specific network functions, schedulers, etc, although we sometimes reference them to motivate our models. Furthermore, when modeling, it is hard to draw a strict line between the transport, the core, the RAN and the edge network: since the transport network interconnects all the elements of the 5G network, when we perform optimization dedicated to one of the components, it often implies that we have to chose how to route the traffic between the different network elements. This is

typically a decision that is implemented on the transport network, *e.g.* by programming SDN switches. For this reason, the approaches described aim at optimizing for either the core, the edge or the RAN, but also always take the transport network into account.

Chapter 3

Core Network Slice Placement: Virtual Network Embedding Problem

3.1 Introduction

In this chapter, we shall focus on the placement of core network slices (CNSs): clients give the operator CNS (therafter also called virtual network) requests in the form of interconnected NFs (i.e. a graph), and the operator tries to embed them onto the physical infrastructure (*i.e.* to provide enough CPU for each virtual node and enough bandwidth for each virtual link between those nodes), by accepting as many CNSs as possible, so as to maximize the operator's gain. This problem is known as Virtual Network Embedding[30] (VNE) problem which has been extensively studied in the recent years[31][32][33]. The VNE being NP-hard and inapproximable [34][35], running an exact algorithm is not an option in most cases. Various methods have been studied for this problem, among which many are heuristic algorithms based on Linear programming[31], ranking algorithms[32] or reinforcement learning (RL) methods [36][37]. The contribution in this chapter is that we design a RL based VNE algorithm, as this type of approaches enable to construct heuristics in an autonomous manner, based on experience and learning while solving the on-line version of the problem, where CNSs arrive and leave the system over time. Research on RL methods is still lacking, as current neural networks based methods either have hard constraints on the network topologies [38] or require very large amount of computing resources for training [37]. On the other hand, Monte Carlo based methods such as [36] still have large room for improvement, as we show in this work. These shortcomings of other RL approaches are further developed in section 2.

3.1.1 Summary of chapter contributions

The new state-of-the-art RL algorithm we develop is called Neighborhood Enhanced Policy Adaptation (NEPA). It combines reinforcement learning techniques with neighborhood search. To our knowledge this is the first time the Monte-Carlo Tree Search (MCTS) based Nested Rollout Policy Adaptation (NRPA) algorithm is complemented with a neighborhood search technique for any problem, enabling it to beat several state-of-the-art algorithms. The MCTS approach (called Maven-S) from [36] uses UCT (Upper Confidence bound for Trees)[39], which is adapted to stochastic problems. On the other hand, NRPA is specifically adapted to deterministic optimization problems. In our case, we do not know the CNSs in advance making the arrival process stochastic; however, once a CNS arrives it is fully observable, and so is the physical network, making NRPA more adapted to tackle the placement of CNSs, given the current state of the network. NRPA learns through exploring the NF placement possibilities of the CNS at random while learning weights for biasing future explorations, which enables it to focus on regions of the search space that have been the most rewarding so far while still maintaining a good level of exploration. These characteristics make it a very efficient algorithm for solving the VNE. However we show it can be further enhanced when combining it with neighborhood search. The key idea is that NRPA bases its search on the tree structure of the search space, which is good for quickly finding good solutions. However it can limit exploration of new, better branch once the algorithm has converged. Neighborhood search enables us to exploit knowledge of those good solutions for jumping to better branches of the search tree (similar to how monkeys jump from branch to branch) and continue the search from there, which enables better future exploration. We also propose a heuristic for initialization of the weights, and we show our numerical results, showing an improvement in the CNS acceptance probability on real and synthetic networks compared to other methods, and therefore an increase in financial gains for an operator. Our contributions in this chapter are then the following :

- We assess whether an exact ILP approach can be made faster by only using a subset of the candidate paths

- We combine NRPA with neighborhood search and our heuristic weight initialization, deriving the Neighborhood Enhanced Policy Adaptation (NEPA) algorithm for the virtual network embedding problem which outperforms state-of-the-art methods in both acceptance and revenue-to-cost ratio on all tested instances, including both synthetic and real topologies. Our approach is particularly effective on real topologies, on which it can even triple the number of accepted CNSs

compared to some of the previous algorithms. We also investigate the topological features of those real topologies and explain how our algorithm can exploit them. Note that NRPA had never been used for the VNE problem.

- We publish a large set of testing scenarios for the community to experiment with, patching a lack of publicly available instances for quicker experimentation and comparisons (126 instances).

- We publish our implementations of several algorithms publicly (including NRPA, NEPA and algorithms from [40] [36]), since during this work, we found most algorithms lacked a well-documented implementation.

- To our knowledge, we are the first to explore the combination of NRPA with neighborhood search for any problem. We believe the idea can be exploited in other application where NRPA has been successful and where good neighborhood search algorithms are known such as the Travelling Salesman Problem (TSP)[41] or the Vehicle Routing Problem (VRP)[42]

- Finally, we assess whether the results of NEPA for the VNE can be improved by utilizing the reward function described in [43] (see Appendix B).

- All the afforementioned results are generated on instances comprising synthetic random topologies as often done in the litterature, as well as real networks and perfectly solvable instances, which are synthetic instances better suited to quantify how good our algorithm does compared to an optimal solution.

- We shall see that on all those types of instances, NEPA performs better than the other algorithms from the litterature in terms of number of accepted CNSs and operator's revenue.

The chapter is organized as follows : Section 3.1 introduced our work, section 3.2 presents our literature review of the VNE, then section 3.3 presents our model. We describe NEPA in section 3.5, section 3.5.2 presents our numerical experiments, and in section 3.6 we summarize our work and we propose extensions and future perspectives.

3.2 Pre-existing methods for VNE

Several methods have already been proposed for the VNE problem.

3.2.1 Mathematical programming

Famously, some work has been done for exact VNE using Mathematical programming. In [44], the authors propose an ILP formulation. This has the advantage to give guaranteed optimal solutions. However, since the VNE is NP-hard, such an approach would not be able to cope with even medium CNSs with a reasonable execution time. Hence, a lot of work in the literature focus on heuristic algorithms. In [31], two heuristics based on linear programming and rounding (either randomized or deterministic) are derived. These give good results in terms of acceptance and revenue-to-cost ratio, although most other approaches manage to beat them ([36][40][37]). These two rounding heuristics also sometimes suffer from relatively high runtimes, as [37] shows they run up to 13 times slower than the approach from [36] for worse results, and that for some cases the approaches are even unable to run due to a lack of computational resources. In [45], an ILP heuristic is derived by reducing the number of candidate paths to a small amount, which enables the solver to find a solution quicker. However since it is ILP-based the algorithm is still non-polynomial. Our approach addresses these issues by proposing a solution which both runs quickly (sub-second runtime) and provides high quality (state-of-the art) embeddings.

3.2.2 Graph neural networks

Some recent papers [46][47] process the problem with a deep neural network for performing the embedding (note that in this section we do not consider approaches using neural networks in conjunction with RL). In [46], the graph is clustered with a graph neural network, which then helps guide the embedding procedure. On the other hand, [47] pre-processes the network in order to reduce the state-space, making the problem more manageable for other algorithms. Overall, [47] addresses a slightly different problem than we do, since the paper is concerned with feeding a VNE algorithm (such as ours) with hints for solving the VNE, and both could be used in conjunction. On the other hand, [46] is concerned with the VNE, and although it has good results, the runtime is a significant problem as it is exponential in the number of nodes. The authors patch this issue with the use of a GPU. However, our experiments show that although the runtime is manageable, it is still higher than all other algorithms we tested (in the order of three times more).

3.2.3 Heuristics and meta-heuristic techniques

There is also a wealth of meta-heuristic algorithms for the VNE. This includes genetic algorithms [48] and ant colony optimization [49]. However the most popular class of meta-heuristic approach for the VNE is particle

swarm optimization (PSO), with several well performing algorithms such as [40], [50] or [51]. These PSO approaches work by initializing "particles" as a swarm of random solutions which move in the space of candidate solutions. They find new solutions by oportunistically combining the best solutions found so far with current solutions.

Regarding heuristics, in [52], authors propose a metric for evaluating a nodes' resource capacity/demand and then match highly demanding virtual nodes to highly available physical nodes. A similar idea is used in [32] where it is combined with the Pagerank algorithm for ranking nodes.

These heuristic and meta-heuristic approaches show relatively good performances that we aim to beat in this chapter. Especially, to our knowledge, none of them exploits the fact that solutions can be improved by keeping virtual nodes close to one another. In that regard, our work could inspire enhanced versions of the cited algorithms.

3.2.4 Reinforcement learning approaches

The family of approaches that interests us the most is reinforcement learning. First of all [36] showed how to use the Monte Carlo Tree Search algorithm (MCTS) [39] for the VNE problem. MCTS intelligently explores the space of possible placement solutions in order to find the best, but its exploration is based on multi-armed bandit theory, which assumes stochastic rewards. Instead, the outcome of a given embedding is deterministic and our method more effectively exploits determinism. Both can be considered *online* methods, since they can immediately take decisions on any CNS arrivals.

By contrast, *offline methods* accumulate knowledge during an extensive learning (training) stage, which is then reused for a near-instantaneous high-quality embedding. Recently, DeepVine [38] used a deep neural network in order to learn embedding. This approach learns from graphs that are turned into images, enabling easy use of convolutional neural network (CNN) architectures. Although successful, this method makes strong assumptions about the input graphs: CNNs rely on the networks to be grid-shaped. Another method is [37] where the neural neural network is fed directly with graphs. In this article, they use the A3C (Asynchronous Advantage Actor-Critic) algorithm for learning, which has been successful for other RL tasks. These approaches rely on function approximators (namely, neural networks) coupled with model-free RL techniques. This use of neural networks enables them to deal with big state-spaces, but comes at the cost of having no convergence guarantees to an optimal embedding or even an approximation. On the other hand, online methods like ours can be tweaked to guarantee that given enough time, they could converge to the optimal solution. They are also able to handle similar state-spaces com-

Notation	Description
$\mathcal{G}(\mathcal{V}, \mathcal{E})$	Physical network with nodes V and links E
$\mathcal{H}^x(\mathcal{V}^x, \mathcal{E}^x, t_a^x, t_d^x)$	x^{th} CNS with nodes V^x, links $\mathcal{E}^x$, arrival and departure dates t_a^x and t_d^x
CPU_{v_i}	CPU capacity of physical node v_i
BW_{v_i,v_j}	BW capacity of physical link (v_i, v_j)
$CPU^o_{v_i}$	Occupied CPU of physical node v_i
$BW^o_{v_i,v_j}$	Occupied BW of physical link (v_i, v_j)
$CPU^d_{v_j^x}$	CPU demanded by virtual node v_j^x
$BW^d_{v_i^x,v_j^x}$	BW demanded by virtual link (v_i^x, v_j^x)
$BW^x_{v_i,v_j}$	Bandwidth used by CNS x on physical link (v_i, v_j)
$CPU^x_{v_j}$	CPU used by CNS x on physical node v_j
$\mathcal{A}$	Set of possible actions in MDP
$s(k)$	State of MDP at step k
a_k	Action chosen in MDP at step k
P	Policy function (associates a State-action couple with its weight)
$\mathcal{P}_m$	Path mapping (set of physical paths, each one associated with a virtual edge)

Table 3.1: Notation

pared to neural-network based methods.

The huge computation needed to perform a very costly a-priori training (*e.g.*, training for 72h on 24 for parallel instances of the problem [37]) may make these offline methods [37] [38] infeasible in practical situations. In particular when applying embedding on different scenarios (or with different conditions or constraints), the huge offline learning phase must start from scratch. It is also an open question whether or not in a real world scenario we will have enough samples in order to enable such algorithms to learn. The advantage of online methods is instead their ability to immediately adapt and take decisions on new instances of the problem.

For these reasons we improve upon the state-of-the-art online methods [36] [7], providing convergence at regime toward the optimal embedding, sample efficiency and better empirical performance.

3.3 VNE model

The physical network belongs to an operator. At any point in time, the operator has a full knowledge of the state of the network, i.e. the amount of resources available, the CNSs it hosts and the resources they use. The operator receives CNS requests from its clients over time. These requests are descriptions of a virtual network they would like to embed on the network, including resources required and topology. The goal of the operator is to place the incoming CNSs on its network in order to maximize a given objective (CNS acceptance rate in our case).

3.3.1 Graph theoretic notation

The VNE problem can be formally described as a graph embedding problem:

- the physical network is represented as an undirected graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of n physical nodes, $v_1, \ldots, v_i, \ldots, v_n$, that represent several physical machines where virtual network functions can be hosted, and $\mathcal{E}$ is the set of the physical edges between the nodes. So we have:

  - Each physical node v_i is characterized by a CPU capacity, CPU_{v_i} and an occupied CPU quantity, $CPU^o_{v_i}$. Without loss of generality, one could extend this model by considering a vector of different resources instead of a single value. For example the first component of the vector could be CPU and the second one could be RAM.
  - On the other hand, each physical edge $(v_i, v_j) \in \mathcal{E}$ is weighted by a maximum bandwidth amount, BW_{v_i,v_j} and an occupied bandwidth amount $BW^o_{v_i,v_j}$. In case $BW_{v_i,v_j} = 0$, then we consider that there is no edge between v_i and v_j.

- We denote by $\mathcal{H}^x(\mathcal{V}^x, \mathcal{E}^x, t^x_a, t^x_d)$ the undirected graph describing the x^{th} CNS with the resources needed:

  - Each virtual node of the CNS, $v^x_i \in \mathcal{V}^x$ carries a CPU demand, $CPU^d_{v^x_i}$
  - Each virtual link $(v^x_i, v^x_j) \in \mathcal{E}^x$ carries a bandwidth demand, $BW^d_{v^x_i,v^x_j}$
  - Since we are in a dynamical system, each CNS also has a time of arrival t^x_a and a time of departure t^x_d.

  Observe that as CNSs are placed or leaving, the physical occupied resources, $CPU^o_{v_i}$ and $BW^o_{v_i,v_j}$ change over time. The problem is to map each virtual node on a physical node and each virtual link on a physical path between the two hosts of its extremities, taking into account the available resources. We do not consider delays in this chapter. However, it would totally be feasible to extend our results to a model where each physical edge has a maximum delay and where each virtual edge requires a maximum delay to respect. We discuss in section 4.2 the changes required to make our solution work in this case.

3.3.2 Problem constraints

If at a certain instant time instant the x^{th} CNS request arrives, placement decisions must satisfy the following constraints:

- Each placed virtual node should have enough CPU, e.g. if we choose v_i hosts virtual node v_j^x we should have $CPU^d_{v_j^x} \leq CPU_{v_i} - CPU^o_{v_i}$ (where $CPU_{v_i} - CPU^o_{v_i}$ represents available CPU on node v_i)

- For virtual link (v_m^x, v_p^x) all physical links (v_i, v_j) it uses should be chosen so $BW^d_{v_m^x, v_p^x} \leq BW_{v_i,v_j} - BW^o_{v_i,v_j}$ (where $BW_{v_i,v_j} - BW^o_{v_i,v_j}$ represents the available bandwidth between nodes v_i and v_j) such that these links form a path between the physical nodes hosting v_m^x and v_p^x).

- If two virtual nodes belong to the same CNS, they can't be placed on the same physical node. This constraint is present in most previous works on the VNE[36, 31, 32]. It ensures reliability by preventing a significant portion of a CNS from going off if a single physical node is down. To our knowledge, the optimal trade-off between sharing physical nodes (thus economizing bandwidth) and redundancy has not been well studied. Our approach, as most of the others cited, could work with no change for a relaxation of this constraint. For example, if we had to tackle the Virtual Network Function (VNF) placement problem [53], we could apply our algorithm by relaxing this last constraint and adding a new one which would limit the node placement possibilities based on node types (*e.g.* for example physical UPFs could only host virtual UPFs). On the other hand, in NFV, it can happen that two functions must be placed on the same physical node. Since those two functions would always be placed on the same node, this case can be modeled by aggregating the two functions as a single virtual node, requiring the sum of the CPU of the two functions. The virtual links of the new aggregated virtual node would be the virtual links of both the original virtual nodes.

3.3.3 Online VNE description

An example of a CNS is shown in Figure 3.1. We solve the VNE online :

- when a CNS x arrives at time t_a^x, we directly try to embed it. If a feasible solution is found, the CNS is placed on the physical network, consuming the corresponding CPU and bandwidth resources, *i.e.* updating the corresponding $CPU^o_{v_i}$ and $BW^o_{v_i,v_j}$. If no solution is found, the CNS leaves the system and is dropped.

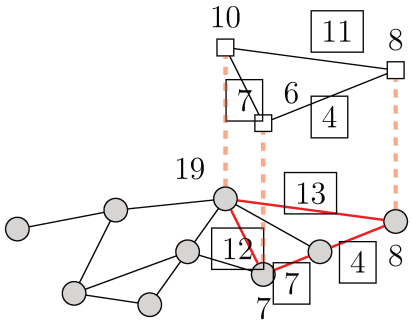


Figure 3.1: CNS (white nodes) embedded on physical network (gray nodes). Link demands and remaining capacities are boxed, used physical links are in red. CPU demands and capacities are non-boxed.

- When time t_d^x is reached, the CNS leaves the physical network and resources are freed.

The full system time is continuous and gives us the arrival and departure dates for CNSs (t_a^x and t_d^x refer to this scale) we assume the VNE is instantaneous: in the same instant the CNS request arrives, and the corresponding placement problem is solved, instantaneously, and the CNS is either placed or discarded.

3.4 VNE as a Markov-Decision Process (MDP)

For each virtual node to be placed, we select a physical node via RL (Reinforcement learning). In order to learn an optimized sequence of decisions for virtual resource placement via RL, one needs to frame the VNE problem as a Markov Decision Process (MDP)[54].

A MDP is a system made up of two elements: the agent (the network operator in our case) and the environment (the description of the CNS to place and of the state of the physical network in our case *i.e.* the amount of resources available and occupied).

3.4.1 MDP description

Observe that our MDP works as a sequence of steps, each step corresponding to the decision of placing a virtual node onto a physical node. Note that these steps do not have any time-dimension, they can be considered to be all taken instantaneously. Also observe that our MDP is fully deterministic: all transitions and all rewards (which we will define later) are deterministic and computable in advance.

In our particular setting, we consider the optimization problem where we have to place a single CNS at a time. This means that as soon as

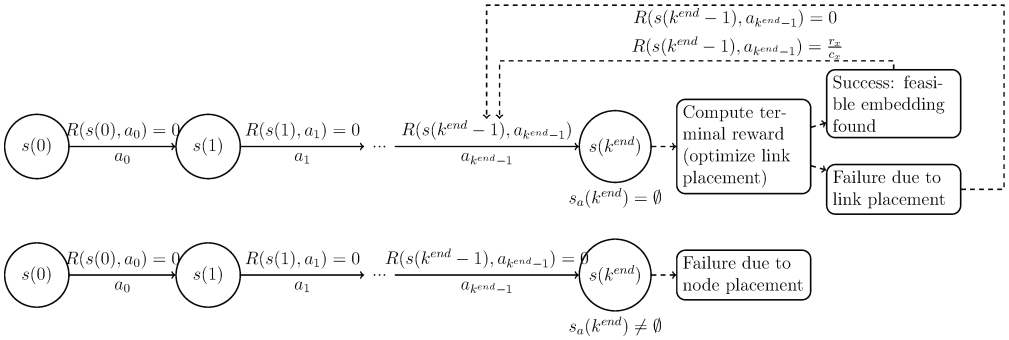


Figure 3.2: Example sequences of actions in the MDP. Dashed arrows are transitions not occurring in the MDP (no action choice). See larger version in Appendix A

one CNS requests arrives, an MDP is initialized in order to decide the embedding of each virtual node and link it demands.

We assume that the agent only decides where to place each virtual node. After all virtual nodes of a certain CNS have been placed, we calculate link placement with a shortest path heuristic (see algorithm 5). Therefore, we adopt MDP only for virtual node placement.

3.4.1.1 Elements of the MDP

Let $s(k) = (s_a(k), s_b(k))$ be a state of the MDP, it is composed of two components :

- $s_a(k)$ is the set of virtual nodes yet to be embedded at step k.
- $s_b(k)$ represents, at step k, the occupation of the physical nodes by the virtual nodes. It is a vector with $|\mathcal{V}|$ elements where $s_b(k)[i] = j$ if virtual node v_j^x from CNS x is hosted on physical node v_i. If v_i hosts no node from the current CNS, $s_b(k)[i] = 0$ (we assume indexes of virtual nodes are strictly positive integers).

For the incoming CNS x, we consider the virtual nodes $v_j^x \in V^x$ one by one. The order in which we iterate through virtual nodes can be chosen arbitrarily[1] and we take an action i which corresponds to placing it on a physical node v_i. Therefore, the set of possible actions $\mathcal{A} = \{1, \ldots, n\}$ corresponds to the physical nodes of $\mathcal{V}$. Choosing action i would mean

[1]Arbitrarily here means any order could be chosen and there would always be a resulting valid MDP. However some orders might be more suitable than others for resolution. We discuss the node ordering we choose during resolution at the end of section 4.6

placing the current virtual node on v_i. We also consider $\mathcal{A}(s(k)) \subseteq \mathcal{A}$ the set of legal actions from state $s(k)$, which will be specified later.

3.4.2 System's evolution

The main steps of the system evolution are described as follows, as well as in Figure 3.2:

a. At step 0, $s(0) = (\mathcal{V}^x, u)$, where u is a vector of $|\mathcal{V}|$ components all equal to 0.

b. At step $k \geq 0$, from the state $s(k)$, let v_l^x be the first virtual node of $s_a(k)$. Then $\mathcal{A}(s(k))$ is the set of actions $j \in \mathcal{A}$ such that $CPU^d_{v_l^x} \leq CPU_{v_j} - CPU^o_{v_j}$ and $s_b(k)[j] = 0$.
Assume the chosen action from $\mathcal{A}(s(k))$ is $a_k = i$. Then the virtual node v_l^x is embedded on physical node v_i and we have a transition to the state $s(k+1) = (s_a(k) - \{v_l^x\}, s_b(k) + b_i)$ where b_i is a vector with the i^{th} component equal to index l of virtual node v_l^x and all other components equal to 0.

The embedding process continues at each step until we reach the final state at a certain step k^{end}, where $\mathcal{A}(s(k^{end})) = \emptyset$. At this point, two situations can occur:

- Either the node embedding is a success, so the set of virtual nodes is $s_a(k^{\text{end}}) = \emptyset$. The second part of the state holds a vector $s_b(k^{\text{end}})$ indicating which physical nodes are used by each virtual node of the CNS. So the final state is $(\emptyset, u')$, where $u'[i] = l$ if virtual node v_l^x is hosted by physical node v_i .
- Or the embedding fails, which means that for a virtual node, there is no suitable physical node to host it i.e. $s_a(k^{end}) \neq \emptyset$. In this case, the entire CNS is rejected.

If node embedding is successful, the link embedding is calculated using algorithm 5 which is a shortest path heuristic. Then, if link embedding is successful too, we need to update the physical network to acknowledge for the used resources, *i.e.* update $CPU^o_{v_i}$ and $BW^o_{v_i,v_j}$ for all physical nodes v_i and physical links v_i, v_j used by the CNS. On the other hand, if one of the two phases fails, the CNS is discarded.

3.4.2.1 Reward Function

We now define the reward obtained by the agent over the course of its actions. Let us first define the revenue of the operator r^x (representing

the revenue gained thanks to a client paying for CNS x) and the cost c^x (the cost induced by operating the physical resources allocated to host the CNS) for a successfully placed CNS x as:

$$r^x = \sum_{\forall v_i^x, v_j^x \in \mathcal{V}^x} BW^d_{v_i^x, v_j^x} + \sum_{\forall v_m^x \in \mathcal{V}^x} CPU^d_{v_m^x} \tag{3.1}$$

$$c^x = \sum_{\forall (v_i, v_j) \in \mathcal{E}} \bar{BW}^x_{v_i, v_j} + \sum_{\forall v_i \in V} \bar{CPU}^x_{v_i}, \tag{3.2}$$

where for CNS x, $\bar{BW}^x_{v_i,v_j}$ is the bandwidth used on physical link (v_i, v_j) and $\bar{CPU}^x_{v_i}$ the CPU used on physical node v_i. In other words, service providers pay proportionally to the resource demands by their CNSs. The cost of operation of a CNS is proportional to the physical resources consumed. We define the immediate reward function of our MDP as:

$$R(s(k), a_k) = \begin{cases} \frac{r^x}{c^x} & \text{if } s_a(k+1) = \emptyset \text{ and node and link} \\ & \text{mapping are successful} \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

Examples of sequences of actions in the MDP are shown in figure 3.2, in which the circular states correspond to the states of the MDP, where the decisions are taken by the agent. The sequence at the top diagram corresponds to a successful embedding (after node and link placement), while the bottom one returns a failure. Note that since rewards happens during transitions, the last reward is $R(s(k^{end}-1), a_{k^{end}-1})$ as it happens during the last transition, from $s(k^{end}-1)$ to $s(k^{end})$.

3.4.2.2 Objective function

From the initial state $s(0)$, we consider a sequence of k^{end} actions : $seq = a_0$, a_1, ...,$a_{k^{end}-1}$. We define the total reward from the state $s(0)$ for seq as follows :

$$R^{seq}(s(0)) = \sum_{k=0}^{k^{end}-1} R(s(k), a_k) \tag{3.4}$$

Then the objective function is:

$$\max_{seq} R^{seq}(s(0)) \tag{3.5}$$

And the agent seeks to find the best sequence of actions:

$$seq^* = \arg\max_{seq} R^{seq}(s(0)) \tag{3.6}$$

and the corresponding reward:

$$R^*(s(0)) = R^{seq^*}(s(0)) \tag{3.7}$$

Notice that in practice, all rewards except the last one are equal to 0 due to equation (3). With this definition of reward, the agent, i.e., the network operator always tries to choose valid embeddings, since any valid embedding has a non-zero revenue-to-cost ratio. It also favors embeddings that use the least possible amount of resources, since the reward increases as $\sum_{\forall(v_i,v_j)\in\mathcal{E}} B\bar{W}^x_{v_i,v_j}$ decreases. An intuitive way to frame this is that the reward encourages the choice of embeddings that lead to placing virtual links on short physical paths, effectively trying to place the CNS on a cluster of physical nodes. We do this based on the idea that if a CNS uses the least possible amount of resources, then it will leave more resources available for future CNSs, thus enabling us to improve the acceptance ratio on the full scenario. Note that, at best, each virtual link is mapped on a physical link of length 1. Note also that for a successfully embedded CNS, $\sum_{v_i\in\mathcal{V}} C\bar{P}U^x_{v_i} = \sum_{v^x_m\in\mathcal{V}^x} CPU^d_{v^x_m}$, hence the best achievable reward is 1 and, the closer the reward is to 0, the worse the embedding is in terms of resource usage (with 0 being the worst reward, reserved for failed embeddings). Therefore, this reward function quantifies the quality of an embedding regardless of the size of the CNS. This has clear advantages over the reward function used in [36] which is $r'_x - c_x$ with $r'_x = \alpha \sum_{\forall v^x_i, v^x_j \in V^x} BW^d_{v^x_i,v^x_j} + \beta \sum_{\forall v^x_m\in\mathcal{V}^x} CPU^d_{v^x_m}$, where α, β are weight parameters which have to be tweaked. In [36] they use parameters of 1 which provides an upper bound of 0 and no lower bound, making it harder to compare the quality of embeddings for different CNS sizes. In the general case they do not provide any bound. This is particularly unfortunate for the MCTS algorithm they use, as it is based on the upper confidence bounds algorithm UCB-1, which provides its theoretical guarantees only for a reward bounded between 0 and 1.

3.4.3 Characteristics of the MDP and implications on resolution

Since the MDP transition model for a given CNS is completely known in advance and deterministic, one could be tempted to use a method such as dynamic programming to solve the problem. However, it would be unrealistic due to the number of states: there are $\frac{|\mathcal{V}|!}{(|\mathcal{V}|-|\mathcal{V}^x|)!}$ final states (which corresponds to the number of possible repetition-free permutations of $|\mathcal{V}|$ physical nodes of size $|\mathcal{V}^x|$), each requiring to calculate link placement. For a CNS of size 12 placed on a 50 nodes network , we have 5×10^{19} possible terminal states.
Also note virtual nodes are taken in an arbitrary order, hence a given final placement is reachable only using a single sequence of actions. This implies the MDP has a tree topology (see Figure 3.3 which illustrates the full tree

of states for a toy example placement). We argue our algorithm should take this structure into account for exploration and exploitation. Particularly, we will see that existing MCTS methods (MaVEN-S from [36] and NRPA) are interesting since they take advantage of the MDP's tree structure for finding good solutions. However this can lead to local optima once the algorithm has converged. The main motivation of our work is to escape these optima by "jumping" to unexplored branches of the tree that we can guarantee are better than the best solutions found. We will show this can be done by getting around the tree topology and sometimes exploring the solution space in a different manner.
Next, we present our online learning algorithm (NEPA) which improves CNS acceptance ratio, with reduced computation time by implementing this idea. As our result section will show, we only need to explore a few hundred complete sequences of actions for our algorithm.

3.5 Contributions for online VNE

3.5.1 NRPA and NEPA

The algorithm we propose in this paper is called NEPA. It is based on NRPA, adding weight initialization and neighborhood-search based refinement. For the sake of clarity, instead of directly presenting NEPA, we first present NRPA and weight initialization.

3.5.1.1 Review of Nested Rollout Policy Adaptation (NRPA)

The NRPA[56] algorithm is a Monte Carlo Search algorithm that aims at finding near-optimal solutions in deterministic environments. It is perfectly suited for our problem as in our model a given action from a certain state always leads deterministically to the same state. This setup is similar to the puzzle games which NRPA solves remarkably well (with a world record for Morpion Solitaire) [56]. We describe NRPA in Algorithm 2. The idea of this algorithm is to consider the MDP as a tree that we have to explore ("search tree"). This is coherent with our model because we treat virtual nodes to place in an ordered manner, hence there is only a single way to reach a given final or inetrmediate state. NRPA explores the search tree with recursive calls to the search function, where l is the level (or the depth) of the search and N is the number of of recursive calls per level. This search function is defined as such:

- At level 0 a search call does a random simulation of legal actions in the MDP. It returns the reward obtained during that run of the MDP along with the sequence of actions used and the virtual link placement solution (*i.e.* the path mapping $\mathcal{P}_m$, calculated using Algorithm 5).

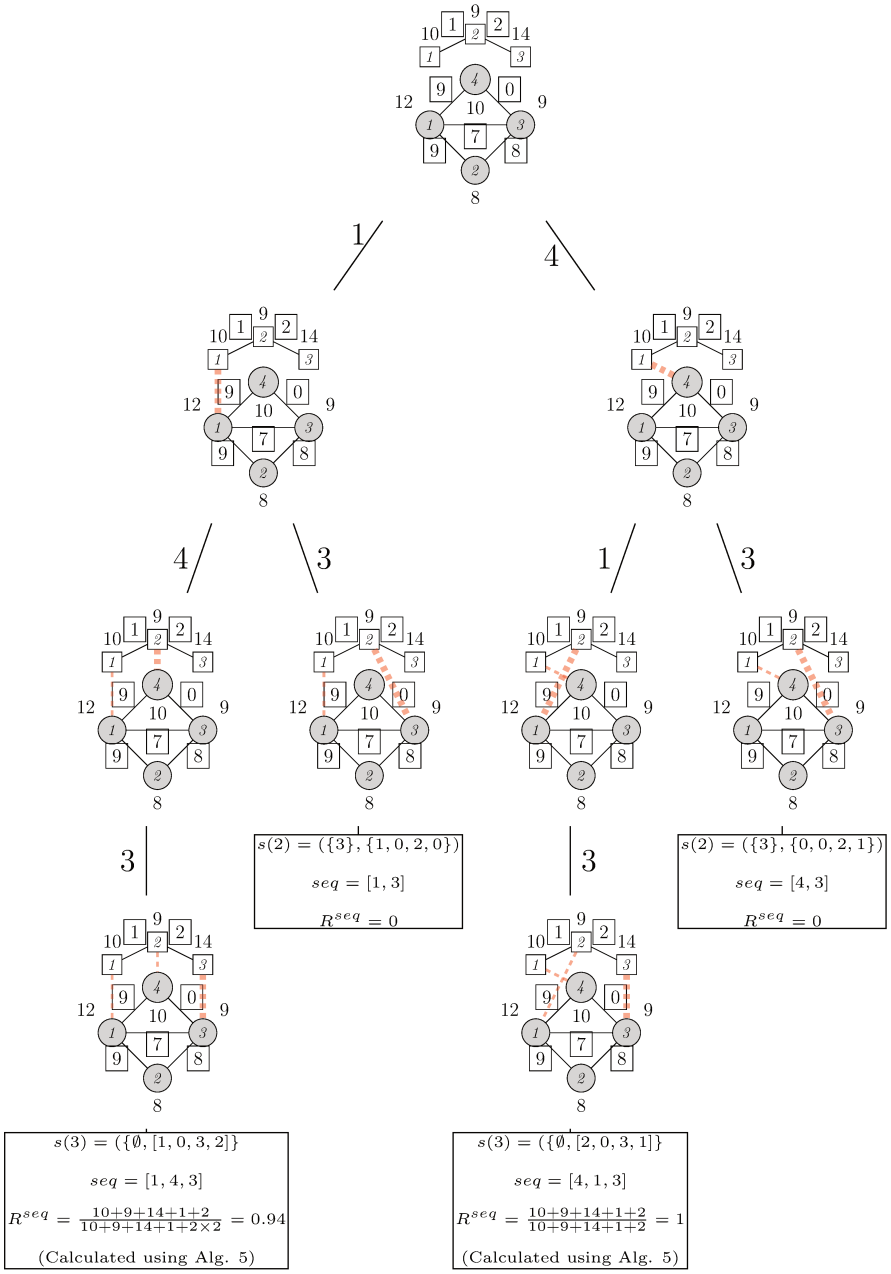


Figure 3.3: Example MDP for a toy example. Observe transitions are deterministic and MDP has a tree topology.

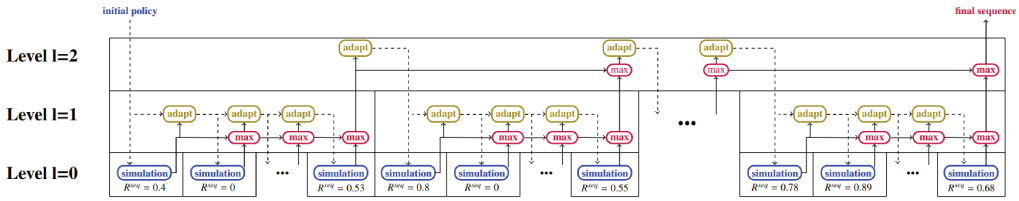


Figure 3.4: Example execution of NRPA for l=2. Dashed arrows represent the policy going from one function to the other, while plain arrows represent sequences returned between functions. A function needs to get values from all its predecessor before executing. See Appendix A for large version. Figure reproduced from [55].

If we refer to figure 3.3, we can see a random simulation as a complete descent from the root of the tree to a leaf (*i.e.* final) state. The return values from that descent are the corresponding seq and R^{seq}.

- At level $l \neq 0$ the algorithm makes N NRPA calls of level $l-1$. It then returns the best sequence returned by these "children" calls to its caller function, which is either a level $l+1$ NRPA call or the main function. In the latter case the returned sequence is the best sequence found over every simulations tried so far (called seq^{best}) and the NRPA algorithm terminates.

- Then, the control flow returns to the main function (algorithm 1), which updates the resources.

The NRPA search function is combined with a policy learning procedure (see Algorithm 3 : Adapt procedure for NRPA).

3.5.1.1.1 Policy improvement The principle is that during each simulation, we choose the sequence of actions seq in a biased manner that leads to final states close to the best final state found so far, $R^{seq^{best}}$, which has been reached through sequence of actions seq^{best}. This random sampling enables us to focus on sequences of actions that resemble seq^{best}.

We shall now give the details through which we learn and then bias the simulations :

- We define a policy matrix, P which associates each possible tuple $(s(k), a_k)$ with a real weight $P[s(k), a_k]$ from which probabilities are calculated during simulation.

- Given a certain initial state $s(0) = (s_a(0), s_b(0))$ and a policy matrix P, the algorithm will try a sequence of random actions dictated by the probabilities calculated from P.

After each NRPA call, the weights of actions of the best sequence found seq^{best} are incremented with respect to the state where they should be chosen, *i.e.* $P[s(i), a_i]$ is incremented for all $a_i \in seq^{best}$ (see Algorithm 3). Then, during the simulation, when we are in state $s(k)$ and need to select the action $a_k = i$ randomly, we draw using Gibbs sampling, i.e. with probability $\frac{\exp P[s(k),i]}{\exp \sum_{1 \leq j \leq |\mathcal{A}|} P[s(k),j]}$. A visualization of those steps is depicted in figure 3.4, where the recursive nature of the algorithm is particularly noticeable.

3.5.1.2 Virtual links placement

Algorithm 5, is used for placing virtual links after the node placement is decided. It is used during each call to the simulation procedure (Algorithm 4). The idea is to treat virtual links one by one by descending bandwidth demands, embedding them on the shortest path (in terms of hops) that has enough bandwidth. Note this is not an exact algorithm and it could be replaced with other methods of link embedding. We do not use an exact method because the underlying problem of placing virtual links is an instance of the unsplittable flow problem which is itself NP-Hard [57]. One alternative could be to relax the problem and allow "path-splitting", making the problem solvable by linear programming [58]. It might be of interest and has been used for the VNE (see for example [36]), with the relaxed version consistently improving performance metrics at the cost of a larger computation time (in the order of 40 times for their small cases). However it is unclear whether such an algorithm would be implementable in practice, due to scalability issues as well as the need to reorder packets on arrival, incurring potential additional delay and CPU processing times. For these reasons, the case of path-splitting is outside the scope of this chapter. Also, we remark our link placement algorithm can be easily adapted to the case of VNE with delay constraints. In such a case, we would need, for each virtual link, to find the shortest path (in terms of hops) which respects the delay constraint from the virtual link. For this purpose we could use a constrained shortest path algorithm such as the state-of-the-art WC-EBBA* algorithm from Ahmadi *et al.* [59]. While this approach is fast in practice, in theory it is non-polynomial which might hinder the complexity of link placement. However we note since we look for a shortest path in terms of hops, it corresponds to a case where all links are of the same length. In this setting, Johnson and Garey [60] argue the problem is polynomial but we were unable to find a reference for an algorithm. Furthermore, we note one could use any of the edge weighting approaches of the literature such as the ones tailored for optical networks from Zhang *et al.* [61]. If this was required, we would replace our hop-based shortest path algorithm with a method that takes weights into account such as

Dijkstra's.

Both those extensions would slow down our algorithm due to the added complexity, however we note this would be the case for all other approaches compared when extended to these cases in the same way.

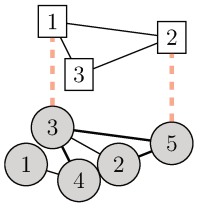
To conclude with $NRPA$, we give in Algorithm.1 the main procedure which describes the calls of the different algorithms related to NRPA for a CNS placement.

3.5.1.3 Heuristic weight initialization

In standard NRPA, when one encounters an unseen state $s(k)$, all its potential following states $s'(k+1)$, reached from $s(k)$ by choosing action a_k are initialized with a weight $P[s(k), a_k] = 0$. However, this leads to exploring completely at random without exploiting knowledge of the problem. We propose to bias the weight initialization towards more interesting actions, drawing inspiration from [42]. Our heuristic for weight initialization (which is called the first time we encounter a given state, *i.e.* in the simulation function) assumes that good embeddings tend to cluster virtual nodes, *i.e.* to place virtual nodes of the same CNS in close-by physical nodes, which reduces the mean length of the virtual links. When a new state-action couple $(s(k), a_k)$ is encountered, we initialize its weight with:

$$P[s(k), a_k] = \begin{cases} -\sum\limits_{1 \leq i \leq n} \frac{d(i,a_k) \times \mathbb{1}(s_b(k)[i])}{\sum\limits_{1 \leq j \leq n} \mathbb{1}(s_b(k)[j])}, & if \quad s_b(k) \neq \vec{0} \\ \frac{1}{n} & otherwise \end{cases} \tag{3.8}$$

where $\mathbb{1}(s_b(k)[i])$ is equal to 1 if $s_b(k)[i]$ is non-zero (meaning that some virtual node from current CNS is associated to physical node v_i) and zero otherwise. The function $d(i, j)$ returns the distance (in terms of hops) between physical node i and j. Note this distance does not take bandwidth into account, making the function computable in advance before starting NRPA, which makes the complexity of weight initialization negligible. In other words we penalise the physical nodes that are far from the ones used up to the current state to embed the current CNS. Figure 3.5 gives an example of such an initialization when the NRPA algorithm first encounters the state ({3}, [0, 0, 1, 0, 2])}. Note that candidate physical nodes that are close to previously placed virtual nodes get a higher weight, since we assume they tend to be more interesting choices. We show an example of such a weight initialization in Figure 3.5. Notice that the highest weight corresponds to placing virtual node 3 on physical node 2, which leads to using the least resources.



Next action	Potential next state	Initial weight value
$a_k = 4$	({}, [0, 0, 1, **3**, 2])	$-\frac{d(3,4)+d(5,4)}{2} = -1.5$
$a_k = 2$	({}, [0, **3**, 1, 0, 2])	$-\frac{d(3,2)+d(5,2)}{2} = -1$
$a_k = 1$	({}, [**3**, 0, 1, 0, 2])	$-\frac{d(3,1)+d(5,1)}{2} = -2.5$

Figure 3.5: Example of state weight heuristic initialization when first choosing action from state ({3}, [0, 0, 1, 0, 2])

Algorithm 1 MAIN placement procedure

Input: $\mathcal{G}(\mathcal{V}, \mathcal{E})$: Physical network, $\mathcal{H}^x(\mathcal{V}^x, \mathcal{E}^x)$: CNS to place

Output: $\mathcal{G}(\mathcal{V}, \mathcal{E})$: Physical network, seq^{best} : best node placement, $\boldsymbol{\mathcal{P}}_m^{best}$: link mapping corresponding to seq^{best}

Choose level parameter l (and level l' for NEPA) and number of iterations per level N
Initialize policy P as all 0s
Derive initial state s(0) from $\mathcal{G}$ and $\mathcal{H}^x$
if we call NRPA **then**
    $R^{seq^{best}}, seq^{best}, \boldsymbol{\mathcal{P}}_m^{best} \leftarrow$ NRPA(l, N, P, s(0), $\mathcal{E}^x, \mathcal{G}$)
end if
if we call NEPA **then**
    $R^{seq^{best}}, seq^{best}, \boldsymbol{\mathcal{P}}_m^{best} \leftarrow$ NEPA(l, N, P, s(0), $\mathcal{E}^x, \mathcal{G}$, l')
end if
if $R^{seq^{best}} \geq 0$ **then**
    Update occupied resources of $\mathcal{G}$ with the placed CNS
end if
return $\mathcal{G}$, $seq^{best}, \boldsymbol{\mathcal{P}}_m^{best}$

Algorithm 2 NRPA Algorithm

Input: l: Search level , N: max iterations, P : Policy, $s(0)$: Initial state, $\mathcal{E}^x$: Virtual links, $\mathcal{G}(\mathcal{V}, \mathcal{E})$: Physical network

Output: $R^{seq^{best}}$: Best score, seq^{best}: best sequence of actions to achieve it, $\boldsymbol{\mathcal{P}}_m^{best}$: link mapping corresponding to seq

if $l = 0$ **then**
  return SIMULATION($s(0)$, P, $\mathcal{E}^x$, $\mathcal{G}$)
end if
$R^{seq^{best}} \leftarrow -\infty$
$seq^{best} \leftarrow \emptyset$
$\boldsymbol{\mathcal{P}}_m^{best} \leftarrow \emptyset$
for N iterations **do**
  $R^{seq}, seq, \boldsymbol{\mathcal{P}}_m \leftarrow$ NRPA$(l-1, N, P, s(0), \mathcal{E}^x, \mathcal{G})$
  if $R^{seq^{best}} \leq R^{seq}$ **then**
    $R^{seq^{best}} \leftarrow R^{seq}$
    $seq^{best} \leftarrow seq$
    $\boldsymbol{\mathcal{P}}_m^{best} \leftarrow \boldsymbol{\mathcal{P}}_m$
  end if
  $P \leftarrow$ ADAPT(P, seq^{best})
end for
return $R^{seq^{best}}, seq^{best}, \boldsymbol{\mathcal{P}}_m^{best}$

Algorithm 3 ADAPT procedure for NRPA

Input: P: Policy matrix, seq : sequence of actions

Output: Update of P biased towards drawing actions from seq

$P_{new} \leftarrow P$
for $k = \{0, ..., |seq - 1|\}$ **do**
  $v_l^x \leftarrow$ first node of $s_a(k)$
  $P_{new}[s(k), a_k]$ += 1 // a_k is the k^{th} action of seq
  for $m \in \mathcal{A}(s(k))$ **do**
    P_{new}[s(k), m] −= $\exp(\frac{P[s(k),m]}{\sum_{j \in \mathcal{A}(s(k))} \exp(P[s(k),j])})$
  end for
  s(k+1) $\leftarrow (s_a(k) - \{v_l^x\}, s_b(k) + b_{a_k})$
end for
return P_{new}

Algorithm 4 SIMULATION procedure

Input: $s(0)$: Initial State, P: Policy matrix, $\mathcal{E}^x$: Set of virtual links, $\mathcal{G}$: Physical network

Output: seq: sequence of actions, R^{seq}: the reward it yielded, $\boldsymbol{\mathcal{P}}_m$: path mapping found by *Alg.* 5 for seq

$seq \leftarrow \emptyset,\ k \leftarrow 0$
while $\mathcal{A}(s(k)) \neq \emptyset$ **do**
    $v_l^x \leftarrow$ first node of $s_a(k)$
    Deduce $\mathcal{A}(s(k))$ for v_l^x
    $a_k \leftarrow$ random-choice($\mathcal{A}(s(k))$)
    // *draws action from* $\mathcal{A}(s(k))$ *with probability* $\frac{\exp(P[s(k),a_k])}{\sum_{j \in \mathcal{A}(s(k))} \exp(P[s(k),j])}$
    $s(k+1) \leftarrow (s_a(k) - \{v_l^x\}, s_b(k) + b_{a_k})$
    $seq \leftarrow seq \bigcup a_k$
    k++
end while
$R^{seq}, \boldsymbol{\mathcal{P}}_m \leftarrow VLINK(\mathcal{E}^x, seq, \mathcal{G})$
return R^{seq}, seq, $\boldsymbol{\mathcal{P}}_m$

Algorithm 5 VLINK (virtual link placement) procedure

Input: Set of virtual links $\mathcal{E}^x$, sequence of actions seq, Physical network $\mathcal{G}$

Output: R^{seq} : Reward yielded by seq, $\boldsymbol{\mathcal{P}}_m = \{\mathcal{P}_{v_i^x, v_j^x}, \forall (v_i^x, v_j^x) \in \mathcal{E}^x\}$: set of physical path used by each virtual link

while $\mathcal{E}^x \neq \emptyset$ **do**
    Pick $(v_i^x, v_j^x) \in \mathcal{E}^x$, the most demanding link.
    Find the shortest path $\mathcal{P}_{v_i^x, v_j^x}$ between the physical nodes hosting v_i^x and v_j^x, minding only physical links with available bandwidth (at least equal to $BW^d_{v_i^x, v_j^x}$)
    if $\mathcal{P}_{v_i^x, v_j^x} \neq \emptyset$ **then**
        Update the physical links occupied by $\mathcal{P}_{v_i^x, v_j^x}$
        $\boldsymbol{\mathcal{P}}_m \leftarrow \boldsymbol{\mathcal{P}}_m \bigcup \mathcal{P}_{v_i^x, v_j^x}$
    else
        return $0, \emptyset$
    end if
end while
compute R^{seq}
return $R^{seq}, \boldsymbol{\mathcal{P}}_m$

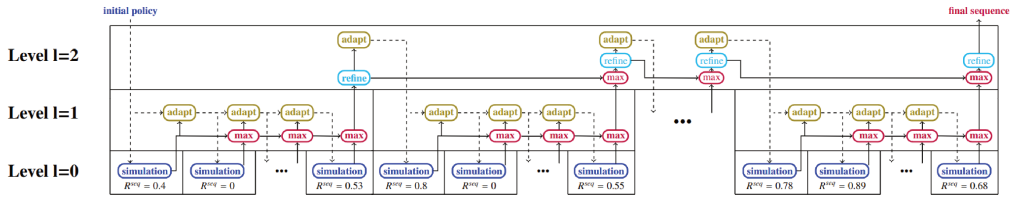


Figure 3.6: Example execution of NEPA for l=2, l'=2. Dashed arrows represent the policy going from one function to the other, while plain arrows represent sequences returned between functions. A function needs to get values from all its predecessor before executing. See Appendix A for large version. Figure inspired from [55].

3.5.1.4 Neighborhood Enhanced Policy Adaptation (NEPA)

Observe that once NRPA has found a reasonably good seq^{best}, the weights of the intermediate state leading to it (*i.e.* the weights on the path from the root of the tree to the best final state found) will start increasing. This means if a random simulation deviates from seq^{best} early on, it will then draw actions from a state which is considered almost unexplored, and where the knowledge of seq^{best} is not used. For example, in Fig. 3.3, if the best sequence found so far is [1, 4, 3] and at first step the chosen action is 4, it would be desirable to exploit the knowledge that in the best sequence found, virtual node 3 goes on physical node 3, as it is true in [1, 4, 3]. In our toy example, this would imply that the algorithm would have a higher chance to find the optimal sequence, [4, 1, 3]. However, with NRPA this is not how things go: if we descend to an unexplored part of the tree, there is no way to reuse the information gained from the known best sequence. This is the reason we introduce NEPA, which we call a *monkey business* algorithm: our goal is to improve NRPA by enabling it to use its knowledge of seq^{best} for finding better branches, similar to how monkeys jump from one branch to another. When monkeys explore the jungle, they swing from branch to branch, they go faster than if they went back down each time they want to move. Similarly, NEPA swings from branch to branch to explore the MDP while NRPA has to go down every time it wants to explore a new zone of the search space.

One possible solution for this could be to increase weights of all states resembling those found by executing seq^{best}. However, this would incur significant cost due to the factorial number of total states, potentially requiring to design approximate methods or workarounds that are outside of the scope of this paper.

Instead, we observe NRPA often finds good embeddings that would be very easy to improve upon by changing the placement of only a small subset of nodes. We also observe that improving such embeddings could

be interesting for discovering better sequences of actions that resemble seq^{best} but would not necessarily be discoverable through NRPA's weight mechanism (such as in our example above). Our key idea is that we can improve upon seq^{best} through neighborhood search, finding a new, better sequence without taking the tree structure of the MDP into account. Then, once this improved sequence has been found, we reinject it into NRPA to use as its new seq^{best}, which it can further improve.

In this section, we devise our method for discovering such sequences while keeping the computational complexity reasonable. We call the resulting algorithm Neighborhood Enhanced Policy Adaptation (NEPA) as it combines NRPA with neighborhood search for improving good solutions. The idea of NEPA is to choose a level l' of search at which the solutions should be improved. Then, when the NEPA search reaches level l', each solution found (which correspond to the best solution of each level $l'-1$ call) is refined through the neighborhood search procedure described in Algorithm 6.

We define the neighborhood of a final state $s(k^{end})$ (corresponding to an embedding solution of the virtual nodes on the physical network), as the set of final obtained by moving a virtual node to another physical nodes.

3.5.1.5 Main steps of NEPA

We describe in Algorithm 7 the main steps of NEPA algorithm. It is similar to the NRPA algorithm except that if we reach a level $l = l'$, then we choose to refine the solution by searching a neighboring solution as described in the following algorithm:

a. Algorithm 6 first finds the nodes with the largest potential improvement among the already placed virtual nodes (*i.e.* we choose a single node v_B^x to move). We find it by calculating:

$$score(v_m^x) = \frac{\sum_{v_p \in V^x} BW^d_{v_m^x, v_p^x} \cdot d(v_m^x, v_p^x)}{deg(v_m^x)} \tag{3.9}$$

for each virtual node v_m^x, where $d(v_m^x, v_p^x)$ is a function returning the length of the physical path used by virtual link (v_m^x, v_p^x) and $deg(v_m^x)$ is the degree of virtual node v_m^x. The virtual node v_B^x which maximizes this metric is considered as the most promising for improvement, since it is the one which consumes the most bandwidth compared to its number of neighbors.

b. The refining procedure is then to try several candidate physical nodes that could be better suited to host the selected virtual node v_B^x, in terms of reducing resource consumption. For each candidate, we

remap the virtual node on them (which corresponds to flipping values in the state vector), then remap its adjacent virtual links. After all candidate physical nodes have been tried, the new placement of v_B^x is then the one that leads to a maximum reward (see eq (3)).

To control the execution time of the algorithm, we introduce two parameters:

- X : is the number of times that the process is repeated, note that the process is also stopped if a full trial does not lead to any improvement. Typically a criterion can be to do no more than $|\mathcal{V}^x|$ tries. This ensures the runtime is reasonable while spending more time on larger CNSs, since they tend to be harder to place.

- K is the number of candidate physical nodes. For choosing candidates, the simplest thing would be to try all possible physical nodes. However this would lead to poor scalability. Instead, we use our weight initialisation function and define our K candidates as the K nodes with the highest distance score. For example, in Fig. 3.7, which shows a refinement iteration with $K = 2$, the two candidates for hosting virtual node 3 are physical nodes 4 and 2 because they are the two nodes that are the closest to physical nodes 3 and 5, which host the other virtual nodes.

After the refinement, the resulting placement is treated like a normal state by NEPA, *i.e.* if it is the best found so far, its weight is incremented. In this sense, NEPA (Alg. 7) maintains the structure of NRPA (Algorithm 2). Note that in practice, NRPA could potentially have found any sequence of actions (*i.e.* node placement) found by NEPA. However, the virtual link embedding corresponding could be different since NRPA places only using Algorithm 5 while in the case of refinements, NEPA uses Algorithm 6 which can find a different link embedding for the sequence than what Algorithm 5 could have found. This is particularly important since in practice, we observe that some sequences found by Algorithm 6 have no valid solution if using only Algorithm 5. Hence, when using NEPA, it is necessary to save not only the best sequence of actions, but also the link embedding result in case it needs to be restored after execution for future use. This is typically done at the end of Algorithm 6 by saving the link embedding solution in a global data structure.

NEPA requires only very little modifications to NRPA (see algorithm 7), which is particularly noticeable in figure 3.6, as it illustrates the way NEPA makes its function calls recursively for $l' = 2$. Also note how few refine calls there are compared to the number of max operations.

Our method enables us to discover better solutions that would not be easy to find once standard NRPA has converged: once NRPA has found a

local optimum, the probabilities of choosing the states of the best sequence found in NRPA will go towards 1, meaning exploration could become poor while there is no point exploiting the same region anymore. With NEPA, since we change the best sequence found so far, we open the opportunity of exploring completely new, but better parts of the search space. A single change in the first few actions can lead to discovering a whole new part of the state-space where most states are undiscovered, leading to a highly explorative phase with a very good sequence as a starting point (which we newly found through the refinement procedure). Hence NEPA exploits its neighborhood search mechanism in order to help NRPA escape local optima.

3.5.1.6 Theoretical analysis

In this section, we give a theoretical analysis of NEPA and NRPA. We start by outlining the limits of NRPA and the convergence results that make NEPA a good solution for improving it. Then, we investigate the computational and memory complexity of the algorithms.

3.5.1.6.1 Computational Complexity

Proposition 1. *The NRPA algorithm has a computational complexity of* $O(|\mathcal{V}| \times N^l)$ *for sparse physical and virtual graphs.*

Let $T(N, l)$ be the function associating the algorithms' parameters (number of iterations per level N and search level l) with the number of simulations executed (*i.e.* the number of calls to Alg. 3). We shall prove $T(N, l) = N^l$ by induction on l: For $l = 0$ the relationship is verified. We now assume that our hypothesis is verified, *i.e.* $T(N, l) = N^l$. We will now show that this implies $T(N, l + 1) = N^{l+1}$.

$$T(N, l + 1) = N \times T(N, l) = N \times N^l = N^{l+1} \tag{3.10}$$

This proves $T(N, l) = N^l$. By the same argument, one could show that the same NRPA search would perform N^l adaptations of its policy (*i.e.* N^l calls to Alg. 2).

Algorithms 3 and 4 are really similar and treat a sequence of length $|\mathcal{V}^x|$. For each element of the sequence, both algorithms loop through the list of legal moves. At worst, at each step, all physical nodes that have not already been chosen are legal. In such a case, the complexity of both nested loops would be $O(|\mathcal{V}^x| \times |\mathcal{V}|)$. In order to compute the rewards in the simulation procedure, we place virtual links of the embedding found. This is done using algorithm 5. A breadth-first search (BFS), used in Alg. 4 for finding shortest path, has a complexity of $O(|\mathcal{V}| + |\mathcal{E}|)$ which we perform

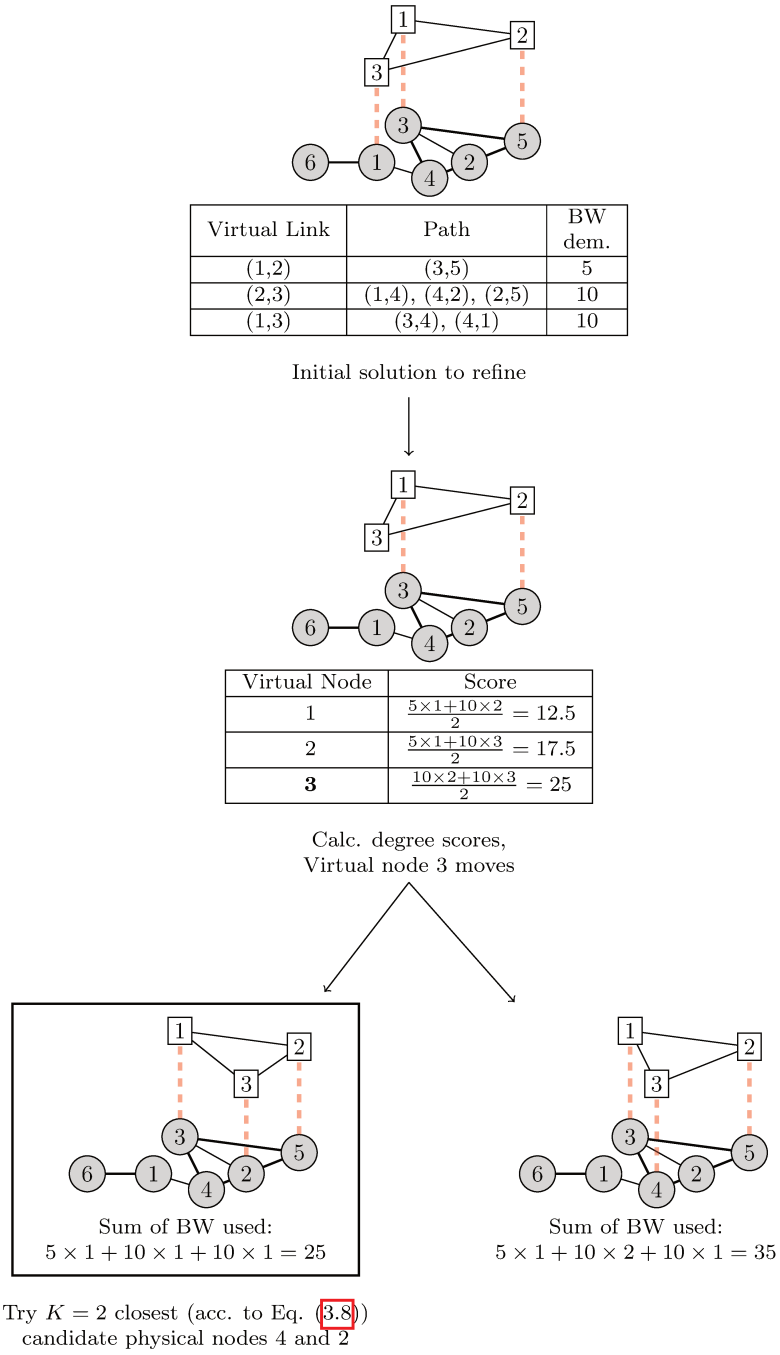


Figure 3.7: Example iteration of Refine with K=2. The placement chosen after the iteration is in the box. We assume all nodes have enough CPU and all links have enough BW. The chosen solution has the lowest amount of bandwidth used as reward only depends on it (CPU demands and uses are the same for a given CNS)

$\mathcal{E}^x$ times in Alg. 4. The complexity of the simulation procedure (Alg. 3) is then $O((|\mathcal{V}| + |\mathcal{E}|) \times |\mathcal{E}^x|)$. Furthermore, at worst we have $|\mathcal{E}| = \frac{|\mathcal{V}|(|\mathcal{V}|-1)}{2}$ and $|\mathcal{E}^x| = \frac{|\mathcal{V}^x|(|\mathcal{V}^x|-1)}{2}$, so the complexity of the link embedding phase is $O(|\mathcal{V}|^2 \times |\mathcal{V}^x|^2)$.

It can then be concluded that the complexity of the NRPA algorithm (Alg. 1) is $O(|\mathcal{E}^x| \times |\mathcal{E}| \times N^l) = O(|\mathcal{V}^x|^2 \times |\mathcal{V}|^2 \times N^l)$. Note that we used the simplest possible embedding function for links. If one swaps it out for a more elaborate function, such as an exact method[62], one taking congestion[63], delays[64] or survivability[65] into account, the complexity would typically increase. We expect such a costlier function to be used in a more realistic setting. Also note that in a typical scenario, both physical network and virtual networks are sparse, *i.e.* $|\mathcal{V}|^2 >> |\mathcal{E}|$ and $|\mathcal{V}^x|^2 >> |\mathcal{E}^x|$ hence the complexity of BFS can be assumed to be reduced to $O(|\mathcal{V}|)$ and the complexity of NRPA to be $O(|\mathcal{V}^x| \times |\mathcal{V}| \cdot N^l)$. Furthermore in most cases $|\mathcal{V}| >> |\mathcal{V}^x|$ and it further reduces to $O(|\mathcal{V}| \times N^l)$.

Proposition 2. *When X goes to infinite, the refinement Algorithm 6 has to execute at most $O(diam(\mathcal{G})^{|\mathcal{E}^x|})$ iterations of the main loop (lines 4 to 25).*

Proof. We observe that, because of lines 22–23, a new iteration can only start if the reward has improved. Hence, the maximum number of iterations the for loop of lines 4–25 can do is bounded by the number of possible values the reward function can take. Intuitively, the worst case scenario is to start with the smallest non-zero reward and to improve it by the smallest possible amount at each iteration, until we reach the maximum value of the reward.

We note the number of possible values for the reward, $\frac{r^x}{c^x}$ depends on the possible values for the bandwidth consumption, $\sum_{\forall (v_i,v_j) \in \mathcal{E}} B\bar{W}^x_{v_i,v_j}$ (as r^x and $\sum_{v_i \in \mathcal{V}} C\bar{P}U^x_{v_i}$ do not change from one solution to the other). The bandwidth consumption of a single virtual edge itself depends only on the length of the physical path it uses. This path is of minimum length 1 and its maximum length is the diameter of $\mathcal{G}$, $diam(\mathcal{G})$. Since it is the case for each paths, at most there are $diam(\mathcal{G})^{|\mathcal{E}^x|}$ possible values for the reward function. Hence, when $X = \infty$ the refinement algorithm becomes non-polynomial with respect to $|\mathcal{E}^x|$. □

For this reason, in the next proposition we analyze the complexity for bounded values of X.

Proposition 3. *The NEPA algorithm has a computational complexity of $O(|\mathcal{V}| \times (N^l + N^{l'} \times K \times X))$ when physical and virtual graphs are sparse, where K is the number of candidates per refinement, X is the maximum number of times we try to refine the solution and l' is the level where refinements are performed.*

Proof. First, note that the number of simulations does not change compared to NRPA and is still N^l. It follows that the total number of operations performed by the simulation part of the algorithm is $O(|\mathcal{V}| \times N^l)$ as in NRPA.
The complexity of NEPA is then $O(|\mathcal{V}| \times N^l + Z)$ where Z is the number of operations incurred by all the refinement steps. At each refinement step, we perform $X \times K$ BFS searches of complexity $|\mathcal{E}|$. The total number of operations performed by refinements is then $Z = O(N^{l'} \times K \times X \times |\mathcal{E}|)$. In the case of a sparse graph, this number is $Z = O(N^{l'} \times K \times X \times |\mathcal{V}|)$. The total computational cost of NEPA is then

$$O(|\mathcal{V}| \times N^l + N^{l'} \times K \times X \times |\mathcal{V}|) = O(|\mathcal{V}| \times (N^l + N^{l'} \times K \times X)) \quad (3.11)$$

Overall, NEPA has a greater theoretical complexity than NRPA. However, numerical results from Appendix C show that NEPA is far more effective than NRPA when they are given equal time. □

3.5.1.6.2 Memory Complexity

Proposition 4. *The NRPA and NEPA algorithms have a memory complexity of* $O(|\mathcal{V}^x| \times N^l)$

Proof. First, in the worst case, each simulation procedure call can lead to finding $|\mathcal{V}^x|$ new unexplored states, each of which requires to store a float representing its weight in the policy. If every state found in every simulation call is seen only once, we have to store $N^l \times |\mathcal{V}^x|$ floats since as seen in the previous proofs, we call the simulation procedure N^l times. The other source of memory consumption in NRPA is the storage of the sequences, *seq* and seq^{best}. Those are both of length $|\mathcal{V}^x|$. Since NRPA calls itself recursively, we also have to count the sequences stored by its infant calls. There are at worst l such infants since the recursive call depth is of l and there is only one call of a given level active at the same time, and once a call returns it frees the memory. Hence the memory consumption of the stored sequences is $O(l \times |\mathcal{V}^x|)$. The total memory complexity of NRPA is $O(|\mathcal{V}^x| \times N^l + l \times |\mathcal{V}^x|) = O(|\mathcal{V}^x| \times N^l)$. For NEPA the memory complexity remains the same as NRPA because the refinement procedure does not incur a significant memory usage, as it only requires memory to store the best solution (a virtual network which requires $O(|\mathcal{V}^x|)$ memory in case of a sparse graph). □

3.5.1.7 Dimensionality reduction and pre-treatment

In practice, we make a slight modification to the MDP model in order to make the NEPA search more effective. First, we note that for a given couple of virtual node v_i^x and physical node v_j, if the maximum amount of

bandwidth required by links adjacent to v_i^x exceeds the maximum available bandwidth of links adjacent to v_j, then we know one of the adjacent links of v_i^x would be impossible to embed if v_i^x was placed on v_j. Hence, we reduce the size of the action space by removing such actions before running NRPA. Similarly, if the sum of the bandwidth adjacent to v_i^x exceeds the sum of bandwidths available on links adjacent to v_j, then we know it would not be possible to place all virtual links if v_i^x was placed on v_j, hence we remove this action from the set of possible actions for placing v_i^x. Finally, before the placement, we sort the nodes according to the number of physical nodes that could host them. This draws on the idea that if a node has only few possibilities for placement, we should treat it first, otherwise there would be a high chance of blocking the possible host with another virtual node placed before. By doing this, we avoid exploring some unfeasible placements.

3.5.2 Numerical Results

In this section we extensively compare NEPA with several other methods from the state-of-the-art, demonstrating the superiority of its performance consistently on various scenarios. We first compare on synthetic physical networks generated randomly. Then, algorithms are tested with real physical networks from the topologyZoo dataset. Finally, in order to assess the performance of each tested algorithm on large problems against the theoretical optimum, we compare on a set of Perfectly Solvable Scenarios [66], which are constructed so the optimal is known but is very hard to achieve. This step is often overlooked in the literature but we argue it is of key importance in order to assess the quality of each algorithm. Note we make sure the range of CPU and Bandwidth capacities fit reality: for CPUs, a typical server CPU (such as intel Xeon) would have between 8 and 56 cores (for example Xeon Platinum 9282). Also note that some server motherboards can host 2 CPUs (for example ASUS WS C621E). For ethernet links, it is common to find bandwidths in the order of 50-100 Gbps, see for example [67].

3.5.2.1 Compared methods

All our experiments are run with an Ubuntu machine with a 16-core Intel Xeon Gold 5222s machine with 32 GB RAM, except for GraphVine which requires to be executed on another machine equipped with a GPU (see below). We compare our proposed method NEPA with the following methods:

- MaVEN-S [36] is a Monte Carlo Tree Search based algorithm which uses a model equivalent to ours for modeling the VNE and the same shortest path algorithm for final reward calculation. It makes sense

to compare it with our method as it is similarly based on randomly simulating node placements but uses a different exploration strategy. This strategy, called Upper Confidence Bound for Trees explores the MDP as a tree of states (rooted in the initial state). It chooses where to descend in the tree by balancing exploration of new states and exploitation of known states, with the objective to minimize the regret of exploring new states given the expectated reward yielded by known states.

- UEPSO [40] is a particle swarm optimization (PSO) based meta-heuristic algorithm that shows good performance for the VNE.

- GraphVine [46] is a recently proposed method that exploits graph neural networks for selecting the physical nodes on which to place the virtual nodes. Note that GraphVine, like us, learns online, different from other neural network approaches such as [37], which requires an extensive offline training first, tied to the physical network. For this reason, a comparison with these other approaches would not be fair. This is why we prefer to compare with [46] instead.

For the sake of clarity, we keep in this section only the comparison with the state-of-the-art methods (mentioned above). We postpone the ablation study of NEPA (and its improvement over NRPA) to Appendix C, which shows the benefits brought to NEPA by weight initialization and neighborhood-based refinements. We implemented all these methods in the Julia programming language and made the code available as open source[68], except for GraphVine for which we use the publicly available Python/Pytorch implementation. In order to compare in the fairest manner possible, we run the following experiments:

- We run NEPA with parameters $N = 5$ and $l = 3$.

- MaVEN-S is executed with a computational budget (*i.e.* the total number of link placement attemps it executes per CNSs) of 445 link placements per CNS. Note we tried to run it for longer times (up to 670 iterations per CNS) without a significant improvement of results.

- Since UEPSO is a non-recursive algorithm, it is easier to stop it at any moment and get a valid placement. Hence here, we simply stop UEPSO after a certain amount of time equal to the mean time taken by NEPA.

- Finally we run GraphVine with the default implementation, as it is a quite different algorithm which does not rely on repeated simulations and since it can exploit a GPU. However with our original machine, we note that it is the slowest to run. As shown in [46], the algorithm

is better suited for using a GPU. For that reason, we run it on another computer which has a GPU (as it gave the best runtime). This machine uses an nvidia A3000 and an Intel i7-11850H CPU.

Runtimes are depicted in Figure 3.8. We run each of the described experiments 10 times with different random seeds, except for GraphVine for which we run it only once due to the high computational cost. Note that we compute 99% confidence intervals of acceptance and revenue-to-cost ratio for MaVEN-S, UEPSO and NEPA. Some figures do not display them because they are too narrow to be visible on figures. (*i.e.* confidence interval in the order of less than $\pm$ 0.01 for acceptance and revenue-to-cost ratio).

3.5.2.2 Results on synthetic physical topologies

We start our experiments with a sensitivity analysis. For this part, we generate scenarios with default parameters and we vary each of these one by one in order to assess the results on a representative set of cases. Default parameters are reported in Table 3.2. We choose to generate our CNSs and virtual networks with the Waxman generation algorithm as it is commonly used in the VNE literature [36] [33]. We choose to generate 500 CNSs per scenario as we validated experimentally this gave enough time for the system to stabilize in terms of acceptance ratio. We then vary parameters in the following ways :

- We generate CNSs with varying Poisson arrival rates between $\lambda = 0.02$ and $\lambda = 0.08$ arrivals per second. (results in Fig. 3.9.1/3.9.5)
- We generate CNSs with sizes (number of virtual nodes) with minimum size $7 + i$ and maximum size $13 + i$ for $i \in [0, 9]$. (results in Fig. 3.9.2/3.9.6)
- We modify the physical network from the default scenario by removing bandwidth and CPU capacities in increments of 5 from links and nodes of the physical network, making resources scarcer. Since initially the resource capacities (CPU and BW) are chosen uniformly at random between 50 and 100, their mean value is about 75. Since we remove from all nodes and links, the mean number of resources for the different scenarios is 70, 65, 60, down to 45. (results in Fig. 3.9.3/3.9.7)
- We generate 10 different physical networks and CNS sets for each physical network size of 50, 60, 70, 80, 90, 100 nodes. In those scenarios, we use the default parameters, but with $\lambda = 0.04$ and CNS sizes as specified in Table 3.3. We scale the size of CNSs with respect to the physical network size since our early experiments showed that

Parameter	Default Value
CNS arrival rate λ	0.02
CNS departure rate μ	0.005
CNS generator	Waxman ($\alpha = 0.5, \beta = 0.2$)
Number of CNSs	500
Min $\|V^x\|$	7
Max $\|V^x\|$	13
$\|V\|$	75
$\|E\|$	273
CPU demands (number of cores)	1 - 50
BW demands (Gbps)	1 - 50
Physical CPU capacities (number of cores per node)	50 - 100
Physical BW capacities (Gbps per link)	50 - 100

Table 3.2: Default scenario generation parameters

Number of physical nodes of	Number of virtual nodes
50	7-13
60	8-14
70	9-15
80	10-16
90	11-17
100	12-18

Table 3.3: Mean number of nodes of virtual networks for each size of physical network tested

if the CNS sizes were the same for all physical networks tested, it resulted in too easy scenarios for larger physical networks, where most algorithms reached performances close to 100% acceptance rate, making the comparison pointless. (results in Fig. 3.9.4/3.9.8).

Figure 3.9 shows that on every tested scenario, NEPA beats all other algorithms by a large margin, consistently beating MCTS of around 50% of acceptance and the best of other contenders (which are close to each other, above MCTS) by 15%, regardless of the case. In terms of revenue-to-cost ratio, it is striking to note that NEPA beats other algorithms by an even larger margin than for acceptance. This means NEPA tends to use less physical resources, which is the reason why it achieves a better acceptance. This suggests that reducing the overall consumption of each CNS enables

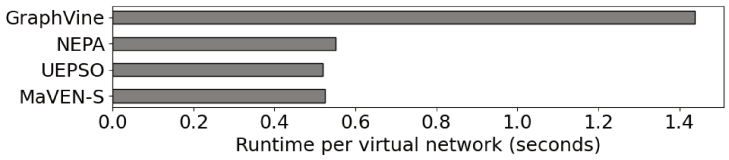


Figure 3.8: Mean runtime per CNS for each algorithm (calulated by averaging runtime per CNS on all runs of varying λ scenarios)

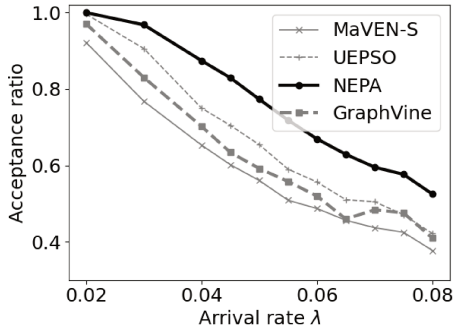
us to leave more resources for future incoming CNSs, making it possible to place them.
For variable size physical networks (figure 3.9.4), we observe again that NEPA beats other contenders by a large margin, since it accepts up to 60% more than MaVEN-S, and consistently beats it by 20 points of acceptance. It is remarkable to note how regular the patterns are in the acceptance plots, especially given that results are averaged for different topologies (recall that in the variable size experiment, for each seed, we generate a different random topology). The difference between algorithms is almost always the same regardless of sizes and difficulty of the instance, with NEPA as a clear winner. Regarding revenue-to-cost ratios, we note that all algorithms except NEPA have average ratios between 0.5-0.6. NEPA beats them by a large margin, since it is the only one to consistently reach 0.7 to 0.75 of revenue-to-cost ratio, showing again the effectiveness of the neighborhood based refinement in increasing the quality of the solutions found.

3.5.2.3 Real Topologies

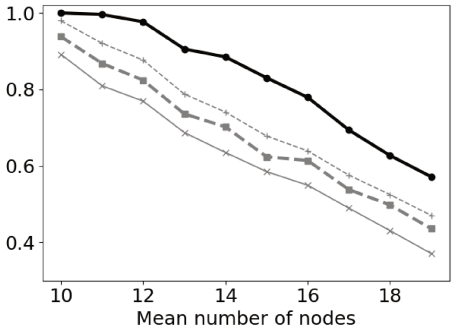
We try all algorithms with real topologies from the TopologyZoo [25] dataset as physical networks. We choose to use topologies that have between 60 and 200 nodes and are connected. This leaves us with 26 topologies with $|\mathcal{V}|$ between 60 and 197. We use bandwidth capacities chosen randomly between 250 and 300 Gbps and CPU capacities between 50 and 100 cores. CNSs are generated with our standard scheme but with $\lambda = 0.04$ arrivals per second.

The results depicted in Figures 3.10 and 3.11 show that NEPA is a lot more effective than UEPSO and MCTS. We achieve improvements of at least one order of magnitude in terms of acceptance compared to these algorithms with our best result being to more-than-triple their acceptance ratio on the Syrin topology by using NEPA. GraphVine interestingly performs much better on those topologies than on random ones, however NEPA still is the best in terms of acceptance rate with only few experiments where GraphVine manages to reach a similar acceptance as NEPA, and only 2 where it beats our algorithm by a thin margin. In terms of revenue-to-cost ratios, results are on par with acceptance, since again NEPA beats other algorithms (GraphVine aside) by an order of magnitude. We note that on some instances, GraphVine has a worse revenue-to-cost ratio than NEPA but still matches it in terms of acceptance (CogentCo, GtsCe, Pern, ...). This observation implies that although improving revenue-to-cost ratio is a key factor in order to reach a higher acceptance, it is not the only parameter to look for, since an approach can have a worse revenue-to-cost ratio but a better long term acceptance ratio.

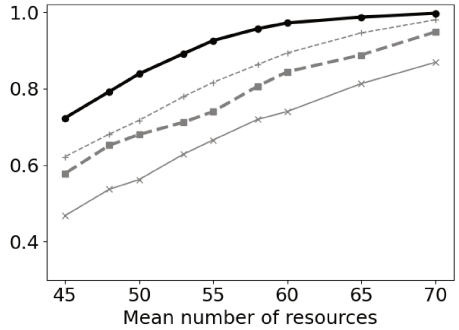
Overall, our results suggest that NEPA is the best suited method com-



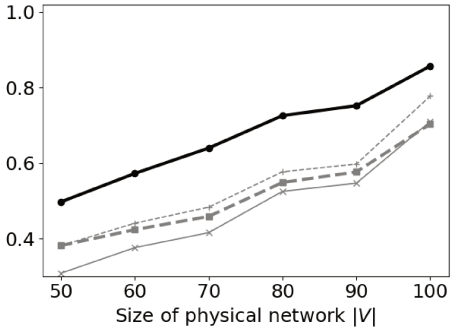
3.9.1 Acceptances for varying arrival rate (λ)



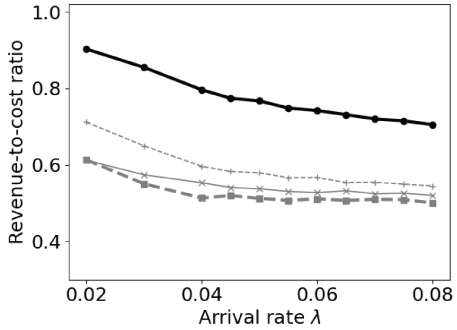
3.9.2 Acceptances for varying CNS size



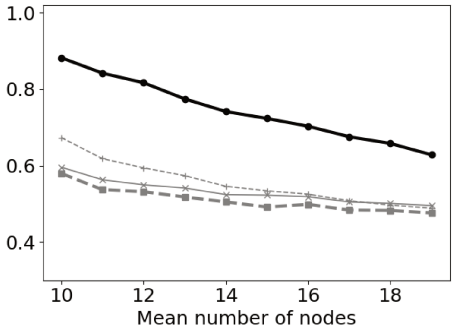
3.9.3 Acceptances (varying CPU & BW capacities)



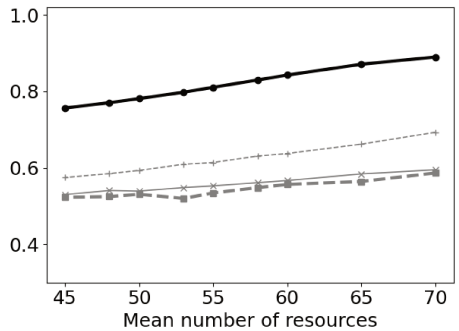
3.9.4 Mean acceptance ratios for varying physical network sizes



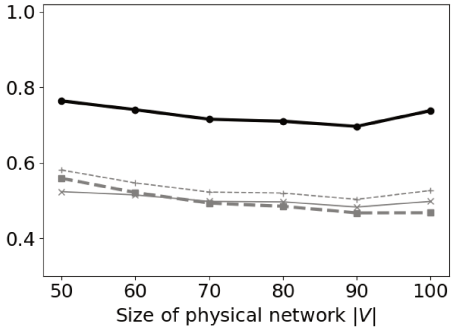
3.9.5 Revenue-to-cost ratio for varying λ



3.9.6 Revenue-to-cost ratio for varying CNS size



3.9.7 Revenue-to-cost ratio (varying CPU & BW capacities)



3.9.8 Revenue-to-cost ratio (varying network sizes)

Figure 3.9: Results for sensitivity analysis experiments

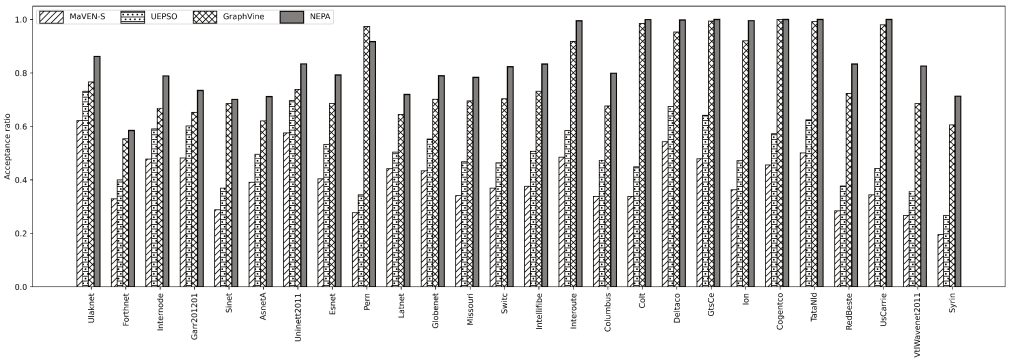


Figure 3.10: Acceptance on real physical networks. Results are ordered by increasing shortest-path length variance. See Appendix A for large version of the figure.

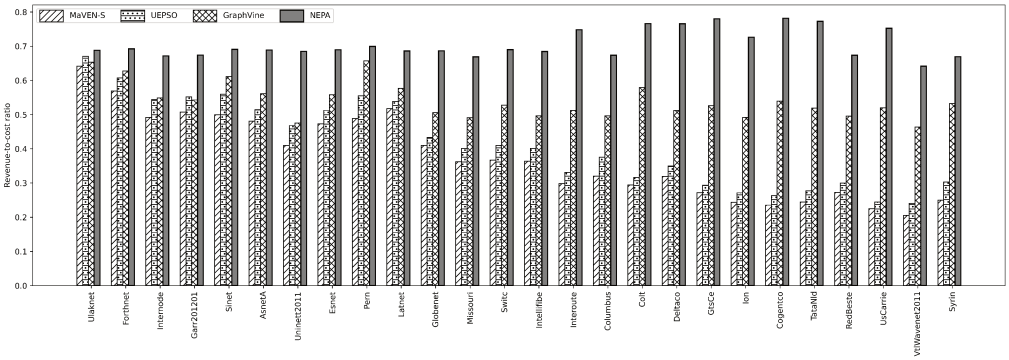


Figure 3.11: Revenue-to-cost ratios on real physical networks. See Appendix A for large version of the figure.

	Mean dist.	Diam.	Std. dev. of shortest path length	Clustering coeff.
Correlation	0.72	0.65	0.71	-0.17
p-value	3×10^{-5}	3×10^{-4}	5×10^{-5}	0.38

Table 3.4: Correlation between graph topological statistics and improvement ratio from NRPA to NEPA for real topologies.

pared to state of the art algorithms when it comes to placing CNSs on real-world networks. We note however that although the GraphVine method struggled on random topologies, it is competitive when it comes to real networks, although not as good as NEPA overall. We think it would be a great area of future research to try to combine both methods, as NEPA might be able to leverage the addition of GraphVine's neural network for reusing information learned accross experiences. Results also suggests that the topology of the physical network has a great influence on the performances of each algorithms.

Starting from that observation, we investigate the key topological features that enable NEPA to peform so much better in those cases. Our data exploration reveals that real topologies tend to have a larger diameter and mean shortest path length (*e.g* overall longer paths) than generated ones. They also tend to have a lower link density (*i.e.* they have "less" edges). We depict statistics for these topologies compared to generated ones in D.2, in Appendix D. The difference between real and synthetic topologies questions the appropriateness of the widely used in the literature Waxman Generator for VNE studies.

Furthermore, there is often a larger standard deviation in the length of shortest paths (*i.e.* distances) in real topologies than in synthetic ones. We notice that the differences between NEPA (which uses distance information a lot) and other algorithms is the most important for real topologies where the standard deviation in shortest path length is the largest. For example with Syrin, where acceptance rises from 0.18 - 0.23 for MaVEN-S and UEPSO to 0.77 with NEPA, and where the shortest path length standard deviation is 6.77. We observe the same pattern with VtlWavenet2011, UsCarrie, RedBeste, Cogentco or TataNld. On the other hand, when standard deviation is low (Ulaknet, Internode, Sinet, Forthnet, ...), we notice that the differences between algorithms are much lower, as the information to be leveraged from distances is less important, since choosing a "bad" placement would result in a smaller augmentation of the cost. Note however that NEPA still beats all other algorithms in those cases, although it is by a thinner margin. We quantify the advantage NEPA gets from exploiting distance information (*i.e.* using weight initialization and refinement) by calculating the augmentation ratio between the acceptance of NEPA and the acceptance of NRPA-W (which is depicted in Appendix C) for each real-topology scenario. We choose to compare against NRPA-W as

it is the same algorithm, but with no help from distance-based information during node placement. We then calculate the correlation between the augmentation ratio and different topological measures for results on the real-world topologies.

Those correlation results (obtained using Pearson correlation coefficient) are depicted in Table 3.4. We find strong positive correlations of 0.65, 0.71 and 0.72 respectively for diameter, standard deviation of distances and mean distance, meaning distance information is particularly important to exploit when the physical network has a high standard deviation in the distribution of distances, such as in many of the real networks studied. This explains why our algorithm can perform so much better on these instances. This is relatively intuitive to understand: these cases correspond to instances where there are a lot of chances to make "high-cost mistakes", *e.g.* where a single virtual link could incur à lot of cost by being placed on two physical nodes that are far from one another. Our distance-based techniques explicitly mitigate this by ensuring virtual nodes are placed close to one another, which results in an even greater performance boost on those cases. Also notice that in this paragraph our analysis was focused on standard deviation but applies to the other distance related metrics, as mean distance, diameter and standard deviation all have a correlation between one another of 0.99, according to our measurements.

3.5.2.4 Specific case: Perfectly solvable scenarios

In this section, we evaluate each algorithm on perfectly solvable scenarios (PSS). A PSS is a kind of scenario proposed by Fischer [66] that is generated such that there are only CNS arrivals and no departure, and such that it is possible to place all CNSs. The scenario is generated so the only solutions where all CNSs are placed leave 0 remaining resources. Hence it is a very hard, but theoretically feasible scenario (*i.e.* 100% of acceptance is reachable).

We argue evaluating algorithms on such scenarios is an important but often overlooked practice in the literature. Indeed, it is generally infeasible to evaluate the suboptimality gap as computing the exact placement would be computationally too expensive. We generate 10 PSS scenarios, using the additive algorithm from [66] . The generation is done by first generating CNSs, then "adding" them in order to form the physical network. The "addition" step is done by treating each virtual node iteratively, either reusing an already created physical node or creating a new one for the current virtual node (the choice is made probabilistically). Then, once all physical nodes have been created, they are linked so that if two nodes host neighboring virtual nodes, bandwidth is added to the link between them equal to the requirement of the corresponding virtual link.

Instance	PSS0	PSS1	PSS2	PSS3	PSS4	PSS5	PSS6	PSS7	PSS8	PSS9
$\|V\|$	67	86	73	94	111	104	124	121	135	150
$\|V^x\|$	7-10	8-11	9-12	10-13	11-14	12-15	13-16	14-17	15-18	16-19

Table 3.5: Number of nodes for physical and virtual networks of PSS scenarios

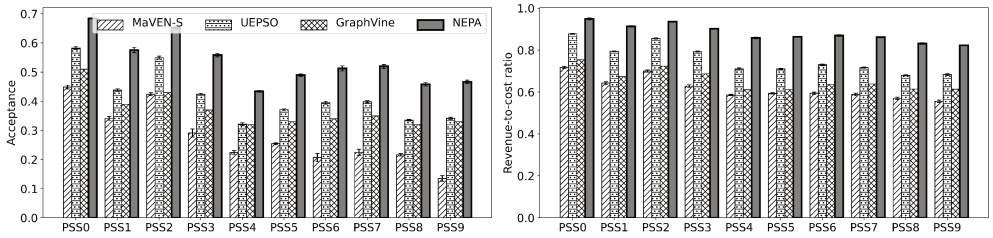


Figure 3.12: Acceptance and revenue-to-cost ratios for perfectly solvable scenarios

Each scenario $PSSi$ is generated from a batch of 100 random CNSs of random size $7+i$ to $10+i$ and a probability of reusing existing nodes of 0.93. This parameter was chosen empirically as it enabled us to generate graphs of sizes similar to those we experimented with in the previous section, as shown in Table 3.5. Our experiments show the same kind of results (shown in figure 3.12) as for the previous part, *i.e.* that NEPA outperforms all other methods by an order of magnitude (consistently beating the second best method, UEPSO by accepting up to 35% more CNSs in PSS9), both in terms of acceptance and revenue-to-cost ratio. However, it is striking to note that we never achieve a result of 100% of acceptance (the best one is NEPA on PSS0 - the smallest case - with 69%), even though the revenue-to-cost ratio gets really close to 1 (up to 0.965 in the first scenario). This means that although we achieve an almost perfect online optimization objective, resources on the physical network are badly used, leaving a lot of "holes" which are unusable. We believe this shows the need for the VNE community to investigate better reward functions which could assess the quality of a solution with other metrics than pure resource usage (with the goal to define whether a virtual network "fits" its embedding or not). In that regard, a recent article [43] made a first step in that direction by proposing to enrich the reward function with degree information, which slightly helps improving acceptance depending on the algorithm used. However, the results shown in Appendix B demonstrate this reward function has no significant impact on the results of the NEPA placement, suggesting it is ineffective when the algorithm is already very good. Another possibility would be to place virtual networks in batches, which might enable us to combine placements better, at the cost of a higher computational complexity.

3.6 Conclusion and remarks

Our results illustrate that the widely adopted idea of optimizing placement for reduced bandwidth [36] [37] [52] [46] consumption in order to let more resources for future virtual networks works well. We show that pushing this logic a step further by explicitly reducing the consumption of the found solution enables our algorithm to reach even better results. The main hurdle with the refinement step is the computational cost, which we overcome by selecting promising solutions to refine instead of trying to refine any solution. The NRPA algorithm is easily adaptable into NEPA due to its recursive nature. It is an open question whether other algorithms such as UEPSO could be modified in order to similarly select promising states to be refined, which would enable them to keep the computational cost low while finding better embeddings.

We shall now focus on the differences between MaVEN-S (which we call a mean-based approach) and NRPA and NEPA (which we call max-based approaches), in an effort to try to explain why max-based approaches perform so much better than the MCTS-based MaVEN-S algorithm (refer to Appendix C which shows the ablation study of NEPA, also demonstrating that NRPA without the improvements brought by NEPA outperforms MaVEN-S), while both types of algorithms are Monte Carlo Search algorithms that try to balance exploration and exploitation of the tree formed by the MDP underlying our embedding problem.

In figure 3.13, we illustrate with a toy example the potential results obtained after executing 6 random simulations (with a policy that could either be given by NRPA/NEPA's policy matrix or by MaVEN-S's tree). The tree represents the MDP, with the final values obtained through simulations at its leaves. This tree will serve us to illustrate the key difference between algorithms: max-based approaches assume that the best solutions lie near the single best solution found so far, hence they will explore regions of the search tree even with low expected value as long as they contain the best solution found so far. On the other hand, MCTS is designed to explore the states with the best expected (mean) value. Hence, on the tree from figure 3.13, MaVEN-S, would exploit more the states of the bottom sub-tree, since their mean value would be of 0.53, while max-based approaches would go for the top sub-tree since the maximum known value is of 0.9, even though the mean value would only be of 0.47.

We argue this is desirable for the VNE problem we solve, because optimizing for the mean expected reward is typically suited for problems where there is uncertainty, *i.e.* where taking one action from a given state can yield to several different states. This is not the case for the VNE, where a choice of action from a given state always yields to the same state: the model is a deterministic MDP. Hence it makes more sense to choose ac-

tions only according to the best sequence found so far and not according to the best mean value. This is what max-based approaches do since they optimize considering the best sequence found, as opposed to mean-based MaVEN-S, which partly explains why MaVEN-S is outperformed.

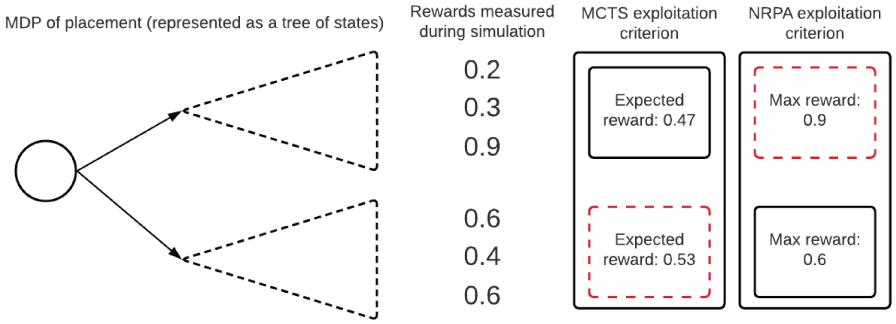


Figure 3.13: Toy example of MDP exploration choices

This property of NRPA, combined with neighborhood search (applied in a frugal way to only the most promising embeddings) and a good heuristic initialization of the weights of the algorithm are the key components which enable us to formulate the NEPA algorithm, which gives state-of-the-art results on commonly used synthetic benchmarks as well as on real network topologies, while keeping the running time comparable to earlier algorithms. Experiments on real topologies show that since it exploits distances between placed nodes, NEPA brings acceptance and revenue-to-cost improvements of an order of magnitude compared other meta-heuristic and Monte Carlo search algorithms. It also beats a state-of-the-art graph neural network based approach (GraphVine). On random topologies, which are the most explored in the VNE literature, we also showed that NEPA is the most robust of the tested approaches since it always reaches the best acceptance and revenue-to-cost ratio. These results will help in solving the resource allocations problems in future 5G networks, but also help the VNE community better evaluate its algorithms, since we characterized how topological features can induce enormous differences between results from different methods. In the future, we would like to demonstrate how to use NEPA on other combinatorial problems where good neighborhood search policies are also available, such as the TSP and the VRP. We also plan on incorporating offline learning by reusing the learned weights from past NEPA runs in order to learn better how to initialize future weights as currently these data are not used once the placement is decided. In that regard, a combination with GraphVine would be particularly appealing. Implementing NEPA on a real 5G network is also planned in the near future. Finally, we contribute to the VNE community by making our set of instances and of implementations available online [68].

Algorithm 6 REFINE procedure

Input: K: Max number of physical nodes candidates, X:Number of iterations, G: Physical network, H^x: CNS, seq: node placement sequence, $\mathscr{P}$: link mapping for seq, R^{seq}: reward yielded by seq and $\mathscr{P}$

Output: $R^{seq^{ref}}$: Refined solution reward, seq^{ref}: refined node mapping, $\mathscr{P}^{ref}$: refined mapping

$seq^{ref} \leftarrow seq$
$R^{seq^{ref}} \leftarrow R^{seq}$
$\mathscr{P}^{ref} \leftarrow \mathscr{P}$
for X iterations **do**
    $previous\_R \leftarrow R^{seq^{ref}}$
    Compute most promising virtual node to move in placement given by seq^{ref}, $\mathscr{P}^{ref}$ using eq.(9)
    Build set of K best physical nodes (ranked using eq.(3.8)) $V_r \subset V$ suitable for hosting the virtual node
    for $v \in V_r$ **do**
        Put virtual node on physical node v, *i.e.*:
        • Compute updated version of seq^{ref} seq
        • Compute new shortest paths $\mathscr{P}$ considering the new position of the virtual node
        • Update resources used on the physical node and links newly used.
        Compute R^{seq} using seq and $\mathscr{P}$
        if $R^{seq} > R^{seq^{ref}}$ **then**
            $R^{seq^{ref}} \leftarrow R^{seq}$
            $seq^{ref} \leftarrow seq$
            $\mathscr{P}^{ref} \leftarrow \mathscr{P}$
        else
            Restore resources used on physical node and link to values matching seq^{ref}, $\mathscr{P}^{ref}$
        end if
    end for
    if previous_R $= R^{seq^{ref}}$ **then**
        break
    end if
end for
return $R^{seq^{ref}}, seq^{ref}, \mathscr{P}^{ref}$

Algorithm 7 NEPA Algorithm

Input: Search level l, Number of iterations N, Policy matrix P, Initial state $s(0)$, Set of virtual links $\mathcal{E}^x$, Physical network $\mathcal{G}$, Refinement level l', K: number of candidate physical nodes, X : number max of refinements

Output: Best score achieved and best sequence of actions to achieve it

if $l = 0$ **then**
  return SIMULATION($s(0)$, P, $\mathcal{E}^x$, $\mathcal{G}$)
end if
$R^{seq^{best}} \leftarrow -\infty$
$seq^{best} \leftarrow \emptyset$
$\boldsymbol{\mathcal{P}}_m^{best} \leftarrow \emptyset$
for N iterations **do**
  R^{seq}, $seq, \boldsymbol{\mathcal{P}}_m \leftarrow$ NEPA$(l-1, N, P, s(0), \mathcal{E}^x, \mathcal{G}, l')$
  if $R^{seq^{best}} \leq R^{seq}$ **then**
    $R^{seq^{best}} \leftarrow R^{seq}$
    $seq^{best} \leftarrow seq$
    $\boldsymbol{\mathcal{P}}_m^{best} \leftarrow \boldsymbol{\mathcal{P}}_m$
  end if
  if l = l' and $R^{seq^{best}} \neq 0$ **then**
    $R^{seq^{best}}$, seq^{best}, $\boldsymbol{\mathcal{P}}_m^{best}$ = REFINE(K, X, $\mathcal{G}$, $\mathcal{H}^x$, seq^{best}, $\boldsymbol{\mathcal{P}}_m^{best}$, $R^{seq^{best}}$)
  end if
  $P \leftarrow$ ADAPT(P, seq)
end for
return $R^{seq^{best}}$, seq^{best}, $\boldsymbol{\mathcal{P}}_m^{best}$

Chapter 4

Edge Network Slice Placement and Dimensionning

4.1 Introduction

In this chapter, we turn from the Virtual Network Embedding and focus on Edge Slice (ES) Placement. The ES are used by slice tenants to serve users of their low latency services such as video streaming, autonomous vehicle management services, machine learning-based data-processing, etc. The particularity of these services is that due to latency constraint, they have to be hosted close to the user instead of being in a datacenter or a cloud. We depict two example placed ESs in Figure 4.1 (where propagation delays and link bandwidth capacities of the underlying network are depicted). Each ES is composed of a set of flows, with each a source or ingress node (*e.g* a base station), where a certain amount of user requests (which we model more precisely later in this chapter) arises, along with a set of potential destinations, where the edge service can be placed. Note each flow can have several valid destinations where to place the service and that we choose one per flow during the optimization. Also note that throughout this chapter, when we talk about edge services, it is an abstraction over the practical implementation: such edge services would need to communicate with the outer world, which, according to the 5G architecture, they should do through a UPF. For this reason, the edge services we are designating are in fact composed of the application provided by the slice tenant as well as a UPF, collocated on the same node.

Furthermore, the key constraint of such edge services is that they have to be served with stringent latency and reliability, *e.g.*, informally, tenants give the operator Service Level Agreements (SLAs) such as "99% of packets should experience an application delay of less than 10ms".

In short, in this chapter, we are given a set of sources, potential destinations, constraints and request characteristics (sizes, intensity, location),

and we have to output a set of paths and CPU allocations such that all services of each slice meet the requirements.

Compared to the VNE, such a problem requires a more flexible model, where latency is taken into account. It is also required to take processing delays into accounts, and to more finely decide the consumed CPU: in the previous chapter, we assumed that the CPU consumed by a virtual node was known, in this chapter, we also take the dimensionning of nodes into account on top of the placement.

Furthermore, it has been reported that the network Operational Expenditure (OPEX) represent 25% of the total costs of operators, of which 90% is energy bills [2]. For this reason, in this chapter, we shall focus on the objective of minimizing the energy consumption of the edge network, instead of the operators' revenue.

In short, in this chapter, we answer the following questions:
How should a Physical Network Operator (PNO) jointly decide (i) placement of services, (ii) routing and (iii) allocation of computational resources to each edge slice? And, how to take those decisions to minimize energy consumption, while satisfying SLAs? Despite their tight inter-dependency, decisions (i), (ii) and (iii) have never been optimized jointly due to high complexity. To the best of our knowledge we are the first to formalize and solve optimally a mathematical program which takes the three aforementioned decisions jointly. Leveraging appropriate inter-relations between computational resources and network paths and applying column generation (CG), we are able to find exact solutions for real-sized networks. Thanks to our joint approach, we get rid of restrictive assumptions used in previous work which either consider that all the resources to allocate are known a priori (*e.g.*, by allocating a fixed amount of CPU to match the a-priori known demand, similar to what is done in the VNE) [69, 8, 36, 70, 71], or formalize the routing problem as a multi-commodity flow problem and assume that the "destination node" is fixed and determined a-priori [72, 73].

We differentiate ourselves from the above works in the following aspects. First, the amount of resources to be allocated to each edge slice is for us an optimization variable, which gives us the opportunity to meet SLA requirements at a finer grain (e.g., allocating more CPUs resources to execute services faster). Second, we fully leverage the flexibility of virtualized networks and let the PNO decide the placement of software components, as well as which nodes to activate/deactivate to minimize energy consumption. Moreover, while latency constraints are usually expressed in terms of mean delay [69] [74], we model SLAs by considering a maximum latency threshold and the corresponding level of reliability, *i.e.*, the minimum fraction of time where the threshold must be met.

By jointly deciding placement, routing and computational resource allocation policies, and by accurately modeling SLAs, we provide each

edge slice with the precise amount of resources needed to satisfy SLAs. This allows minimizing energy and reducing the risk of over- and under-provisioning.

We formulate the problem as a Mixed Integer Mathematical Programming problem, which we show can be formulated as an Integer Linear Program (ILP). However that ILP has a number of variables exponential in the size of the graph, so we cannot solve it directly with an off-the-shelf solver. For this reason, we resort to column generation, a technique where we solve iteratively by progressively including more variables into the problem. This technique requires to select the best variable to include at each step. This is called the pricing problem, and in our case, we will see that it can be formulated as a bi-objective shortest-path problem with forbidden paths. For this reason, we shall first detail how to solve that problem.

The rest of the chapter is organized as follows. Section 4.2.1 surveys related work on edge service placement as well as on all the methodological tools used in the rest of the chapter. Section 4.3 details the first contribution on multi-objective, multiconstrained shortest paths problems. It is itself split into the following subsections:

- Section 4.3.1 presents multi-objective, multi-constrained shortest path problem with forbidden paths that we treat
- Section 4.3.2 details the algorithm we propose to solve it
- Section 4.3.3 presents our numerical results.

Then, in Section 4.4, we present the edge placement problem. This section has the following subsections:

- Section 4.4.1 presents our edge placement model and assumptions
- Section 4.4.2 formulates the problem of joint edge service placement, routing and computational resource dimensioning to minimize power consumption as a Non-Linear Mixed Integer Program (NLMIP)
- Section 4.4.3 shows how the NLMIP can be transformed into a more tractable Mixed Integer Linear Program (MILP) and describes our exact algorithm
- Section 4.4.4 evaluates the proposed algorithm numerically.

Then, in Section 4.5, we investigate why the problem we formulate is longer to solve than previously existing, similar problems. In that section we show that a simplified of the problem is NP-hard, while the same simplification of the existing problem is polynomial. Finally, in Section 4.6 we conclude the chapter with final remarks.

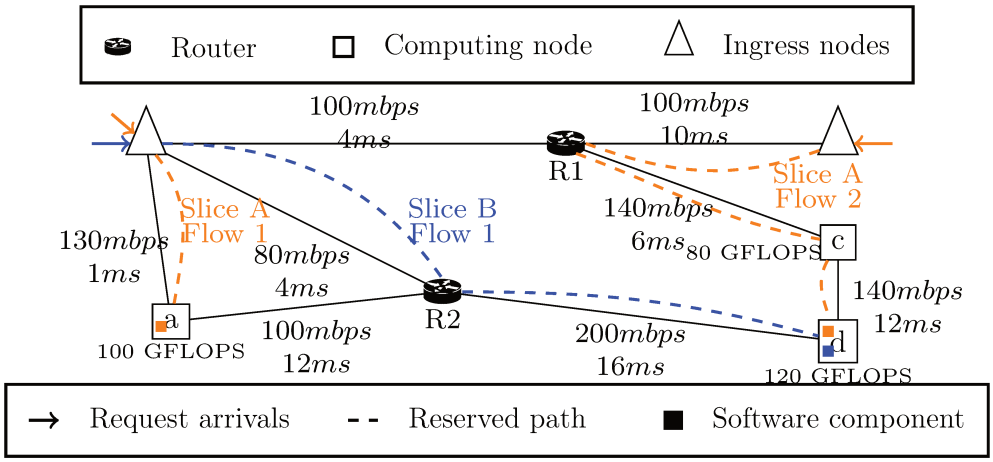


Figure 4.1: Example edge network.

4.2 Background

4.2.1 Other pre-existing methods and models for edge placement

Edge service placement is a widely studied problem in the literature. Most works in this field do not take the path between the service and its clients into account, and only a small fraction of those consider both energy and latency when placing edge services [75, 76, 77, 71, 78].

To take paths into account, one possible approach would be to adapt VNE techniques. Especially, the works taking aspects such as SLA [79] or energy [70] have been studied and could be adapted. However, these VNE techniques also don't provide the flexibility of modifying the amount of CPU resources allocated to nodes depending on the exact placement.

Another work which seeks to place services and paths jointly, but does not belong to the VNE literature, is [80]. Similar to us, it models the problem as a flow problem. However, again, SLA constraints are not considered. The problem of allocating resources while considering paths is also investigated by other works in the context of classic multicommodity-flow (MCF) problems [81] or OSPF weight optimization [82, 83]. In general, MCF is formulated as a resource minimization problem [5]. Another common objective is load balancing [84, 73], but it is of less interest for energy as it leads to using many nodes to distribute load. Overall, while the aspects we treat have received significant attention, our literature review shows that jointly allocating paths and resources while minimizing energy and satisfying SLAs is largely unexplored. Moreover, we also notice that our review does not suggest a good off-the-shelf candidate to perform a fair numerical comparison. Indeed, Salaht *et. al.* [85] pointed out that

"the service placement problem has been highly discussed in the literature [...] Based on different application descriptions, network assumptions, and expected outcomes, these solutions are generally difficult to compare with each other."

4.2.2 Background on used method: column generation

4.2.2.1 Introduction

Column Generation (CG) is a technique which consists in solving a continuous (in our case, Linear) Mathematical Programming problem by iteratively including variables in the problem: instead of directly solving for all variables, as done with the simplex method [86], the problem is solved for a subset of variables, then one variable is added, and the problem is solved again. A column generation algorithm typically iterates until we can prove adding any of the currently excluded variables cannot improve the current objective function. This technique has had great success in areas such as Transportation [87], Energy Networks optimization [88] or computer networks [89]. Column Generation can also be used to solve Integer Mathematical Programs. In this case, it has to be embedded in a branching scheme: the CG algorithm is used as a subroutine, to solve the continuous relaxation of the problem, and the algorithm branches on the integer variables, in the exact same way as a branch-and-bound algorithm [90]. Such an algorithm is usually called a branch-and-price algorithm.

The main motivation of CG is that it enables us to deal with very large numbers of variables. Typically, the problems solved with CG have a number of variables that grows exponentially with some parameter of the problem. In such cases, classic techniques that consider the whole problem at once (such as the Simplex Method or Interior Point Algorithms) would quickly be limited, while CG would not. This section is organized as follows: we start with a refresher on Linear Programming and the simplex method in section 4.2.2.2, then section 4.2.2.3 explains how CG can be used to solve LPs, section 4.2.2.4 then introduces Branch-and-Bound for solving Integer Linear Programs and in section 4.2.2.5 we explain how CG can be used in conjunction with BnB, deriving the BnP algorithm.

4.2.2.2 Solving Linear Programs with the Simplex algorithm

In this section, we consider a Linear Program (LP) in standard form with n variables and m constraints

$$\begin{aligned} \max \quad & \mathbf{z} = \mathbf{c}^T\mathbf{x} && (4.1)\\ s.t. \quad & \mathbf{A}\mathbf{x} \leq \mathbf{b} && (4.2)\\ & \mathbf{x} \geq 0 && (4.3) \end{aligned}$$

. For every linear program in standard form, there also exists an equivalent linear program in canonical form, in which slack variables have been added to the constraints:

$$\begin{aligned} \max \quad & \mathbf{z} = \mathbf{c}^T\mathbf{x} && (4.4)\\ s.t. \quad & \mathbf{A}\mathbf{x} = \mathbf{b} && (4.5)\\ & \mathbf{x} \geq 0 && (4.6) \end{aligned}$$

. Where $\mathbf{x} \in \mathbb{R}^{n\times 1}$ is the vector of variables, and we have $\mathbf{A} \in \mathbb{R}^{m\times n}, \mathbf{b} \in \mathbb{R}^{m\times 1}$. We note any LP can be transformed into an equivalent standard form LP by using linear transformations. We assume that $m \leq n$ and rank($\mathbf{A}$)[1]. Hence, there exists some square non-singular submatrix $\mathbf{B}$ of $\mathbf{A}$ of order m. We split $\mathbf{A}$ into ($\mathbf{B}$ $\mathbf{N}$), where $\mathbf{B} \in \mathbb{R}^{m\times m}$ is non-singular and $\mathbf{N} \in \mathbb{R}^{m\times n-m}$. We call $\mathbf{B}$ the basic matrix, which holds the basic columns and we call the columns of $\mathbf{N}$ non-basic. We also split $\mathbf{x} = \begin{pmatrix} \mathbf{x}_B \\ \mathbf{x}_N \end{pmatrix}$ and $\mathbf{c} = \begin{pmatrix} \mathbf{c}_B \\ \mathbf{c}_N \end{pmatrix}$. We call $\mathbf{x}_B$ the vector of basic variables. We can then rewrite the problem as

$$\begin{aligned} \max \quad & \mathbf{z} = \mathbf{c}_B^T\mathbf{x}_B + \mathbf{c}_N^T\mathbf{x}_N && (4.7)\\ s.t. \quad & \mathbf{B}\mathbf{x}_B + \mathbf{N}\mathbf{x}_N = \mathbf{b} && (4.8)\\ & \mathbf{x}_B, \mathbf{x}_N \geq 0 && (4.9) \end{aligned}$$

from equation (4.8) we can derive

$$\mathbf{x}_B = \mathbf{B}^{-1}\mathbf{b} - \mathbf{B}^{-1}\mathbf{N}\mathbf{x}_N \quad (4.10)$$

Definition 1. *(from [91]) We call* $(\mathbf{x}_B^T, \mathbf{x}_N^T = 0^T)$ *a basic solution to* $\mathbf{A}\mathbf{x} = \mathbf{b}$ *with respect to the basis* $\mathbf{B}$*. If some of the basic variables of a basic solution are zero, then the basic solution is said to be a degenerate basic solution. A vector* $\mathbf{x}$ *satisfying* $\mathbf{A}\mathbf{x} = B, \mathbf{x} > 0$ *is said to be a feasible solution. A feasible solution that is also basic is called a basic feasible solution. If the basic feasible solution is a degenerate basic solution, then it is called a degenerate basic feasible solution.*

[1] this can be ensured in practice by adding one slack variable per constraint.

Let us also introduce the dual of a linear program. We call the dual of the afforementionned program the following LP with dual variables $\boldsymbol{\pi}$:

$$\begin{aligned} \min \quad & \mathbf{d} = \mathbf{b}^T \boldsymbol{\pi} && (4.11) \\ s.t. \quad & \mathbf{A}^T \boldsymbol{\pi} = \mathbf{c} && (4.12) \\ & \boldsymbol{\pi} \geq 0 && (4.13) \end{aligned}$$

. The general principle of the simplex algorithm is to find an initial basic feasible solution, and then to iterate from one basic solution to the other. At each step, we improve the objective function. Then, once it is not possible to improve the objective anymore, we stop because it means the solution is optimal. Assuming we already have a basic feasible solution, we can rewrite the objective function as the following:

$$\begin{aligned} \mathbf{z} &= \mathbf{c}_B^T \mathbf{x}_B + \mathbf{c}_N^T \mathbf{x}_N && (4.14) \\ &= \mathbf{c}_B \mathbf{B}^{-1} \mathbf{b} - \mathbf{c}_B \mathbf{B}^{-1} \mathbf{N} \mathbf{x}_N + \mathbf{c}_N \mathbf{x}_N \quad \text{substituting (4.10)} && (4.15) \\ &= \mathbf{c}_B (\mathbf{B}^{-1} \mathbf{b}) + \mathbf{x}_N (\mathbf{c}_N - \mathbf{c}_B \mathbf{B}^{-1} \mathbf{N}) && (4.16) \end{aligned}$$

Since $\mathbf{x}_N = \mathbf{0}$, the current value of the objective function is $\mathbf{c}_B(\mathbf{B}^{-1}\mathbf{b})$. We call the term $\bar{\mathbf{c}} = \mathbf{c}_N - \mathbf{c}_B \mathbf{B}^{-1} \mathbf{N}$ the reduced cost vector of the current solution with elements $\bar{c}_j$. This quantity has been studied extensively. The two things of importance for us is that it can be shown [86] that:

a. If a variable x_j is increased by an infinitesimal quantity, we can expect the objective function to see a change of $rc_j \cdot x_j$. If the reduced cost of a variable is non-positive, including it in the basis won't improve the current solution. Furthermore all basic variables have a reduced cost of 0.

b. $\mathbf{c}_B \mathbf{B}^{-1}$ is equal to the vector of the dual variables $\boldsymbol{\pi}$, *e.g.* we can write $\bar{\mathbf{c}} = \mathbf{c}_N - \boldsymbol{\pi} \mathbf{N}$

The simplex algorithm can then be described as Algorithm 8:

Algorithm 8 Simplex algorithm

1: Find an initial basic feasible solution $\mathbf{B}$
2: **while** There are variables with positive reduced cost (resp negative in case of a minimization problem) **do**
3: Pick a variable x_j with negative reduced cost rc_j
4: Pivot on x_j (*e.g.* include x_j in $\mathbf{B}$ by correspondingly setting one of the current basic variables to 0)
5: **end while**

4.2.2.3 Solving Linear Programs (LPs) with CG

Let us call the LP we have to solve the Master Problem MP, with variables Ω. We denote by RMP(Ω_i) the Restricted Master Problem in which the only variables $\Omega_i \subset \Omega$ are considered. We denote its dual by D(Ω_i). CG consists in solving a LP = RMP(Ω) using the Simplex algorithm, by starting with solving RMP(Ω_0), and then finding an entering variable x with positive reduced cost (respectively, negative if LP is a minimization problem).

Then, $\Omega_1 = \Omega_0 \bigcup \{x\}$ is formed and RMP(Ω_1) is solved using the simplex method (one can reuse the solution to RMP(Ω_0) as a starting point). This process goes on until no variable with positive (resp negative) reduced cost is found.

Of course, the big question is how to find variables with positive (resp negative) reduced cost. In particular, we are interested in finding the largest (resp smallest) reduced cost, as doing so also provides a stopping criterion: if the best reduced cost is non-positive (resp non-negative), then we know we have found a set of variables Ω_i which is sufficient to optimally solve LP.

This optimization problem is called the pricing problem, which we formalize in the maximization case as the following:

$$\max \{rc_j : x_j \in \Omega\} \tag{4.17}$$

Solving this problem depends on the structure of the problem at hand and on the objects the variables represent. Typically, one will use column generation in formulations where variables represent complex objects. For example, variables could be paths, in which case the pricing problem could be amenable to some kind of shortest path problem or tours in the case of vehicle routing problems.

4.2.2.4 Branch-and-Bound

Let us now describe how an integer, binary or mixed linear program can be solved with the Branch-And-Bound (BnB) method. The branch and bound method relies on the simple principle of:

a. Solving the continuous relaxation of the problem (*e.g.* without the integrality constraints)

b. If the solution has at least one non-integer variable $x = A$, split the decision space in a way that excludes the value of the variable while keeping all integer solutions, *e.g.*, typically, create two instances of

the problem each with constraints $x \leq \lfloor A \rfloor$ and $x \geq \lceil A \rceil$ [2] Add the two newly found problems to the list of relaxed problems to solve, l_relax.

c. Start over recursively for one of the problems of l_relax.

This process is the branching part, it is combined with "bounding" (hence branch-and-bound), which gives us a stopping criterion to avoid exploring the whole tree of possibilities: once an integer solution is found, we know any fractional solution with a worst objective can be discarded, as no better integer solution lies in the decision space it defines.

4.2.2.5 Branch-and-Price

We now describe Branch-and-Price (BnP), *e.g.* the combination of Branch-and-Bound and Column Generation. BnP consists in using the CG instead of the simplex algorithm for solving the continuous relaxation. Then, the BnB algorithm is executed normally. While this principle is simple, there are a few technicalities to solve, which we shall now present. First, the branching strategy has to be adapted: if one uses the standard branching presented in the previous part, this adds a new constraint, which has a an extra dual variable associated with it, which modifies the structure of the pricing problem. Hence, a bad branching strategy can make the pricing problem harder or even impossible to solve in practice.

For example, in a problem such as the multicommodity flow problem [72], one can formulate the problem with one binary variable per path (each binary variable encodes whether the path is used or not). It can be shown that the pricing problem in that case is a shortest path problem, solvable with Dijkstra's algorithm [92]. If one uses the naive BnB branching strategy, it comes down to having paths which are forced to be excluded, making the shortest path problem much harder to solve. On the other hand, [72] proposed a branching scheme based on the structure of the pricing problem which keeps the pricing problem as a shortest path problem, and instead involves modifying the weights of the graph during the pricing problem solving, at no cost in terms of complexity.

Another issue of branch-and-price algorithms in practice is that solvers typically do not give access to their inner workings, and particularly, their data structures do not allow for adding columns (*e.g.* variables) during the BnB solution process. This means we can use commercial solvers as LP solvers but cannot leverage the heuristics and optimizations they implements for ILP.

[2] Any branching strategy can be chosen as long as the space of solution is partitionned in a way which excludes the current solution and in which all integer solutions are part of the joint spaces.

4.2.3 Background on Shortest path problems

We now turn to the litterature on various shortest paths problems with extra constraints or objectives. The reason for reviewing this type of problems is that, as we will see in Section 4.4.3.3, we shall formulate the pricing problem in the column generation algorithm as a combination of those problems.

The shortest paths (SP) problem is a classic problem of computer science that has been extensively studied since its introduction [92][93]. Formally, given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, with weights $w_{i,j}$ on each arc (i, j), the problem consists in finding the SP (with respect to the sum of weights $w_{i,j}$) between a source $s \in \mathcal{V}$ and a destination $t \in \mathcal{V}$. While this is a well known polynomial problem, several of its variants are NP-hard. Among them, three are particularly relevant: (i) the *resource-constrained SP problem* [94, 95], (ii) the *multi-objective SP problem* [96, 97] and (iii) the *SP problem with forbidden paths* [98, 99].

Those different problems have various applications. In transportation, related questions are "what is the best path, with minimum fuel consumption and time duration ?" or "among the paths with total tolls less than a certain threshold, which one is the shortest ?". In Operations research, the above problems arise in Vehicle Routing [100], Multicommodity Flow [72] and within subroutines of Column Generation [101]). The problems above are also relevant to computer networks [102]. In the Internet, we might need to know "what is the best path in terms of both latency and monetary cost" or "what is the best path that avoids traffic to be routed through a hostile nation [103]".

Those important variants of the SP problem have each been treated separately. However, in many cases the joint resolution of the three aforementioned problems may be required. There is still lack of solutions to do so and, to the best of our knowledge, we are the first to propose an algorithm that solves multi-objective SP problems while considering forbidden paths and resource constraints. This algorithm will then be used in our column-generation based solution.

4.2.3.1 Multi-objective SP problem (MOSPP)

In the MOSPP, each arc (i, j) has a vector of K weights $w^k_{i,j}$. Correspondingly, a path $\mathcal{P}$ is characterized by a vector of weights $w_{\mathcal{P}}$, equal to the sum of the weights of the arcs of the path $\left(\sum_{(i,j)\in\mathcal{P}} w^k_{i,j} | 1 \leq k \leq K \right) \forall 1 \leq k \leq K$. Each dimension is an *objective*. The goal is to find paths from a source to any destination which minimizes the objectives. Obviously, objectives may be conflicting. Therefore, the goal is not to find a single path, but the set of

all non-dominated paths. A path is non-dominated if there is no other path that is better along all objectives. In theory, this problem has been shown to be intractable by Hansen [104], because the set of non-dominated paths can be of size exponential in $|\mathcal{V}|$. However, in practice efficient algorithms exist. The one from [105] is based on a k-SP algorithm. Later approaches are based on label-setting [106, 107] and label-correction [108].

Our algorithm adds to the one from Brumbaugh-Smith and Shier the feature of expressing capacity constraints and preventing forbidden paths, joining the approaches described in the next two subsections. We choose to adapt this algorithm over more recent approaches like [109**?**] as those methods solve the one-to-one problem and assume extra information such as geographic distance is available. However, we are interested in solving cases where such information is not available and one-to-all SPs are necessary, although we note our proposal could also be adapted to those methods.

4.2.3.2 Resource-constrained SP (RCSP)

The second problem we consider is the RCSP. In this problem, the goal is to find a single SP $\mathcal{P}$ which is minimal with regards to the first dimension (weights $w^1_{i,j}, (i,j) \in \mathcal{E}$). We are given constraints C_k and the path found should be such that $\sum_{(i,j)\in\mathcal{P}} w^k_{i,j} \leq C_k$ for $k = 2, \ldots, K$. This problem is well studied and similarly to MOSP, various exact algorithms based off SP algorithms have been studied [110, 111]. While this problem is NP-hard [60], the cited studies show that it is often fast to solve exactly in practice.

4.2.3.3 SP with forbidden paths (SPFPP)

Finally, we are interested in the SP problem with forbidden paths. In this problem, we are given a set of paths $\mathcal{F}$ which we cannot use. The question is then to find the SPs of $\mathcal{G}$ that do not belong to $\mathcal{F}$. This problem first appeared in the context of computation of k-SP, where [112] iteratively constructed a variation of graph $\mathcal{G}$ in which the previously computed SP could not be used. [98] then pointed out that this construction is not practical when the set of forbidden paths is large, as the variations of graph $\mathcal{G}$ would also become too large. Their proposal was hence to compress $\mathcal{F}$ using a string matching algorithm [113]. Such an algorithm builds an automaton. By properly including this automation in the original graph, they ensured that by construction $\mathcal{F}$ cannot be used anymore. The main issue of this approach is that a simple path of the augmented graph might have a loop when transformed back to the original one. Pugliese *et. al.* then proposed a label-correcting algorithm for the SPFPP which solved this issue. Their algorithm works by reducing SPFPP to an instance of the

RCSP and solving it by several methods among which is a label-correcting algorithm: in their reduction, each of the forbidden paths $\mathcal{F} = \{\mathcal{P}^k\}_{k=2}^{|\mathcal{F}|+1}$ is associated with a resource constraint $C^k \in \mathbb{N}$ which depends on the length of forbidden path $\mathcal{P}^k$. Then, if $(i,j) \in \mathcal{E}$ is part of $\mathcal{P}^k$ they set $w_{i,j}^k$ to 1. Otherwise, it is set to 0. Ensuring that any path $\mathcal{P}$ respects $\sum_{(i,j)\in\mathcal{P}} w_{i,j}^k \leq C^k$ prevents forbidden path $\mathcal{P}^k$ from being taken.

4.2.4 Background on proving NP-Hardness of a problem

The third contribution of this chapter is to study the edge placement problem from the theoretical point of view and to understand the effect of its objective on its complexity. For this reason, we introduce here the necessary background on how NP-hardness can typically be proved. First, recall that decidability problems can be divided in complexity classes, among which there is the class P of polynomially solvable problems and the class NP, of problems which can be verified in polynomial time (*e.g.* for which there exists an algorithm which can decide in polynomial time whether a given solution is correct or not). One of the very famous questions of computer science is whether P = NP, *e.g.* if problems in NP are all polynomially solvable.

Let us now give several definitions.

Definition 2. *NP-Completeness: A problem c is NP-complete if there exists a polynomial reduction (e.g. an algorithm that runs in polynomial time) of any problem in NP to c, and if c itself is in NP*

Definition 3. *NP-hardness: A problem c is NP-hard if it is at least as hard as an NP-complete problem.*

We illustrate the complexity classes with the Euler diagrams of Figure 4.2.

The question is then how to prove the NP-hardness of an optimization problem. This is typically done in two steps:

- First, the optimization problem is turned into its decision version. Typically, this means that instead of asking for a solution which minimizes a given function, we turn the problem into asking whether there exists a solution such that the value of the function to optimize for is less/more than a value k. For example, if the problem at hand is the maximum independent set problem, instead of asking what is the largest set of disconnected nodes in a given graph, the decision version asks whether there exists a set of disconnected nodes in a graph with size at least k.

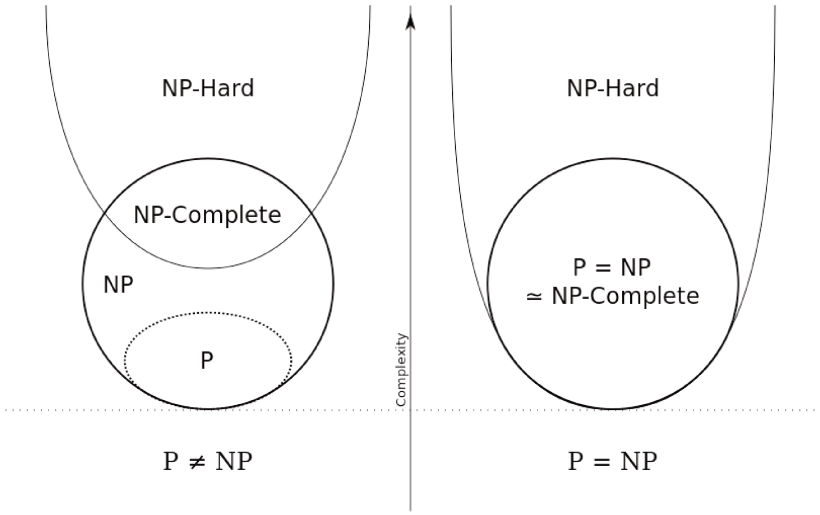


Figure 4.2: Illustration of complexity classes. The left side would hold if P $\neq$ NP while the right side illustrates the case P = NP. Note though not all problems of P or NP are NP-complete hence the $\approx$ sign. Figure from [114]

- Second, we prove that the decision version of the problem is NP-complete.

The question is then how this proof is done. The first problem that has been shown to be NP-Complete is the boolean satisfiability problem (SAT). The proof is based on Turing Machine theory and was given by Cook [115]. We shall not detail it or the definition of the SAT problem here. Based on this initial breakthrough for SAT, Karp [116] exhibited 21 NP-complete problems and devised a systematic method for proving NP-completeness. We shall explain the technique he used here, as it is also the one we shall base our NP-completeness proof on. The technique to prove that a problem c is NP-complete consists in taking an NP-complete problem c', and reducing c' to c. Here, reducing means to show that there exists a polynomial algorithm that, for any instance of c', generates an instance of c such that the two instances are equivalent. This means that the answer to the decision question of the instance of c' is yes if and only if the answer to the decision question of the instance of c is yes. The existence of such a reduction means that c is at least as hard as c', since solving any instance of c' can be done by reducing it to c and then solving the generated instance.

4.3 Contribution 1: Multi-objective multi-constrained shortest path problem with forbidden paths

Before describing the edge placement problem, we start by introducing an algorithm we developped for the multi-constrained, multi-objective shortest path problem with forbidden paths. The reason we introduce this algorithm here is that it will be used as a subroutine in the resolution of the edge placement problem.

4.3.1 Considered problem

Our goal is to find a unique algorithm that can solve any combination of the three problems described in the previous section, *i.e.* a SP algorithm with resource constraints, forbidden paths and multiple objectives. To do so we modify the method from Brumbaugher *et. al.* for the MOSPP to handle the case of resource constraints, hence solving the RCMOSP. Then, we use the reduction of the SPFPP to the RCSP to reduce forbidden path constraints in our problem to resource constraints, which means our RCMOSP algorithm can solve it. We note while [99] have already proposed to reduce Forbidden Path constraints to Resource Constraints, our reduction slightly differs and fixes an inconsistency in their label-setting and node-selection algorithms in the case of the one-to-all version of the problem. The problem is formalized as follows:

We are given a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ along with a set of K weights $\{w_{i,j}^k\}_{k=1}^K$ on each edge $(i,j) \in \mathcal{E}$. The first N weights $\{w_{i,j}^k\}_{k=1}^N$ are intended to be minimized when computing SPs. The other weights $\{w_{i,j}^k\}_{k=N+1}^K$ are intended instead as resources consumed by the SPs, hence their sum cannot exceed as certain available amount $\{C^k\}_{k=N}^K$. A set $\mathcal{F}$ of forbidden paths is given: any path that contains or corresponds to some path of $\mathcal{F}$ cannot be used. We are given a source node $s \in \mathcal{V}$ and our objective is to compute SPs with respect to the first N weights toward any other nodes $t \in \mathcal{V} \setminus \{s\}$. The decision variable is a matrix $\mathbf{X} \in \{0,1\}^{(|V|-1)\times|E|}$ with elements $x_{i,j}^t \forall t \in \mathcal{V} \setminus \{s\}, (i,j) \in \mathcal{E}$, indicating, for each edge, whether it belongs to a SP to t. We call $\mathcal{P}(\mathbf{X}, t)$ the set of selected paths (to any node t of $\mathcal{V} \setminus \{s\}$) derived from the values $x_{i,j}^t$ that are equal to 1. The optimization problem

can be formalized as follows:

$$\min_{\mathbf{x}} \left\{ \sum_{(i,j)\in\mathcal{E}} x_{ij}^t \cdot w_{i,j}^k \right\}_{k=1}^{N} \tag{4.18}$$

$$\text{s.t.} \sum_{(i,j)\in\mathcal{E}} x_{ij}^t - \sum_{(j,i)\in\mathcal{E}} x_{ji}^t = \delta_i \qquad \forall i \in \mathcal{V} \tag{4.19}$$

$$\delta_i = \begin{cases} 1 & \text{if i=s} \\ -1 & \text{if i = t} \\ 0 & \text{otherwise} \end{cases} \qquad \forall i \in \mathcal{V}, \forall t \in \mathcal{V} \setminus \{s\} \tag{4.20}$$

$$\sum_{i,j\in\mathcal{E}} x_{ij}^t \cdot w_{i,j}^k \leq C^k \qquad \text{for } k = N+1, \ldots, K \tag{4.21}$$

$$\mathcal{P}' \nsubseteq \mathcal{P} \qquad \forall \mathcal{P}' \in \mathcal{F}, \mathcal{P} \in \mathcal{P}(\mathbf{X}, t), t \in \mathcal{V} \setminus \{s\} \tag{4.22}$$

$$x_{ij}^t \in \{0, 1\} \qquad \forall (i,j) \in \mathcal{E} \tag{4.23}$$

Equations (4.19)-(4.20) ensure that all paths go from the source to their intended destination. Constraint set (4.21) expresses the resource constraints. Equation (4.22) prevents selected paths to include any forbidden paths. The objective (4.18) implies that, for any destination $t \in \mathcal{V} \setminus \{s\}$, set $\mathcal{P}(\mathbf{x}, t)$ contains exactly all the paths from s to t that are not dominated: for any path $\sqrt{} \in \mathcal{P}(\mathbf{x}, t)$ no other path $\sqrt{}''$ from s to t can exist that jointly satisfies constraints (4.19)-(5.9) and also minimizes the objective (4.18) along all dimensions $k = 1, \ldots, N$. In short, $\mathcal{P}(\mathbf{x}, t)$ is the set of Pareto-optimal paths from s to t.

Similar to [99], we transform the forbidden path constraints (4.22) into additional resource constraints. Let us enumerate the set of forbidden paths $\mathcal{F} = \{\mathcal{P}^f\}_{f=1}^{|\mathcal{F}|}$. We introduce additional weights $\{w_{i,j}^{K+f}\}_{f=1}^{|\mathcal{F}|}$ for any edge $(i,j) \in \mathcal{E}$. We set $w_{i,j}^{K+f}$ to 1 if (i,j) is part of forbidden path $\mathcal{P}^f$ and to 0 otherwise. We set capacity C^{K+f} to the number of arcs forming forbidden path $\mathcal{P}^f$. Constraints (4.22) thus becomes:

$$\sum_{(i,j)\in\mathcal{E}} x_{ij}^t \cdot w_{i,j}^{K+f} \leq C^{K+f}, \quad \text{for } f = 1, \ldots, |\mathcal{F}|. \tag{4.24}$$

The resulting problem (4.18)-(4.21),(5.9),(4.24) is a multiobjective integer linear program which we shall now attempt to solve by using a label-correcting algorithm.

4.3.2 Algorithm

Let us denote with $\mathbf{w}_{i,j} = (w_{i,j}^k)_{k=1,\ldots,K,K+1,\ldots,K+|\mathcal{F}|} \in \mathbb{R}_+^{K+|\mathcal{F}|}$ the vector of weighs on edge (i,j). The corresponding vector of a path is thus

$\mathbf{w}_\mathcal{P} = \sum_{(i,j)\in\surd} \mathbf{w}_{i,j}$. Our algorithm is a modification of the one used by [108]. For each node $n \in \mathcal{V}$, we maintain a set $\mathcal{D}(n)$ of *labels*. Each label is the weight vector $\mathbf{w}_\mathcal{P}$ of a path from s to n. Since we are solving a multiobjective problem, each set $\mathcal{D}(n)$ can contain multiple labels. The aim of the algorithm is to populate $\{\mathcal{D}(n)\}_{n\in\mathcal{V}}$ with the labels of non-dominated paths.

At the beginning, all $\mathcal{D}(n) = \emptyset$, except $\mathcal{D}(s)$ that has one single label, corresponding to the dummy path starting from s with no edges. A node n becomes "labeled" if $\mathcal{D}(n) \neq \emptyset$. At each iteration, a node $i \in \mathcal{V}$ is popped from the list of labeled nodes. Then, for each of its $j \in \mathcal{V}$ neighbors, we check if concatenating a path from $\mathcal{D}(i)$ with arc $(i, j) \in \mathcal{E}$ results in a new path to j, which is currently non-dominated[3]. If that is the case, we update set $\mathcal{D}(j)$ to include the new non-dominated paths toward j and remove the old dominated ones, which is done with the Merge$(\cdot)$ operation. In that case, we also add j to the list of labeled nodes, so as to traverse it again to improve the paths passing through it. We treat the additional constraints using the Filter function (line 8 of Algorithm 9). This function checks whether, for each label of $\mathcal{D}_M(j)$, any constraint is violated. If it is the case, the label cannot lead to a valid path and it is removed from $\mathcal{D}_M(j)$. This is the idea applied by [99] in the context of the FPSPP. However, in the case where the solution is required to give a one-to-all solution (which they claim to do), their scheme is flawed: in effect, if we had a forbidden path $a - b - c$ which was the subpath of $a - b - c - d$, then $a - b - c - d$ would not be reachable with the afforementionned procedure because the constraint would be violated. Our solution to solve this issue is to preprocess the graph beforehand, by adding one dummy node d_t per node $t \in V \setminus \{s\}$, with a single arc (t, d_t) (with weights all set to 0 except the ones corresponding to the forbidden path constraints of t). The forbidden paths to each t are also augmented with d_t. In this way, we are sure that forbidden paths are not subpaths of other paths. Note forbidding subpaths is still possible by using more general resource constraints. Now that we have this solution, we are ready to evaluate its performance, and mainly its run-time.

4.3.3 Computational experiments

We evaluate our algorithm on graphs generated using the NetMaker procedure [117]. This generator first generates a cycle out of all the nodes of the

[3]When checking for domination, we check if the path is dominated with regards to the objectives but also to the constraints. The reason is maybe p dominates p' regarding objectives but they are respective subpaths of q, q' and q violates a constraint but not q'

Algorithm 9 Label-correcting algorithm for RCMOSPP

Input: Graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$, source node $s \in \mathcal{V}$
Output: Labels corresponding to non-dominated paths

Let $\mathcal{D}(s) = \{(0, ..., 0)\}$, $\mathcal{D}(n) = \emptyset \ \ \forall n \in \mathcal{V} - \{s\}$.
Labeled $\leftarrow$ {s}
while Labeled $\neq \emptyset$ **do**
    Choose i from Labeled
    for $j \in$ outneighbors(i) **do**
        $\mathcal{D}'(j) \leftarrow \mathcal{D}(i) \bigoplus \mathbf{w}_{i,j}$ // $\bigoplus$ means summing the weights $\mathbf{w}_{i,j}$ with each individual label of $\mathcal{D}(i)$
        $\mathcal{D}_M(j) \leftarrow$ Merge$(\mathcal{D}(j), \mathcal{D}'(j)$ // update set $\mathcal{D}(j)$ to include the new non-dominated paths toward j and remove the old dominated ones)
        Filter$(\mathcal{D}_M(j), j)$ // remove paths that violate a constraint
        if $D_M(j) \neq \mathcal{D}(j)$ **then**
            $\mathcal{D}(j) \leftarrow D_M(j)$
            Labeled $\leftarrow$ Labeled $\bigcup\{j\}$
        end if
    end for
end while

graph, and then adds arcs such that a node cannot be linked with another one that is further away than a certain distance parameter. Skriver and Andersen empirically showed that combining this procedure with random weight initialization gives instances with large Pareto-fronts. We implement the algorithm as well as the generator in Julia and run it on a machine with $16GB$ of ram and an intel $i7 - 11850H$ CPU running on Fedora 35.

In our experiment, we evaluate the run-time of the algorithm with two objectives, under different number of constraints (0 to 3) and of graph sizes (between 50 nodes and 500). The results are depicted in Figure 4.3. What we observe is that, while the run-time tends to increase with the size of the graph, the most important factor tends to be the number of constraints, with up to an order of magnitude of difference in the run-time for each additional constraint once the graphs are large enough (300 - 400 nodes). In the second experiment, we investigate the impact of including forbidden path constraints. To do so, we take a NetMaker graph with 100 nodes and iteratively add forbidden paths selected randomly among the SPs of the graph (which were first found without any constraint) and evaluate the run-time under different number of objectives (2 to 4). The run-time is depicted in figure 4.4 and the size of the Pareto-front is in Figure 4.5. Observe that while the number of objectives is important for run-time, the number of forbidden paths considered is also key. We also observe a very large positive correlation between the run-time and the size of the Pareto-front.

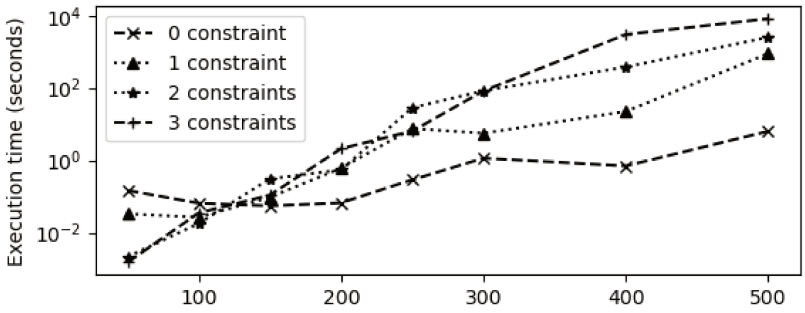


Figure 4.3: Runtime with Various Graph sizes.

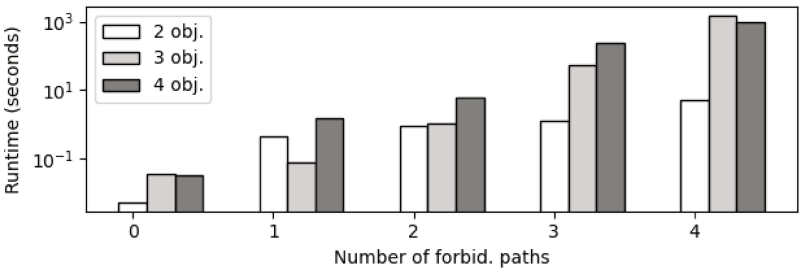


Figure 4.4: Runtime with different number of objectives and forbidden paths

Our general conclusion from those two experiments is then quite counter-intuitive: while one could expect that adding constraints (be it forbidden paths or resource constraints) would reduce the number of admissible paths and hence the size of the solution space, which we expected would reduce the solving time, we find out the opposite is true. Indeed, it seems that adding constraints increases the number of non-dominated paths (*e.g.* the size of the Pareto-front), which is clearly correlated to the run-time growth.

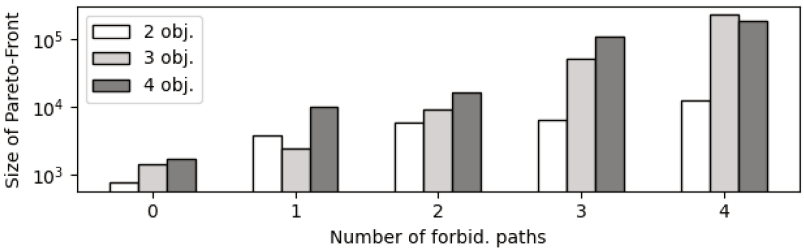


Figure 4.5: Size of Pareto-front with different number of objectives and forbidden paths

Notation	Description
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Physical network with nodes $\mathcal{V}$ and edges $\mathcal{E}$
CPU_{v_i}	CPU capacity of node v_i
BW_{v_i,v_j}	Bandwidth capacity of edge (v_i, v_j)
D_{v_i,v_j}	Propagation delay of edge (v_i, v_j)
$\mathcal{P} \subseteq \mathcal{E}$	Path of graph $\mathcal{G}$
$D_{prop}(\mathcal{P})$	Propagation delay on path $\mathcal{P}$
$r \in \mathcal{R}$	Slice r in set of all slices $\mathcal{R}$
D_r	Maximum latency of slice r
SL_r	Reliability level of slice r
OPS_r	Number of operations for application of slice r
$\mathcal{B}_r$	Set of base stations from which service requests arrive
$k \in \mathcal{K}_r$	flow k in set of flows $\mathcal{K}_r$ of slice r
s_k	source of flow k
AT_k	Inter-arrival time distribution of user requests
$T_k \subset V$	candidate computing nodes
BW_k^d	Bandwidth required
$\mathcal{K}$	set of all flows of all slices
cpu_k	decision variable equal to the amount of computing power allocated to flow k
μ_k	service rate of flow k
$D_{tot}^k(\mathcal{P}, \mu_k)$	Total delay on pat $\mathcal{P}$
$D_{wait}(\mu_k, AT_k)$	Waiting time of requests due to computations
pow(v_i)	Average power consumed by node v_i
l(v_i)	Average utilization of node v_i
$e(v_i)$	Power consumed when v_i is idle
$c(v_i)$	Slope of the function associating $l(v_i)$ to consumed power
$\mathbb{P}_{idle}^k$	Probability that software component of flow k is idle
$f_k(\mathcal{P})$	Binary variable deciding whether path $\mathcal{P}$ is used by flow k
$\mu_k(\mathcal{P})$	Continuous variable deciding value of μ_k if $\mathcal{P}$ was to be used
$cpu_k(\mathcal{P})$	Continuous variable deciding value of cpu_k if $\mathcal{P}$ was to be used
$\boldsymbol{\mathcal{P}}_k$	Set of all paths suitable to route flow k
$\delta_{v_i,v_j}(\mathcal{P})$	Indicator function indicating if (v_i, v_j) is in $\mathcal{P}$
$\phi_{v_i}(\mathcal{P})$	Indicator function indicating if v_i is the last node of $\mathcal{P}$
$\omega_{v_i}(\mathcal{P})$	Indicator function indicating if v_i is part of $\mathcal{P}$
$a(v_i)$	Binary variable indicating whether node v_i is activated
OA_{v_i}	Set of outer arcs of v_i

Table 4.1: Notation

4.4 Contribution 2: Edge Service placement

In this section, we describe the main contribution of this chapter. We formulate the problem of placing edge services as a non-linear programming problem, which we then linearize. Then, in order to solve the linearized version, which has exponentially many variables, we leverage the previous Multi-Objective Shortest Path algorithm in order to design a Column Generation algorithm, with the idea that CG enables us to solve the problem optimally, while only solving for a fraction of the variables. We summarize the notations in Table 4.1.

4.4.1 Graph model and assumptions

Let us now turn to the main problem of this chapter, *e.g.* the placement of Edge Services with the objective of minimizing the energy consumption.

We consider the system depicted in Figure 4.1. The network is owned by a Physical Network Operator (PNO) and is modeled as a graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$. Nodes $\mathcal{V}$ can be of three types: ingress nodes, computing nodes and routers. Ingress nodes are entry points of the network (e.g., base stations (BS)) where traffic from a set of Mobile Virtual Network Operators (MVNOs) arrives. Computing nodes can host edge and cloud services to perform computation tasks (e.g., computer vision, video transcoding), and routers steer traffic throughout the network.

Each node $v_i \in \mathcal{V}$ of the graph is equipped with a capacity CPU_{v_i} of CPU resources. Computing nodes can host services and the PNO can allocate portion of such CPU resources to execute such services. Router and ingress nodes do not host any service and only execute networking functionalities, thus they can be modeled as nodes with 0 CPU capacity, *i.e.*, $CPU_{v_i} = 0$.

Nodes are connected via links, which are represented by the edges $\mathcal{E} = \mathcal{V} \times \mathcal{V}$ of the graph. For each link $(v_i, v_j) \in \mathcal{E}$ connecting nodes v_i and v_j, we assume a maximum bandwidth capacity BW_{v_i,v_j}. Each link $(v_i, v_j) \in \mathcal{E}$ is characterized by its propagation delay D_{v_i,v_j}. Let us define a path $\mathcal{P} \subseteq \mathcal{E}$ as a sequence of links connecting distinct nodes of the graph. The propagation delay on a path $\mathcal{P}$ is

$$D_{prop}(\mathcal{P}) = \sum_{(v_i,v_j)\in\mathcal{P}} D_{v_i,v_j}. \tag{4.25}$$

We consider a set $\mathcal{M}$ of MVNOs willing to leverage the physical infrastructure to deploy and offer a variety of services (e.g., AR/VR, video streaming and autonomous driving) to their customers. Services offered by MVNOs have different performance requirements.

To ensure isolation, the PNO allocates a dedicated edge slice $r \in \mathcal{R}$ to each MVNO, where $\mathcal{R}$ being the set of all edge slices. If the same MVNO is allocated multiple edge slices, we will model them as virtually separate MVNOs. For simplicity, we assume each MVNO only runs one software component. We define software components as virtualized programs which are run on the computational nodes and can serve requests for the users of the edge slice they belong to.

For any given network edge slice r, the corresponding MVNO specifies a SLA as follows:

- A maximum tolerable latency value D_r (in ms).
- A reliability level SL_r indicating the minimum fraction of requests that should be satisfied within D_r. For example, if $D_r = 2ms$, $SL_r = 0.95$, the MVNO requires 95% of its subscribers' requests to be served within $2ms$.

- OPS_r, the number of floating point operations to perform by software component of edge slice r. OPS_r is a random variable and we assume the MVNO communicates the parameters describing its probability distribution.

- $\mathcal{B}_r \subset \mathcal{V}$: base stations from which service requests arrive.

Since each edge slice request r can include base stations located at geographically different locations, we decompose the slice into a set $\mathcal{K}_r$ of $|\mathcal{B}_r|$ flows, one for each base station. Figure 4.1 shows an example with two edge slices. Edge slice A has two flows, edge slice B has one flow. A flow $k \in \mathcal{K}_r$ is defined by the following elements:

- The ingress/source node, $s_k \in V$

- The inter-arrival time distribution AT_k of user requests.

- The set $T_k \subset V$ of candidate computing nodes that can produce feasible solutions for flow k and can thus host services for k. Let $\mathcal{P}_k$ be the set of candidate paths with starting node s_k and any destination $t_k \in T_k$. These are all the loopless paths which have a propagation delay lower that D_r.

- The amount of bandwidth BW_k^d requested.

Let us denote with $\mathcal{K} = \bigcup_r \mathcal{K}_r$ the set of all flows requested by MVNOs. We assume that each flow $k \in \mathcal{K}_r$ is assigned a single SLA corresponding to edge slice r and defined as the tuple (D_r, SL_r). Since flows $k \in \mathcal{K}_r$ belong to the same edge slice r, the SLA of r applies to all $\mathcal{K}_r$. Therefore, we set $D_k = D_r$, $SL_k = SL_r$ and $OPS_k = OPS_r$ for all $k \in \mathcal{K}_r$.

The PNO aims to (i) serve all flows $k \in \mathcal{K}$ while satisfying the SLA requirements (which we consider to be constraints); and (ii) use the least amount of energy (which is the objective function). This is done by jointly placing the software components, allocating CPU to them, and deciding which path to use for routing requests from the BS to the software component.

We now introduce our assumptions.

Assumption 1. *Each flow k is routed between its source s_k and its chosen destination $t_k \in T_k$ on a single path.*

This assumption is justified by considering that using multiple routes for a single flow can introduce jitter, which may be undesirable for services with tight latency constraints such as the ones we consider in this chapter.

Assumption 2. *We assume that OPS_k is exponentially distributed. Decision variable cpu_k is the amount of CPU that the PNO decides to allocate to the software component of flow k, expressed in floating point operations per second (FLOPS).*

The time necessary to process a service request of flow k (e.g., the time spent by the software component of that flow to process the instructions in the request) is exponentially distributed with mean $E(OPS_k)/cpu_k$.

The next proposition naturally follows.

Proposition 5. *OPS_k/cpu_k The service time of the request is exponentially distributed with mean $\mathbb{E}[OPS_k]/cpu_k$. Therefore, the software component of each flow k can be modeled as a $G/M/1$ queue where AT_k is the arrival process and $\mu_k = cpu_k/\mathbb{E}[OPS_k]$ is the mean service rate.*

Assumption 3. *Queuing delay on links is negligible, and the total delay for flow k using path $\mathcal{P} \in \boldsymbol{\mathcal{P}}_k$ to reach destination node t_k is*

$$D^k_{tot}(\mathcal{P}, \mu_k) = 2 \cdot D_{prop}(\mathcal{P}) + D_{wait}(\mu_k, AT_k), \tag{4.26}$$

where $D_{wait}(\mu_k, AT_k)$ is a random variable representing the waiting time of any request at the software component Prop. 5) and $D_{prop}(\mathcal{P})$ is the propagation delay (4.25).

This assumption, common in recent works [118, 69, 8], is reasonable since we consider that the PNO reserves to each flow the amount of bandwidth demanded by the respective MVNO on the entire route. We assume MVNOs request a sufficient amount of bandwidth, so that large queuing times on such links are unlikely to happen.

Assumption 4 (Constant Energy for Network processing)**.** *The energy consumed for routing is constant regardless of the BW.*

This assumption has been demonstrated experimentally for routers [119] and for dedicated network cards [120] [121].

We are now ready to derive the energy model of the system, which will then be used to formulate the optimization problem.

First of all, our model is based on the fact that it has empirically been shown that the energy consumption of a server can be represented as an affine function of the cpu utilization [122]. We note this affine power function for node v_i as

$$\text{pow}(v_i) = c(v_i) \cdot l(v_i) + e(v_i),$$

where $e(v_i)$ is the power consumed when node v_i is idle (the y-intercept of $\text{pow}(v_i)$), $c(v_i)$ is the slope, and $l(v_i) = \mathbb{E}[\text{load}(v_i)]$ is the average utilization

of the node (which will be computed later).[4] We assume $c(v_i)$ and $e(v_i)$ are known for each type of node, hence in the remaining we focus on calculating the load. We note that more recent sources point that other factors such as memory can have an impact [123]. However, we observe in the cited work that the CPU ratio is already a fairly accurate predictor of the energy consumption, although not the best one. Since we were unable to find studies that precisely analyze the impact of memory or other factors on the global energy consumption, and since, as mentionned, CPU is already a pretty good predictor, memory consumption or other factors is outside the scope of this chapter. Let us denote the probability that the software component associated with flow k is idle as $\mathbb{P}^k_{idle}$.

Recall each software component can be modeled as a G/M/1 queue (see **Proposition 5**). Hence, the average throughput of the queue modeling software component of flow k is:

$$\mathbb{E}[\text{throughput}(k)] = \mu_k \cdot (1 - \mathbb{P}^k_{idle}) = \mathbb{E}[AT_k]^{-1}$$

because in G/M/1 queues we have $1 - \mathbb{P}^k_{idle} = \frac{\mathbb{E}[AT_k]^{-1}}{\mu_k}$ [124] (as long as the queue is stable *e.g.* $\mathbb{E}[AT_k]^{-1} \leq \mu_k$). From **Assumption 2**, we derive the average amount of cpu consumed by the software component:

$$cpu_{avg} = \mathbb{E}[\text{throughput}(k)] \cdot \mathbb{E}[OPS_k] \tag{4.27}$$

from which we can derive expected the percentage of CPU utilization $l(k, v)$ induced by a single software component:

$$l(k, v_i) = \frac{cpu_{avg}}{CPU_{v_i}} = \frac{\mathbb{E}[AT_k]^{-1} \cdot \mathbb{E}[OPS_k]}{CPU_{v_i}} \tag{4.28}$$

Finally, since the queues are independent, the expected load on a node v_i hosting a set $\mathcal{S}$ of software components is:

$$l(v_i) = \mathbb{E}[\text{load}(v_i)] = \sum_{k \in \mathcal{S}} l(k, v_i) \tag{4.29}$$

4.4.2 Integer Programming Problem formulation

The problem of placing all edge slices' flows while minimizing energy consumption and respecting SLAs has decision variables $\mathbf{x} = \{a(v_i), l(v_i) \ \forall v_i \in \mathcal{V}, \quad f_k(\mathcal{P}), \mu_k(\mathcal{P}), cpu_k(\mathcal{P}) \quad \forall k \in 1 \dots K, \quad \mathcal{P} \in \boldsymbol{\mathcal{P}}_k\}$. $f_k(\mathcal{P})$ variables indicate whether path $\mathcal{P}$ is used by commodity k, hence it decides for the routing, while $\mu_k(\mathcal{P})$ and $cpu_k(\mathcal{P})$ decide respectively for the service rate and amount of CPU of commodity k, *e.g.* they determine dimensionning. They also decide placement as if they are equal to 0 it implies the software

[4] We note the power is easily converted into an energy measured in watt-hour

component is not placed on the node. Note that the quantities cpu_k and μ_k have been replaced with their counterparts, *i.e.*, $cpu_k(\mathcal{P}), \mu_k(\mathcal{P})$. Indeed, those quantities depend on the path $\mathcal{P}$ we select to route flow k, as if path $\mathcal{P}$ is shorter (in terms of delay), we can afford to have a lower service rate $\mu_k(\mathcal{P})$ by giving less $cpu_k(\mathcal{P})$ to the software component of flow k. Moreover, $cpu_k(\mathcal{P})$ and $l(k, v_i)$ can be calculated from $\mu_k(\mathcal{P})$ using (4.28) and Prop. 1. We also denote by $\delta_{v_i,v_j}(\mathcal{P})$, $\phi_{v_i}(\mathcal{P})$, $\omega_{v_i}(\mathcal{P})$ the indicator functions which respectively indicate if (v_i, v_j) is in path $\mathcal{P}$, if v_i is the last node of $\mathcal{P}$ and if v_i is part of path $\mathcal{P}$. $\delta_{v_i,v_j}(\mathcal{P})$, $\phi_{v_i}(\mathcal{P})$, $\omega_{v_i}(\mathcal{P})$ are input parameters of the problem.

$$\min_{\mathbf{x}} \quad \sum_{v_i \in \mathcal{V}} a(v_i) \cdot e(v_i) + \sum_{v_i \in \mathcal{V}} l(v_i) \cdot c(v_i) \tag{4.30}$$

$$\text{s.t.} \quad \sum_{1 \leq k \leq K} \sum_{\mathcal{P} \in \boldsymbol{\mathcal{P}}_k} f_k(\mathcal{P}) \cdot BW_k^d \cdot \delta_{v_i,v_j}(\mathcal{P}) \leq BW_{v_i,v_j} \quad \forall (v_i, v_j) \in \mathcal{E} \tag{4.31}$$

$$\sum_{\mathcal{P} \in \boldsymbol{\mathcal{P}}_k} f_k(\mathcal{P}) = 1 \qquad \forall k \in 1...K \tag{4.32}$$

$$\sum_{1 \leq k \leq K} \sum_{\forall \mathcal{P} \in \boldsymbol{\mathcal{P}}_k} f_k(\mathcal{P}) \cdot \phi_{v_i}(\mathcal{P}) \cdot cpu_k(\mathcal{P}) \leq CPU_{v_i} \qquad \forall v_i \in \mathcal{V} \tag{4.33}$$

$$a(v_i) \geq \frac{1}{K} \sum_{1 \leq k \leq K} \sum_{\mathcal{P} \in \boldsymbol{\mathcal{P}}_k} f_k(\mathcal{P}) \cdot \omega_{v_i}(\mathcal{P}) \qquad \forall v_i \in \mathcal{V} \tag{4.34}$$

$$l(v_i) = \sum_{1 \leq k \leq K} \sum_{\forall \mathcal{P} \in \boldsymbol{\mathcal{P}}_k} f_k(\mathcal{P}) \cdot \phi_{v_i}(\mathcal{P}) \cdot l(k, v_i) \qquad \forall v_i \in \mathcal{V} \tag{4.35}$$

$$\mu_k(\mathcal{P}) \in \left\{ x | x \in \mathbb{R}, \mathbb{P}(D_{\text{tot}}^k(\mathcal{P}, x) \leq D_k) \geq SL_k \right\} \tag{4.36}$$

$$\forall k \in 1 \ldots K, \mathcal{P} \in \boldsymbol{\mathcal{P}}_k$$

$$f_k(\mathcal{P}) \in \{0, 1\} \qquad \forall f_k(\mathcal{P}) \tag{4.37}$$

$$a(v_i) \in \{0, 1\} \qquad \forall v_i \in \mathcal{V} \tag{4.38}$$

$$l(v_i) \geq 0 \qquad \forall v_i \in \mathcal{V} \tag{4.39}$$

The objective (4.30) of the PNO is to minimize energy consumption, by turning off nodes (setting $a(v_i) = 0$) and reducing their dynamic energy $l(v_i)$. Constraints (4.31) prevents from using more bandwidth than what is available. Constraints (4.32) ensure we select exactly one path per flow k among those in the candidate set $\boldsymbol{\mathcal{P}}_k$. Constraints (4.33) enforce CPU capacities. Constraints (4.34) ensure that any node that is used by a path of software component is turned on. Constraints (4.35) correspond to (4.29). Because of this constraint, if $a(v_i) = 0$, then $l(v_i) = 0$, *e.g.* when a node is turned off, its average load must be 0. Finally, Constraints (4.36) are non-linear and enforce that given a chosen path and a request arrival distribution, the service rate $\mu_k(\mathcal{P})$ is high enough to accommodate the SLA constraints. In order to potentially accommodate software components of many flows in single nodes (which allows to turn more nodes off and save

energy), Constraints (4.33) suggest to have cpu_k as small as possible for any flow k. However, we cannot reduce cpu_k too much, otherwise $D_{\text{wait}}(\mu_k, AT_k)$, and thus $D^k_{\text{tot}}(\mathcal{P}, \mu_k)$ (Eq. (4.26)), would excessively increase and violate the SLA (4.36). Therefore, $\forall k \in 1 \dots K, \mathcal{P} \in \boldsymbol{\mathcal{P}}_k$, we can replace (4.36) with the optimal value of cpu_k, *e.g.* such that

$$\mu_k(\mathcal{P}) = \min_x \left\{ x | x \in \mathbb{R}, \mathbb{P}(D^k_{\text{tot}}(\mathcal{P}, x) \leq D_k) \geq SL_k \right\} \tag{4.40}$$

4.4.3 Proposed Solution

In this section, we shall detail the solution to the afforementionned Non-linear programming problem. As shown in Figure 4.6, the section is organized as follows: we first simplify the problem as a Mixed Integer Programming problem (Section 4.4.3.1), then, we detail how to make this MILP formulation more tractable in section 4.4.3.2. The solution to its linear relaxation using column generation is then presented in section 4.4.3.3. Finally, we use this linear relaxation solution to solve the integer version of the problem in section 4.4.3.4

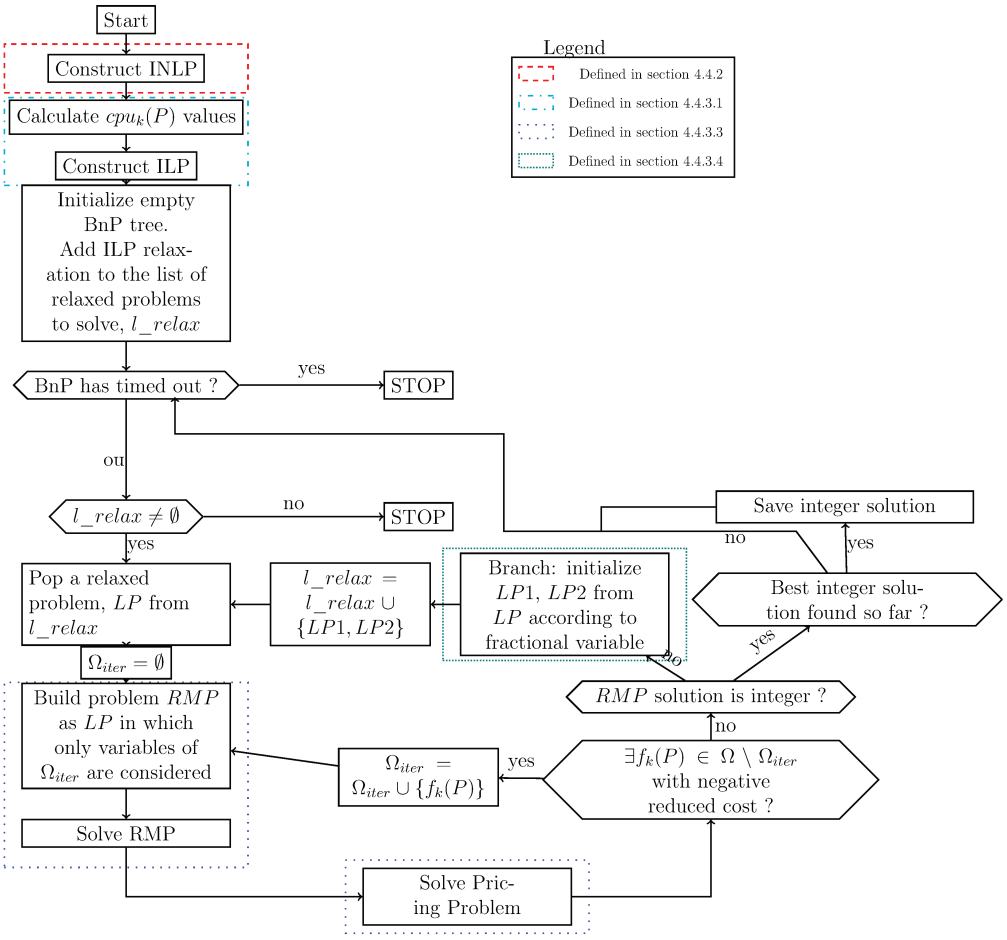


Figure 4.6: Block diagram of the exact solution algorithm

4.4.3.1 Simplification to a MILP

The goal of this section is to demonstrate that we can simplify the non linear problem(4.30)-(4.39) and cast it as a MILP while maintaining optimality. Theorem 1 is instrumental to demonstrate our claim.

Theorem 1. *If the requests from a flow k have general independent inter-arrival times and exponentially distributed request sizes, given a path $\mathcal{P} \in \boldsymbol{\mathcal{P}}_k$, the optimal service rates (4.40) can be computed in polynomial time.*

Sketch of Proof. Equation (4.40) suggests that we need to compute the minimum valid amount of CPU to satisfy SLA constraints. This minimum CPU value is a strictly increasing function of the propagation delay of the path considered (intuitively, the longer the path, the faster we need to process flow requests into software components to compensate for higher path latency). Finding the smallest satisfactory service rate then comes down to solving a root-finding problem with a unique root (the uniqueness

is a property of the delay distribution of G/M/1 queues), which can be pre-computed using a bisection algorithm in polynomial time. □

Thanks to Theorem 1, we can pre-compute all values of $cpu_k(\mathcal{P})$ and take them as input of the following MILP, thus removing the non-linear constraints (4.36) in which the decision variables are $\mathbf{x}' = \{a(v_i), l(v_i) \ \forall v_i \in V, \ \ f_k(\mathcal{P}) \ \ \forall k \in 1 \dots K, \ \ \mathcal{P} \in \boldsymbol{\mathcal{P}}_k\}$.

$$\begin{aligned} \min_{\mathbf{x}'} \quad & \sum_{v_i \in V} a(v_i) \cdot e(v_i) + \sum_{v_i \in V} l(v_i) \cdot c(v_i) \\ \text{s.t.} \quad & (4.31), (4.32), (4.33), (4.34), (4.35) \end{aligned} \tag{4.41}$$

4.4.3.2 Making the MILP more tractable

An issue with this formulation is that once we relax it into a linear program (LP), both variables $a(v)$ and $f_k(P)$ become fractional, and we would have to branch [86] on both variables to find an integer solution. Our early experiments have shown this makes even small problems unsolvable in a reasonable amount of time. Therefore, we introduce the following set of constraints which ensures that we only have to branch on $f_k(P)$

$$a(v_i) \geq f_k(\mathcal{P}) \cdot \omega_{v_i}(\mathcal{P}), \forall v_i \in \mathcal{V}, k = 1 \dots K, \mathcal{P} \in \boldsymbol{\mathcal{P}}_k. \tag{4.42}$$

These constraints state that if any binary variable $f_k(\mathcal{P})$, tied to path $\mathcal{P}$, which uses node v_i, is equal to 1, variable $a(v_i)$ has to be 1. Otherwise if no path uses v_i, $a(v_i)$ will be set to its lowest possible value (*e.g.* 0) due to the objective function.

4.4.3.3 Column Generation-based solution

We first solve the LP relaxation of ILP eq. (4.41) (augmented with constraints 4.42), as it will later be embedded in a BnP procedure (see section 4.2.2.5). It could be tempting to solve LP using generic techniques (*e.g.*, the simplex algorithm). However, at worst we have one variable $f_k(\mathcal{P})$ per loopless path $\mathcal{P}$ between each source and destination of each flow k. It is well known [4] that the number of paths in a graph is exponential in its number of nodes, making it impossible to enumerate the set of path variables $\Omega = \bigcup_{\forall k \in \mathcal{K}, \forall \mathcal{P} \in \boldsymbol{\mathcal{P}}_k} f_k(\mathcal{P})$ and solve the LP relaxation explicitly in reasonable time. Instead, we resort to *column generation (CG)* (see section 4.2.2.3). By using CG we only consider a sequence of Reduced Master Problems (*RMP*s)). Each RMP(Ω_l) comprises only a subset of the path variables $\Omega_l \subset \Omega$ of the full LP and "excludes" all the others, implicitly forcing them to 0. In our case, we start with $\Omega_0 = \emptyset$. Then, at every iteration l,

we select one variable $f_k(\mathcal{P}^*_{(l)})$ among $\Omega \setminus \Omega_l$ to form $\Omega_{l+1} = \Omega_l \bigcup \{f_k(\mathcal{P}^*_{(l)})\}$. At the l-th iteration of CG, variable $f_k(\mathcal{P}^*_{(l)})$ we choose to add is the one with the smallest "reduced cost". The reduced cost of each variable $f_k(\mathcal{P})$ measures how much the objective function would change if $f_k(\mathcal{P})$ was to be increased of an infinitesimal amount. Intuitively, if the reduced cost is positive, increasing the variable will increase the objective function, and if it is negative, it will decrease it. Hence by always adding variables with negative reduced cost at each step, optimality is guaranteed [86]. For more details about CG we report the reader to section 4.2.2.

In the usual LP setting that uses the simplex method (and not CG), all variables are included in the problem and computing reduced costs of any single variable for each intermediate solution is straightforward (see [86], §9) and done in constant time. However, doing so for exponentially many variables like our problem requires is intractable. CG allows to skip this long computation by instead finding the variable with minimum reduced cost over the set of all excluded variables $\Omega \setminus S_l$. Such a problem is called the "pricing" problem and is usually solved by first proving that it is equivalent to another problem for which a tractable algorithm is known. Again, we report to section 4.2.2 for more details on the pricing problem. A key effort of this work is to find such an equivalent—yet easier—problem, which is summarized in the following theorem.

Theorem 2. *The pricing problem is equivalent to a bi-objective shortest path problem (BOSPP) with forbidden paths, which can be solved efficiently (e.g. [125]).*

Sketch of Proof. The first step of our proof is to decompose the reduced cost of $f_k(\mathcal{P})$ into the sum of 3 terms. The former is constant regardless of k and $\mathcal{P}$, the second depends on dual variables associated with edges of P, and the third depends on the CPU consumed if $\mathcal{P}$ was to be used. Since the higher the propagation delay $D_{\text{prop}}(\mathcal{P})$ on a path $\mathcal{P}$, the higher the CPU to be consumed to compensate for high $D_{\text{prop}}(\mathcal{P})$, it follows that minimizing this third term can be done by using the path with shortest delay. On the other hand, minimizing the second objective (which depends on dual variables) can be done by assigning the dual variables associated to edges as weights on such edges and finding the shortest path. Since these two objectives can be conflicting, we solve the pricing problem by computing the whole Pareto frontier for this problem[5], then finding the path with the smallest reduced cost among them. Finally, we prove that Equations (4.42) do not modify the reduced costs of the excluded variables. This is however

[5] *e.g.* by solving the biobjective shortest path problem with weights $w^1_{i,j}$ equal to dual variables associated with edges of (i,j) and weights $w^2_{i,j}$ equal to the propagation delay of the link

not the case for the included variables. For this reason, computing the bi-objective shortest paths while forbidding all path associated with variables that have already been included ensures the Pareto-frontier contains the path with minimum reduced cost, *e.g.* the solution to the pricing problem. □

4.4.3.4 Branching rule

Now that we have a solution to the LP relaxation (summarized up in Fig. 4.7), we can embed it in a branch-and-bound procedure. To do so, we need to use a branching-rule which preserves the pricing problem, *e.g.* which does not make it harder to solve. We choose to use the same branching rule as in [72] which was proposed in the context of the classic multicommodity flow problem. This rule works as follows:

- Find the commodity k which is splitted (*e.g.* that uses more than a single path) and has the largest flow demand BW_k^d;
- Find the two paths $\mathcal{P}, \mathcal{P}'$ which carry the largest amount of flow for k;
- Find the node v_i where $\mathcal{P}$ and $\mathcal{P}'$ diverge. We denote by (v_i, v_j) and (v_i, v'_j) the two different outer arcs used by $\mathcal{P}$ and $\mathcal{P}'$;
- Build the set OA_{v_i} of outer arcs of v_i;
- Partition OA_{v_i} in two roughly equally-sized sets $OA^1_{v_i}$, $OA^2_{v_i}$ such that $OA^1_{v_i}$ contains (v_i, v_j) and $OA^2_{v_i}$ contains (v_i, v'_j);
- Finally, we create two new nodes in the branch-and-bound tree. In the first node, $OA^1_{v_i}$ is forbidden and cannot be used for routing commodity k. In the second one, we forbid $OA^2_{v_i}$. Concretely, each node of the branch-and-bound tree is associated with one copy of the problem. In the first one, we add the constraints $\sum_k \sum_{\mathcal{P} \in \boldsymbol{\mathcal{P}}_k | \exists e \in \mathcal{P} | e \in OA^1_{v_i}} f_k(\mathcal{P}) = 0$ and in the second one $\sum_k \sum_{\mathcal{P} \in \boldsymbol{\mathcal{P}}_k | \exists e \in \mathcal{P} | e \in OA^2_{v_i}} f_k(\mathcal{P}) = 0$

The major advantage of this branching rule is that the constraints added do not modify the pricing problem. We can solve the same BOSPP, but only need to pre-process the graph and set the weights of the forbidden arcs to infinity with respect to both objective functions.

However, it is worth mentioning that this procedure assumes that all paths of a given commodity have the same source and destination. This is not the case in our problem, as a given service could have many different potentially suitable nodes for its execution. This can cause issue if $\mathcal{P}$ is a

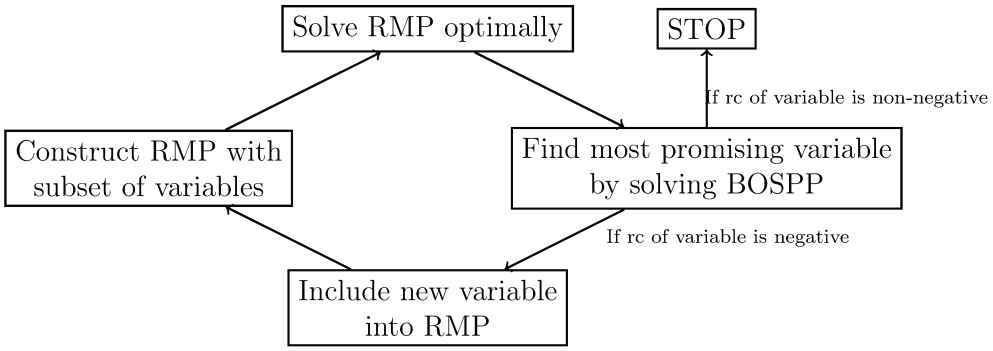


Figure 4.7: Block diagram of the pricing algorithm. Corresponds to "Solve pricing problem" block in Figure 4.6

prefix of $\mathcal{P}'$, in which case branching is not well defined. For this reason, we add a meta-node per commodity. Each meta-node only has inner arcs and no outer arcs. Each of the potential destination of a commodity has an arc towards the meta-node which has infinite capacity. The meta-node consumes zero energy and the CPU consumption happens on the last node of the original path (and not on the meta-node). In this way we ensure branching is still well-defined while not modifying the problem instance.

Finally, note that here, we only branch on the set of path variables. On the other hand, variables $a(v_i)$ also are integers. However, this is not a problem because the integrality of those variables is implied by the integrality of the path variables, thanks to constraints (4.42).

4.4.4 Performance Evaluation and Comparison

We compare our approach Energy Minimization Optimization (**EMO**) to two state-of-the-art strategies. The first one is the MCF CG algorithm from [72]. Since MCF CG minimizes resource utilization, we call it Resource Minimization Optimization (**RMO**) here. Note that we modify the original pricing problem of [72] to take SLA and CPU capacity constraints into account. The second compared approach is **UEPSO** [40], a VNE algorithm to which we also compared our VNE solution in the previous chapter. Since VNE consists in placing a virtual graph onto a physical graph, we transform an instance of our problem into a VNE instance by transforming the set of flows into a virtual requests graph made of disjoint edges (one per flow). Since VNE assumes fixed CPU demands, we set the CPU demands equal to the lowest possible value, obtained by assuming that the path chosen has 0 propagation delay. Hence, UEPSO is evaluated in idealized conditions. We consider three graphs from the TopologyZoo dataset [25], namely Abilene, Agis and INS IXC Services, with 11, 24 and 30 nodes, respectively. Each node is assigned a role, either router or edge

Graph	SLAs	Min BW (mbps)	Max BW (mbps)	Min OPS_k	Max OPS_k	Min D_k (ms)	Max D_k (ms)
Abilene	[0.87,0.9,0.95]	10	30	150	200	600	1000
Agis	[0.87,0.9,0.95]	20	40	150	200	1400	1800
INS IXC	[0.87,0.9,0.95]	20	40	150	200	1400	1800

Table 4.2: Random generation parameters.

c_1^k, c_2^k	a_1^k	${b_1^k}^2$	a_2^k	${b_2^k}^2$
$[0, 1]$	$[2, 4]$	$[1, 2]$	$[3, 20]$	$[5, 12]$

Table 4.3: Random generation parameters of the inter-arrival time laws (same for all graphs). All parameters are selected uniformly random from the given intervals.

node. In Abilene we randomly choose 5 nodes as edge nodes, in Agis, we pick 14 and in IXC Services 16. We randomly generate 30 edge slices per graph. Each edge slice has a single flow. Note that after respectively 27, 21 and 11 edge slices SLA and capacity constraints become infeasible. Simulation details are summarized in Table 4.2. For the inter-arrival time law parameters, we choose to use a mixture of 2 Normal random variables: $c_1^k AT_k^1 + c_2^k AT_k^2$ with $AT_k^1 \sim \mathcal{N}(a_1^k, {b_1^k}^2)$ and $AT_k^2 \sim \mathcal{N}(a_2^k, {b_2^k}^2)$. The parameters of this mixture are chosen uniformly randomly for each flow and are depicted in Table 4.3

We implement our solution in Julia (the code is available at [126]), using the HiGHS solver for the LP solution (branching and CG are done in plain Julia). The algorithms run for a max. of 1 hour.

Figures 4.9 and 4.10 show that resource and energy minimization are contradicting objectives. In order to save energy, we may need to consume more resources. Indeed, EMO saves up to 50% of energy, when load is high (many edge slices), via using more resources than RMO. This counter-intuitive result can be explained by the fact that RMO tends to use shortest paths, which yields to a lower propagation delay and hence a lower CPU consumption. It also tends to select paths with fewer hops, hence consuming less bandwidth. On the other hand, minimizing energy consumption yields to taking slightly longer paths when possible, which might generate more resource consumption, but enable sharing of active nodes: the selected paths will share as many nodes as possible, so as to turn as many nodes off and consume less energy. An example of such a case is depicted in Figure 4.8, in which compared to Figure 4.1, Flow 1 of edge Slice b uses a longer path which enables to turn router R2 off. Those benefits come at the cost of runtime, as EMO is computationally harder to solve than RMO (Fig. 4.11). We relate this difficulty to the aforementioned difference between the two solutions: it is much harder to find the best way to share paths than to find a solution which, at its core, uses a large amount of the shortest paths. For example, on the Agis network, EMO times out for 11 edge slices, while the results of the other methods indicate that the

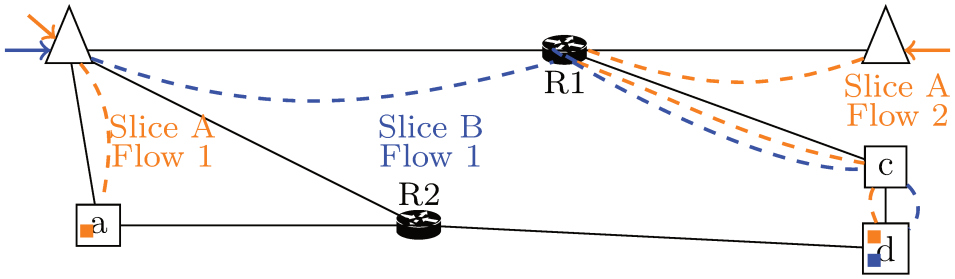


Figure 4.8: Ex. more energy efficient placement (compared to Fig. 4.1).

problem is feasible for at least up to 21 edge slices. On the other hand, for INS IXC Services, RMO obtains feasible solutions near-instantaneously while for the most loaded scenarios, EMO reaches the time-limit. Finally, we note that for all instances and metrics, UEPSO is much worse than both approaches. This is unsurprising since it only decides placement of software components and routing. This confirms the importance of jointly deciding placement, routing and resource dimensioning.

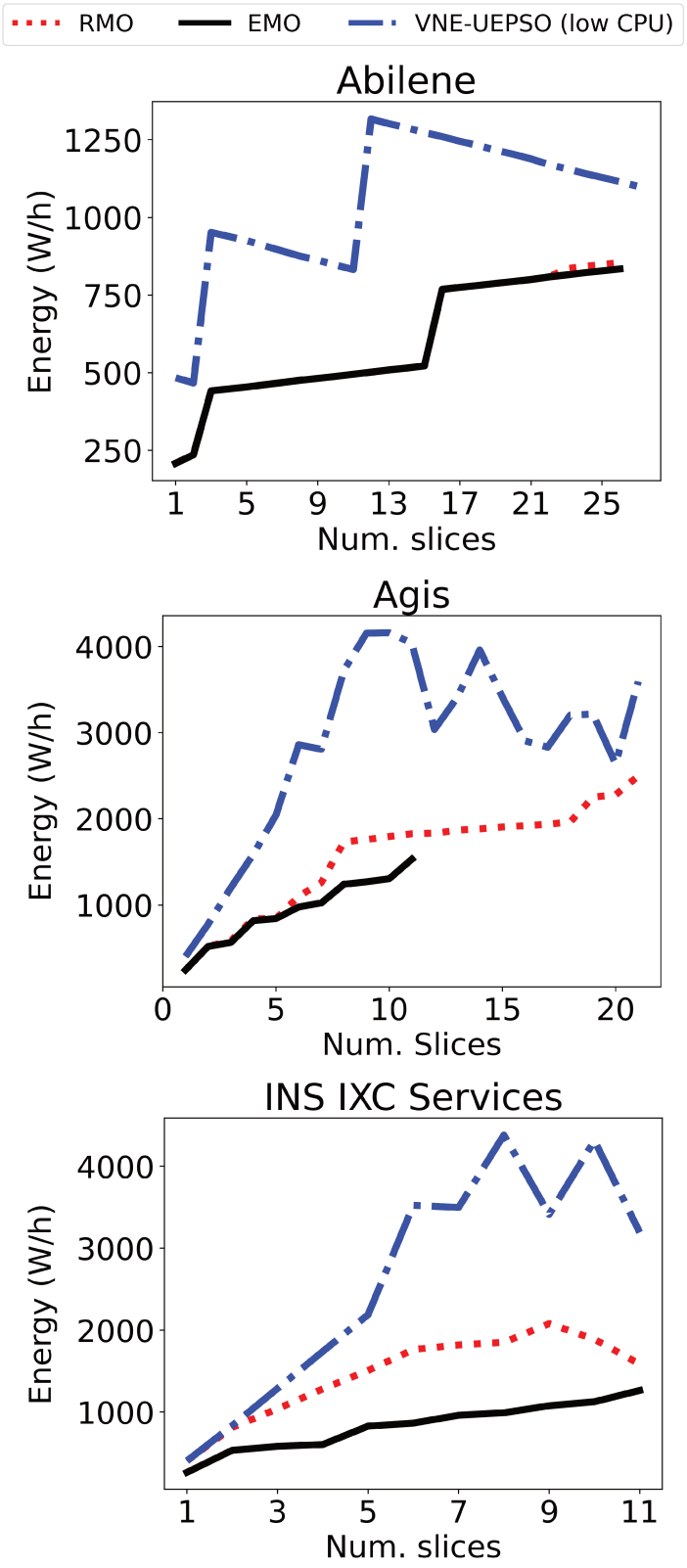


Figure 4.9: Energy consumption

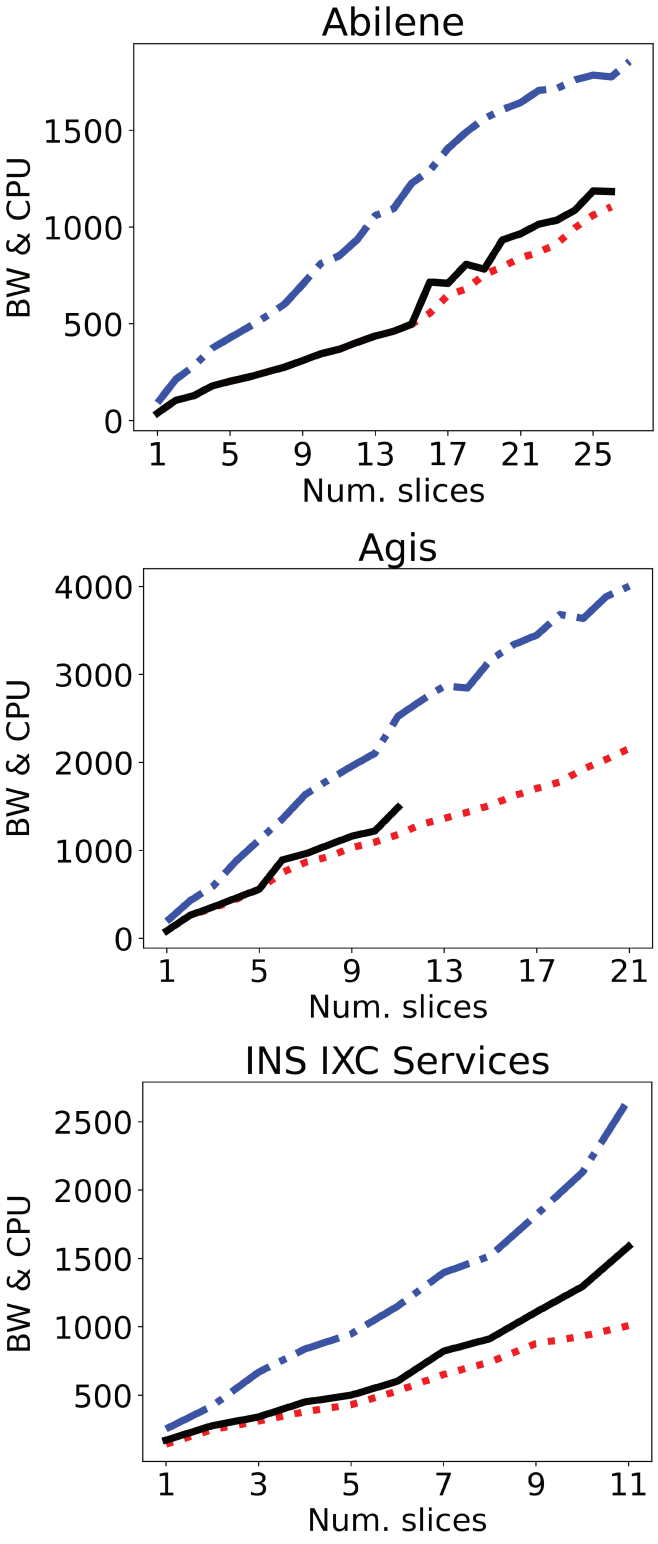


Figure 4.10: Resources consumption

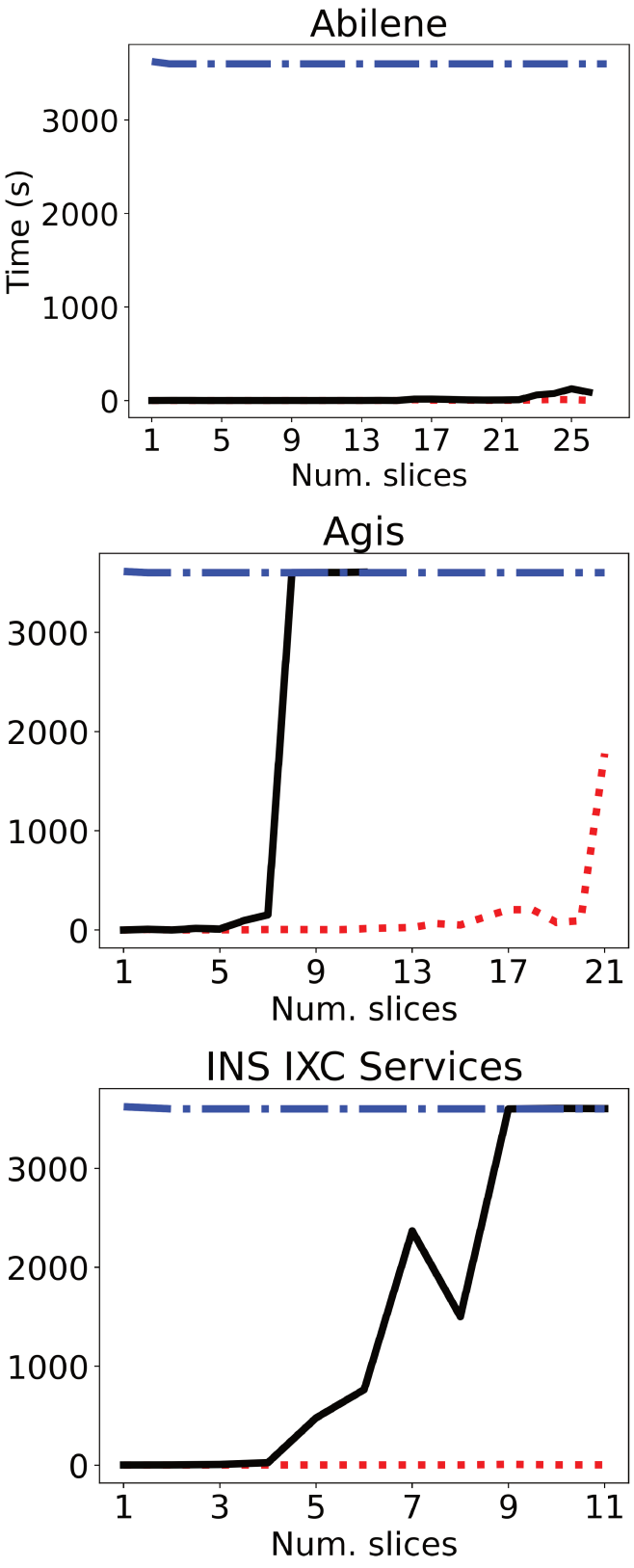


Figure 4.11: Runtime

4.4.5 Proofs

In this section, in order to improve readability, we fully detail the proofs of the theorems underlying our algorithms. The first theorem that we shall prove here is Theorem 1. Let us restate it here:

Theorem 1. *If the requests from a flow k have general independent inter-arrival times, given a path $\mathcal{P} \in \boldsymbol{\mathcal{P}}_k$, the optimal service rates (4.40) can be computed in polynomial time.*

In order to prove it, let us start from the following Lemma:

Lemma 1. *If $f : \mathbb{R} \to (\mathbb{R} \bigcup +\infty)$ is convex and nondecreasing and $g : \mathbb{R}^n \to (\mathbb{R} \bigcup +\infty)$ is convex, then $h = f \circ g$ is convex*

Proof. See chapter 7 of [127] □

Which we use to prove the following proposition:

Proposition 6. *Function*

$$f(\eta) = exp\left[\sum\left((c_i^k(\mu_k(\eta-1)))a_i^k + \frac{1}{2}b_i^{k^2}c_i^{k^2}(\mu_k(\eta-1))^2\right)\right] - \eta$$

has a single root r in $[0,1[$ and it can be found by the bisection method [128].

Proof. For the uniqueness of the root, we refer to [129]. What remains to do to make sure we can use the bisection method is to show that $f(\eta)$ is of opposite signs on $[0, r]$ and $[r, 1]$, For finding r, since this method converges to a root provided we give it an interval $[a, b]$ that contains r such that $f(a)$ and $f(b)$ are of opposite sign.

We first need to prove the convexity of $f(\eta)$.

In our case, the exponential function is clearly convex and non-decreasing and the term inside the exponential is a degree-2 polynomial of η which is a convex expression. Hence by Lemma 1, $f(\eta)$ is convex. Second, recall the geometrical definition of a convex function:f : $X \to \mathbb{R}$ is convex iff for any two points A, B of its function graph, the line segment $[A, B]$ lies above the graph of f.

Also observe that $f(1) = 0$, and that by definition of r we have $f(r) = 0$. Hence, in the $x - y$ plane, the line segment between the point $(r, 0)$ and the point $(1, 0)$ (*e.g.* the portion of the x-axis between r and 1) is above the graph of f, meaning f is always negative on $[r, 1]$. For the sign of f on $[0, r]$, recall the exponential function is always positive, hence $f(0) > 0$. This, combined with the uniqueness of the root r on $[0, 1[$ proves f must be positive on $[0, r]$. □

We are now ready to prove theorem 1, let us restate it here:

Theorem 1. *If the requests from a flow k have general independent inter-arrival times, given a path $\mathcal{P} \in \boldsymbol{\mathcal{P}}_k$, the optimal service rates (4.40) can be computed in polynomial time.*

Proof. As a case study, and without loss of generality, we focus on the case where the inter-arrival time follows a mixture of Gaussian processes. We choose this class of distributions due to its flexibility, but the formulas for the G/M/1 queue that we shall use could be used with any other arrival distribution as long as the arrivals are independent.

First note that from **Assumption 2** we have

$$cpu_k(\mathcal{P}) = \mu_k(\mathcal{P}) \cdot \mathbb{E}[OPS_k] \tag{4.43}$$

Hence, this section will focus on the computation of the minimal suitable value of μ_k as $\frac{1}{\mathbb{E}[OPS_k]}$ is constant. Our approach for solving the MINLP will then be to compute the value of $cpu_k(\mathcal{P})$ in advance for each path, as it will enable us to simplify the problem by removing constraints (4.33) and (4.36).

Observe that an acceptable value of μ_k (*e.g.* such that constraint 4.36 is respected) should induce a waiting time such that $\mathcal{P}(D_{wait}(\mu_k, AT_k) \leq D^k_{total} - 2 \cdot D_{prop}(\mathcal{P})) \geq SL_k$. Furthermore, since the arrival distribution is an input to our algorithm, this probability is an increasing function of μ_k, *e.g.*, the more CPU is allocated to service k, the less time it takes to treat a single request. This implies that for finding the minimum suitable value of μ_k, we have to find μ_k such that

$$\mathbb{P}(D_{wait}(\mu_k, AT_k) \leq D^k_{total} - 2 \cdot D_{prop}(\mathcal{P})) = SL_k. \tag{4.44}$$

We now detail the way this minimum value of μ_k can be computed. First, we note that since there is no queuing delay on links (**Assumption 3**), the service-time is exponentially distributed (**Assumption 2**), and arrivals are independent, the service of one commodity can be modeled as a G/M/1 queue. The service rate μ_k of this queue depends on the amount of CPU reserved for the service.

As shown in [129] [130] the distribution of Equation (4.44) can be computed by first finding the unique root in $[0, 1[$ of

$$\eta = M_A(\mu(\eta - 1)). \tag{4.45}$$

In our case, the arrival distribution AT_k for service k is a mixture of Gaussians

$$AT_k = \sum_i AT^i_k c^k_i \tag{4.46}$$

Such that $\sum_i c_i^k = 1,\ c_i^k \geq 0\ \forall i$ and $AT_k^i \sim \mathcal{N}(a_i^k,\ {b_i^k}^2)$

Since AT_k is a linear combination of independent random variables, its moment-generating function M_{AT_k} can be computed as:

$$M_{AT_k}(t) = \prod_i M_{AT_k^i}(c_i^k \cdot t)$$

Furthermore, recall that the the moment generating function of $AT_k^i \sim \mathcal{N}(a_i^k,\ {b_i^k}^2)$ is

$$M_{A_i^k}(t) = e^{ta_i^k + \frac{1}{2}{b_i^k}^2 t^2}$$

We then deduce that we have

$$M_{AT_k}(t) = exp\left[\sum_i \left(c_i^k \cdot t \cdot a_i^k + \frac{1}{2}{b_i^k}^2 {c_i^k}^2 t^2\right)\right].$$

Hence, for a fixed value of μ_k, we have to find the root r of

$$\eta = exp\left[\sum \left((c_i^k(\mu_k(\eta - 1)))a_i^k + \frac{1}{2}{b_i^k}^2 {c_i^k}^2 (\mu_k(\eta - 1))^2\right)\right] \tag{4.47}$$

which is equivalent to finding the root of the function

$$f(\eta) = exp\left[\sum \left((c_i^k(\mu_k(\eta - 1)))a_i^k + \frac{1}{2}{b_i^k}^2 {c_i^k}^2 (\mu_k(\eta - 1))^2\right)\right] - \eta \tag{4.48}$$

Hence, due to Propositon 6 the solution we propose for finding r is to use the bisection method with an initial interval $[0, 1 - \epsilon]$ with a small enough ϵ. In theory it could happen that r lies in $[1 - \epsilon, 1]$. If this happens, we can simply reduce epsilon and start over again.
Armed with this method for finding r, we propose to find μ_k similarly, using the binary search algorithm described in Algorithm 10.

Algorithm 10 Binary search for finding minimum value of μ_k

Input: Considered path propagation delay $D_{prop}(P)$, maximum delay D_k, maximum CPU of t_k, CPU_{t_k}
Output: Minimum value of μ_k

$lb \leftarrow 0$
$ub \leftarrow \frac{CPU_v}{OPS_k}$
while $True$ **do**
    $\mu \leftarrow (lb + ub)/2$
    if $\lambda/\mu < 1$ **then**
        Calculate μ using equations 4.47 and 4.48;
        Calculate $\mathbb{P}(W \leq D_k(t_k))$;
        if $\mathbb{P}(W \leq D_k(t_k))$ is close to SL_k **then**
            break
        else
            if $\mathbb{P}(W \leq D_k(t_k)) \geq SL_k$ **then**
                ub $= \mu$
            end if
            lb $= \mu$
        end if
    end if
end while

The idea is to maintain a lower and an upper bound on μ_k, lb and ub. At each iteration we calculate $\mathbb{P}(W_k \leq SL_k | \mu_k = (lb + ub)/2)$.

Then, if this probability is too high it means we are using too much resources and we decrease the upper bound and if it is too low we increase the lower bound as it means we would lack resources. We do this until we find the minimum acceptable value of μ_k, such that the probability of having an acceptable waiting time equals SL_k. □

Let us now restate the second main theorem we want to prove:

Theorem 2. *The pricing problem is equivalent to a bi-objective shortest path problem (BOSPP) with forbidden paths, which can be solved efficiently (e.g. [125]).*

In order to prove it, we start by proving the following restricted version:

Lemma 2. *The pricing problem, when the Constraints (4.42) are not considered, is equivalent to a bi-objective shortest path problem (BOSPP).*

Proof. Let us first report how to compute the reduced cost $\bar{c}_k(\mathcal{P})$ of path variable $f_k(\mathcal{P})$. (again, when (4.42) are not considered). Let w_{v_i,v_j}, σ_k, z_{v_i}, h_{v_j}, and q_{v_i} be the dual variables associated with constraints (4.31), (4.32), (4.33), (4.34), (4.35), respectively. Let $\mathbf{y}$ be the concatenation of these dual variables into a single vector. At each iteration l of the column

generation procedure, we compute the optimal solution $\mathbf{x}_*^{(l)}$ of RMP(Ω_n). Then we compute the corresponding dual variables $w_{v_i,v_j}^{(l)}$, $\sigma_k^{(l)}$, $z_{v_i}^{(l)}$, $h_{vi}^{(l)}$, $q_{v_i}^{(l)}$ for each path $\mathcal{P}$ with source node v_i and destination node v_j (and their concatenation $\mathbf{y}_*^{(l)}$) from $\bar{\mathbf{x}}_*^{(l)}$.Recall that the vector of reduced costs is $\bar{\mathbf{c}}^{(l)} = \mathbf{c} - \mathbf{A}^T \cdot \mathbf{y}_*^{(l)}$. The reduced cost associated with variable $f_k(\mathcal{P})$ is hence

$$\begin{aligned}
\bar{c}^{(l)}(f_k(\mathcal{P})) &= \mathbf{e}_k^T(\mathcal{P}) \cdot \bar{\mathbf{c}}^{(l)} && (4.49)\\
&= \mathbf{e}_k^T(\mathcal{P}) \cdot \left(\mathbf{c} - \mathbf{A}^T \cdot \mathbf{y}_*^{(l)}\right) && (4.50)\\
&= \mathbf{e}_k^T(\mathcal{P}) \cdot \mathbf{c} - \mathbf{e}_k^T(\mathcal{P}) \cdot \left(\mathbf{A}^T \cdot \mathbf{y}_*^{(l)}\right) && (4.51)\\
&= 0 + \sum_{\forall (v_i,v_j) \in \mathcal{P}} w_{v_i,v_j}^{(l)} \cdot BW_k^d - \sigma_k^{(l)} + && (4.52)\\
& cpu_k(\mathcal{P}) \cdot z_{end(\mathcal{P})}^{(l)} + q_{end(\mathcal{P})}^{(l)} \cdot l(k, v_i) + \frac{1}{K} \sum_{v_i \in \mathcal{P}} h_{v_i}^{(l)}. && (4.53)
\end{aligned}$$

In the formula above we make use of a slight abuse of notation by interchangeably considering a path as a set of arcs in the first sum and as a set of nodes in the last one. To find $f_k(\mathcal{P}_{(l)}^*)$, we first find the "best path" for any commodity k, i.e. the path $\mathcal{P} \in \boldsymbol{\mathcal{P}}_k$ that minimizes the reduced cost. Then, we compare the best paths of all the commodities and we choose the one with the smallest reduced cost. In other words, the first step is to solve:

$$\arg \min_{\mathcal{P} \in \boldsymbol{\mathcal{P}}_k} \bar{c}^{(l)}(f_k(\mathcal{P})) \text{ for any given commodity } k \in \mathcal{K}. \quad (4.54)$$

We remark the sum of equation (4.52) can be split into three distinct terms:

- Term 1 is $\sum_{\forall (v_i,v_j) \in \mathcal{P}} w_{v_i,v_j}^{(l)} \cdot BW_k^d + \frac{1}{K} \sum_{v \in \mathcal{P}} h_v^{(l)}$. To minimize it, we can simply compute the shortest path on a graph having costs on arcs from v_i to v_j set as $w_{v_i,v_j}^{(l)} \cdot BW_k^d + \frac{1}{K} \cdot h_{v_i}^{(l)}$.

- Term 2 is $cpu_k(\mathcal{P}) \cdot z_{end(\mathcal{P})}^{(l)}$ and depends on cpu_k. Similarly, if we had only this term, we would be able to find the path which minimizes it by using a shortest path algorithm. This is because we know that, although it is non-linear, the amount $cpu_k(\mathcal{P})$ of CPU consumed by the software component of commodity k grows monotonically with the propagation delay of the path chosen to serve that commodity. Hence, choosing the shortest path in terms of propagation delay also leads to minimum $cpu_k(\mathcal{P})$.

- Term 3 is $\sigma_k^{(l)} + q_{end}^{(l)} \cdot l(k, v_i)$. Differently from the previous two terms, this term only depends on the a-priory requirements of commodity k and not on the path we decide to use to serve it.

Let us now prove the following lemma:

Lemma 3. *The path which minimizes the reduced cost is non-dominated with respect to the bi-objective shortest path problem (BOSPP) where Term 1 and Term 2 are the objectives*

Proof. Assume the path which minimizes the reduced cost is dominated, then, since the reduced cost is monotonically increasing in both objectives, it means there is another path which is better than the best path. This is a contradition. ■ **End of subproof of Lemma 3**

Therefore, for a given commodity k, problem (4.54) can be solved by first solving a bi-objective shortest path problem (BOSPP) and then evaluating the objective function (*e.g.* the reduced cost) of each path and picking the path with minimum reduced cost. This procedure is summarized in algorithm 4.7. ■ **End of proof of Lemma 2**

We now turn to extending this lemma to the case which includes constraints (4.42), we will need the following Lemma:

Lemma 4. *Let* $\mathbf{A} \in \mathbb{R}^{n \times n}$. *We note its elements as* $a_{x,y}$. *Let*

$$\mathbf{A}' = \left| \begin{array}{cccccccc} & & & & & & & 0 \\ & & & \mathbf{A} & & & & \vdots \\ & & & & & & & 0 \\ 0 & \cdots & 0 & -1 & 0 & \cdots & 0 & 1 \end{array} \right|$$

with elements $a'_{x,y}$, *and let* $g \in [0, n]$ *be the index of the unique column such that* $a'_{n+1,g} = -1$. *The inverse of* $\mathbf{A}'$ *is the matrix*

$$\mathbf{B} = \left| \begin{array}{cccc} & & & 0 \\ & \mathbf{A}^{-1} & & \vdots \\ & & & 0 \\ a_{g,1} & \cdots & a_{g,n} & 1 \end{array} \right|$$

. *We note* $b_{x,y}$ *the elements of* B.

Proof. Let $i_{x,y}$ be the elements of the identity matrix of rank $n + 1$. For the first n lines of $\mathbf{B}$ the elements of the product $\mathbf{A}'\mathbf{B}$ for all $x \leq n$ and

$y \leq n+1$ are:

$$p_{x,y} = \sum_{k=1}^{n+1} a'_{x,k} b_{k,y} \tag{4.55}$$

$$= a'_{x,n+1} b_{n+1,k} + \sum_{k=1}^{n} a'_{i,k} b_{k,j} \tag{4.56}$$

$$= 0 + \sum_{k=1}^{n} a_{i,k} b_{k,j} \tag{4.57}$$

$$= i_{x,y} \tag{4.58}$$

For the last line, and $\forall\ j \leq n$, we have:

$$p_{n+1,j} = \sum_{k=1}^{n+1} a'_{n+1,k} b_{k,j} \tag{4.59}$$

$$= -1 \times b_{g,j} + 1 \times b_{n+1,j} \tag{4.60}$$

$$= 0 = i_{n+1,j} \tag{4.61}$$

and for $j = n+1$:

$$p_{n+1,n+1} = \sum_{k=1}^{n+1} a_{n+1,k} b_{k,n+1} \tag{4.62}$$

$$= -1 \times 0 + 1 \times 1 = 1 = i_{n+1,n+1} \tag{4.63}$$

□

We are now ready to prove the following proposition:

Proposition 7. *The dual variable $u^{(l)}_{f_k(\mathcal{P})}$ associated with the constraint $a(v_i) \geq f_k(\mathcal{P}) \cdot \omega_v(\mathcal{P})$ in (4.42) is always equal to zero for any variable $f_k(\mathcal{P})$ (tied to a path $\mathcal{P}$ that can potentially serve a commodity k) not yet considered in RMP(Ω_l). Furthermore, including a new variable (and hence an additional constraint (4.42)) does not alter the value of other dual variables when using the optimal solution of RMP(Ω_l) as the starting point of the optimization of RMP(Ω_{l+1}).*

Proof. Recall that reduced costs $\bar{\mathbf{c}}$ are calculated from dual variables $\mathbf{y}$, which are calculated using the inverse of the basic matrix $\mathbf{B}^{-1}$ (see equation (4.16)). For this reason, our proof shall focus on the construction of this basic matrix and on the analysis of its inverse at each step of the construction. Recall that $\mathbf{B}$ is a square matrix, and that its number of rows is equal to the number of constraints in the problem.

We shall construct the initial square basic matrix of RMP(Ω_{l+1}), $\mathbf{B}(l+1)_{init}$ by iteratively adding one row and one column to the basic matrix $\mathbf{B}(l)_*$ of the optimal solution of RMP(Ω_l) for each of the constraints $c_1, \cdots c_t, \cdots c_{|\mathcal{V}|}$ that we have to take into account when considering a new variable. Let us note $\mathbf{Y}_t^{(l+1)}$ the t^{th} intermediate basic matrix computed during the construction of $\mathbf{B}(l+1)_{init}$. $\mathbf{Y}_t^{(l+1)}$ is a matrix where the columns and rows corresponding to the inclusion of the t first constraints have been added. At the beginning, when we have not performed any addition, we have $\mathbf{Y}_0^{(l+1)} = \mathbf{B}(l)_*$. We set the initial basic matrix $\mathbf{B}(l+1)_{\text{init}} = \mathbf{Y}_{|\mathcal{V}|}^{(l+1)}$.

Observe that if we were to solve the full problem (*e.g.* with all paths considered) at once, at the initialization of the simplex algorithm (see Section 4.2.2.2), all the initial basic variables would be slack variables (this is true for any LP). Hence, since constraints c_t were not considered in RMP(Ω_l), we know when including them that the corresponding slack variable s_t must be basic in RMP(Ω_{l+1}). Furthermore we have two cases: either node variable $a(v_t)$ is non-basic, in which case, since the only basic variable appearing in c_t is s_t, the basic coefficient matrix extended with c_t is:

$$\mathbf{Y}_t^{(l+1)} = \begin{vmatrix} & & & 0 \\ & \mathbf{Y}_{t-1}^{(l+1)} & & \vdots \\ & & & 0 \\ 0 & \cdots & 0 & 1 \end{vmatrix}$$

In this matrix, the value 1 is the value of s_t, and the zeros mean that no other basic variable appears in c_t. This matrix has inverse:

$$(\mathbf{Y}_t^{(l+1)})^{-1} = \begin{vmatrix} & & & 0 \\ & (\mathbf{Y}_{t-1}^{(l+1)})^{-1} & & \vdots \\ & & & 0 \\ 0 & \cdots & 0 & 1 \end{vmatrix}$$

Otherwise, if $a(v_t)$ (the only other variable appearing in c_t besides s_t) is basic, we assume variable $a(v_t)$ has index g in the basis, and the new basic coefficient matrix is

$$\mathbf{Y}_t^{(l+1)} = \begin{vmatrix} & & & & & & & 0 \\ & & & \mathbf{Y}_{t-1}^{(l+1)} & & & & \vdots \\ & & & & & & & 0 \\ 0 & \cdots & 0 & -1 & 0 & \cdots & 0 & 1 \end{vmatrix}$$

Were the g^{th} element of the new row is equal to -1, *e.g.* the coefficient of $a(v_t)$ in the normalized version of the constraint. Note we still have the

value 1 since s_t is basic regardless of the fact $a(v_t)$ is basic. Assuming the g^{th} row of $\mathbf{Y}_t$ is noted $\zeta_{g,1}, \zeta_{g,2}, ..., \zeta_{g,m+j}$, by lemma 2, the inverse of $\mathbf{Y}_t$ is then:

$$(\mathbf{Y}_t^{(i+1)})^{-1} = \begin{vmatrix} & & & & 0 \\ & (\mathbf{Y}_{t-1}^{(l+1)})^{-1} & & & \vdots \\ & & & & 0 \\ \zeta_{g,1} & \zeta_{g,2} & \cdots & \zeta_{g,m+j} & 1 \end{vmatrix}$$

Hence, the basic matrix from which the optimization of RMP(Ω_{l+1}) starts, $\mathbf{B}(l+1)_{init}$ could be constructed by iteratively adding new rows and columns to $\mathbf{B}(l)_*$, regardless of the path added. Furthermore, since the vector of values of dual variables in RMP(Ω_{l+1}) is calculated as $\mathbf{y} = \mathbf{c}_B^T \cdot \mathbf{B}(l+1)_{init}{}^{-1}$, we deduce that we have $u_{f_k(\mathcal{P}'}^{(l)} = 0$ for all of the potential new constraints associated with potential entering variables $\mathcal{P}'$. Finally, since the objective coefficient of slack variables is 0, we know that the non-zero new elements of the matrix will all be multiplied by zero upon calculating the reduced cost. Hence, the dual variables of other constraints will remain the same as in the optimal RMP. □

We can now prove Theorem 2:

Theorem 2. *The pricing problem is equivalent to a bi-objective shortest path problem (BOSPP) with forbidden paths, which can be solved efficiently (e.g. [125]).*

Proof. From Proposition 7, we know that if the only paths considered in the pricing problem were the excluded ones, then solving the associated BOSPP would be enough to find the next variable to include. However, if we use a BOSPP algorithm, we will also consider the path variables that are already included in the problem, for which we have no indication of the value of $u_{f_k(\mathcal{P}'}^{(l)}$[6]. Hence, to make sure we always solve the pricing problem correctly, we have to forbid these paths during the resolution of the pricing problem. This means the pricing problem can be reduced to a BOSPP where all already considered paths are forbidden. This problem can be solved using the approach which we presented in Section 4.3. □

[6]indeed, during our experimentations, we observed examples where $u_{f_k(\mathcal{P}'}^{(l)}$ could be either positive or negative for the path variables that are already considered in the problem

4.5 Contribution 3: Understanding what makes energy minimization harder

The numerical results section we have just presented underline the fact that, in practice, the problem is much longer to solve when the objective considered is the energy consumption over the resource consumption. For this reason, in this section, we investigate the hardness of a simplified version of the problem from the theoretical point of view. Namely, we ignore the CPU resources and the delay constraints and focus only on the multicommodity flow problem where each commodity has exactly one possible destination. More specifically, we observe that if splitting paths is authorized (*e.g.* if variables $f_k(\mathcal{P})$ are continuous), the multicommodity flow problem with resource minimization becomes polynomially solvable with a Linear Programming solver. However, in the case of the energy minimization, the on/off variables $a(v_i)$ remain binary. The question hence is whether this problem is NP-hard or not. We answer this question positively, even in the case where there is exactly one commodity to route.

4.5.1 Energy Minimization Flow Problem (EMF)

For the sake of completeness, let us redefine the simplified problem here. We consider a directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with capacities BW_{v_i,v_j} on $(v_i, v_j) \in \mathcal{E}$ and a set of commodities $\mathcal{K}$. Each commodity k_i is made a triplet (s_i, t_i, d_i) where s_i is its source, t_i its destination, and d_i is the amount of flow to route from s_i to t_i. Furthermore, we are given an objective function to minimize, which is equal to $\sum_{v_i \in \mathcal{V}} e(v_i) \times a(v_i)$, where $a(v_i)$ is a binary variable with value 1 if the node v_i is used in the flow and 0 otherwise, and $e(v_i)$ is the cost of using node i. We call this problem the Energy Minimization Flow problem (EMF). This problem can either be defined with splittable flows (*e.g.* a single commodity can be routed on one or more paths), or with unsplittable flows, in which case exactly one path has to be used for each.

4.5.2 Proof of NP-Hardness

We focus on the splittable case, because the proof for the unsplittable version of EMF directly follows from the well-known fact that the problem of finding a feasible unsplittable multicommodity flow without any objective is already NP-Hard [131]. Furthermore, we focus on an even simpler case, as we shall prove that EMF is NP-Complete even if $|\mathcal{K}| = 1$ and all the costs $e(v_i)$ are equal to 1. We shall do so by a reduction from the Deterministic Network Interdiction problem. Let us now introduce the decision version of both problems:

Problem: Deterministic Network Interdiction (DNI): We are given a Directed Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with capacities BW_{v_i,v_j} on $(v_i, v_j) \in \mathcal{E}$ and distinguished nodes s and t, an integer positive number U and an integer postive number R.

Question: Does there exist a set of arcs $\mathcal{E}' \subset \mathcal{E}$ with $|E'| \leq R$ such that there exists a flow between s and t in $\mathcal{G}' = (\mathcal{V}, \mathcal{E} - \mathcal{E}')$ with value at least U ?[7]

Problem: EMF: We are given a Directed Graph $\mathcal{G}_2 = (\mathcal{V}_2, E_2)$ with capacities BW_{v_i,v_j} on $(v_i, v_j) \in \mathcal{E}_2$ and distinguished nodes s_2 and t_2, an integer positive number U_2 and an integer postive number R_2.

Question: Does there exist a set of nodes $\mathcal{V}_2' \subset \mathcal{V}_2$ with $|\mathcal{V}_2'| \leq R_2$ such that there exists a flow between s and t in $(\mathcal{V} - \mathcal{V}_2', \mathcal{E})$ with value of at least U_2 ?

We shall now prove that **EMF** is NP-Complete. To do so, we reduce the **DNI** problem into **EMF**, using the technics described in section 4.2.4.

Consider a **DNI** problem where s has exactly one outer neighbor and 0 inner neighbor, and t has exactly one inner neighbor and 0 outer neighbor. We also consider that the unique arcs between s or t and their unique respective neighbor has capacity of ∞. We note it is easy to transform any **DNI** instance to satisfy such requirement. We create a graph $\mathcal{G}_2 = (\mathcal{V}_2, \mathcal{E}_2)$ in the following way: $\mathcal{G}_2$ is the line digraph of $\mathcal{G}$, *e.g.* we create one vertex $i\bar{j}$ per edge $(v_i, v_j) \in \mathcal{E}$. We then connect $u\bar{v}, w\bar{x} \in \mathcal{V}_2$ iff $v = w$. We set the capacity of all the edges of $\mathcal{E}_2$ as ∞. Then, for each node $i\bar{j}$ of $\mathcal{N}_2$, we add a node $i\bar{j}'$ and add an edge $(i\bar{j}, i\bar{j}')$ to $\mathcal{E}_2$ with capacity BW_{v_i,v_j}. All the outer edges $(i\bar{j}, x\bar{y})$ of $i\bar{j}$ are removed from $\mathcal{E}_2$ and replaced with $(i\bar{j}', x\bar{y})$. Finally, we set $s_2 = s\bar{x}$, $t_2 = \bar{y}t'$ where x and y respectively are the unique neighbors of s and t in $\mathcal{G}$. We also set $U = U_2$ and $2R = R_2$. We illustrate the construction of such a graph in Figure 4.12

Lemma 5. *Any flow in $\mathcal{G}$ has an equivalent flow in $\mathcal{G}_2$ with the same value*

Proof. We consider a flow of $\mathcal{G}$ as a collection of paths. Consider any path $s, x_1, x_2, ..., x_n, t$ in $\mathcal{G}$. Due to the construction of $\mathcal{G}_2$, there is always a corresponding path of $\mathcal{G}_2$, which can be written as $s\bar{x}_1, s\bar{x}_1', x_1\bar{x}_2, x_1\bar{x}_2', ..., x_n\bar{t}, x_n\bar{t}'$. Furthermore, the capacity of any arc $(x_i, x_{i+1}) \in \mathcal{E}$ is equal to that of $(x_i\bar{x}_{i+1}, x_i\bar{x}_{i+1}') \in \mathcal{E}_2$, and the other arcs

[7] We use the negated version of the original question here, which is "Does there exist a set of arcs $\mathcal{E}' \subset \mathcal{E}$ with $|E'| \leq R$ such that the maximum flow between s and t in $\mathcal{G}'$ has value of at most U ?". Note this is equivalent as one can construct a trivial reduction where the answer to one problem is yes when the other is no.

in the path of $\mathcal{G}_2$ have a capacity of ∞. Hence the capacity of the path is the same, which also proves any flow of one of the graphs has an equivalent flow with the same capacity in the other one. □

Lemma 6. *Let us call $\mathcal{P}(i,j)$ the set of paths of $\mathcal{G}$ which use arc (v_i, v_j) and $\mathcal{P}_2(i,j)$ the set of paths equivalent to the paths of $\mathcal{P}(i,j)$ in $\mathcal{G}_2$. Removing (v_i, v_j) from $\mathcal{E}$ is equivalent to removing one specific node from $\mathcal{V}_2$, in the sense that both transformations respectively prevent the use of $\mathcal{P}(i,j)$ and $\mathcal{P}_2(i,j)$ while not altering other paths.*

Proof. By construction of any path of $\mathcal{P}_2(i,j)$, $i\bar{j}$ is part of all the paths of $\mathcal{P}_2(i,j)$ and it is also part of no other path of $\mathcal{G}_2$. Hence the transformation consists in removing $i\bar{j}$ from $\mathcal{V}_2$. □

Lemma 7. *If there exists a solution to **EMF** in $\mathcal{G}_2$ with value U_2 and forbidden nodes V_2' such that V_2' cannot be expressed as a set of couples $\{\{x_i\bar{x}_j, x_i\bar{x}_j{}'\}...\{x_{\bar{k}}x_l, x_{\bar{k}}x_l{}'\}\}$, then there exists a better (e.g. which allows more flow to pass) or equal solution with such a form and with the same number of forbidden nodes.*

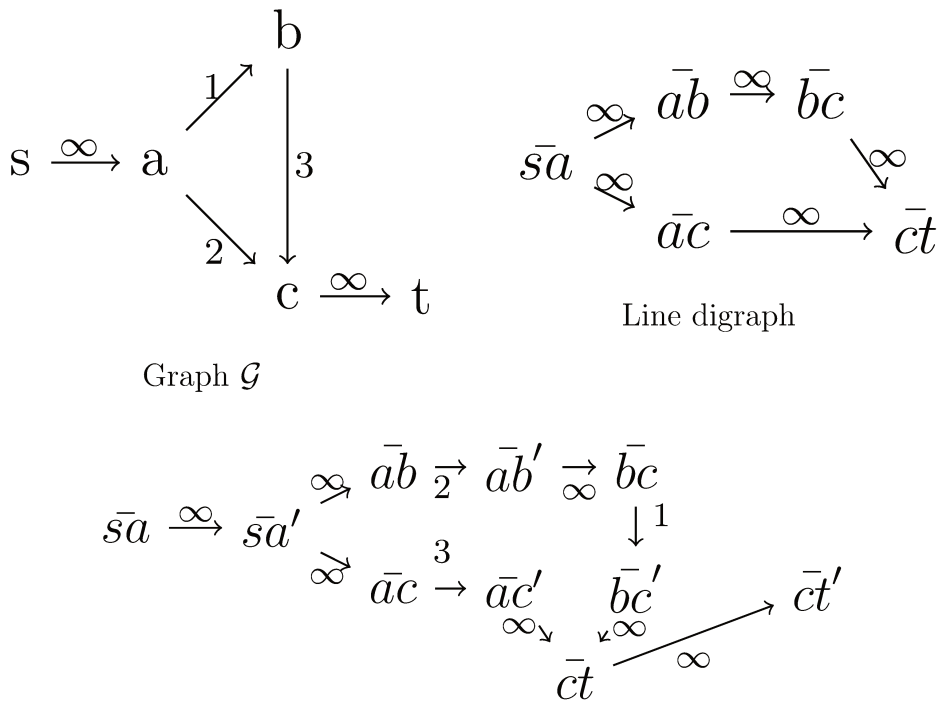
Proof. It suffices to observe that removing $x_i\bar{x}_j$ also removes its only outer arc which is $(x_i\bar{x}_j, x_i\bar{x}_j{}')$, hence the better solution is one where the same number of nodes are forbidden, but less arcs are. □

Theorem 3. *EMF is NP-Complete in the strong sense, e.g. it has no polynomial or pseudo-polynomial time solution unless $P = NP$.*

Proof. By Lemmas 5 and 6, given a solution to **DNI** in $\mathcal{G}$, where the R arcs (i,j) are forbidden, one can construct a solution to the **EMF** instance in $\mathcal{G}_2$ by removing the $2R$ nodes $i\bar{j}$ and $i\bar{j}'$.

For the other way of the equivalence proof, given a solution to **EMF** in $\mathcal{G}_2$, one can use Lemma 7 to improve it into a solution with $2R$ forbidden nodes V_2' $\{i\bar{j}, ...k\bar{l}\} \bigcup \{i\bar{j}', ...k\bar{l}'\}$. Then, this solution can be turned into a solution to **DMI** by setting $V' = \{(v_i, v_j)\}$. Finally, **EMF** clearly is in NP (*e.g.* given a solution, it is feasible in polynomial time to verify whether the solution answers the question by yes or no), hence it is NP-complete in the strong sense because **DNI** is NP-complete in the strong sense. □

We argue that this hardness proof partly explains the longer solution times we found in the previous section: the **EMF** problem is inherently harder than the Resource Minimization one.



Graph $\mathcal{G}$

Line digraph

Graph $\mathcal{G}_2$

Figure 4.12: Construction of graph $\mathcal{G}_2$

4.6 Conclusion

In this chapter, we have derived a novel model to describe the placement of edge services problem under tight performance requirements. Our model accounts for processing and propagation delays across the network and adds flexibility to infrastructure owners by giving them the possibility to scale the CPU assigned to each service in order to meet strict SLAs. We designed an exact solution which minimizes the consumed energy while ensuring that all services and their performance requirements are satisfied. Both aspects that enable efficient network slicing for 5G and beyond. Our results show that accurately controlling CPU usage across nodes of the network and allocating services optimally enable substantial energy savings, up to 50%, compared to that of a classic resource-consumption minimization approach. Furthermore, we have proved that the energy minimization problem is inherently harder than the classic resource minimization approach, as even a simplified version of the problem with a flow is NP-hard. Hence, in our future work we will improve our solution in terms of speed, based on these theoretical knowledge, and we will develop heuristics and algorithms with approximation guarantees. Moreover, we will consider the case where the

distribution of requests and loads is not known in advance and must be learned via observations. The main limitation of our method in its current form is that we model each Edge Slice as a set of flows, and that each flow is served by a given instance of the service and its collocated UPF. This means that potentially, we could be wasting resources since it could be more efficient to consider that two instances of the same service (which serve different flows of the same slice) could be modeled by a single queue, potentially saving some resources. On the other hand this ensures we are able to quickly move an instance of a service if for example, the number of requests arrising from a given flow has to suddenly increase, which might be more challenging if several flows are served by a single queue. We leave the question whether this is worth doing, and of the amount of saved energy and resources achievable by considering this aspect as a future work. Another limiting aspect of this work is that only CPU was considered when modeling energy. It would be of interest to model it in a finer way by including other factors such as memory. Indeed, some sources show that models of the energy consumption that take only the CPU into account can be refined by also incorporating those other factors. [123]. This would be an important aspect to consider, which would also require to first conduct analysis of the energy consumption on a large scale, similar to the ones provided in [122].

Chapter 5

Radio Access Network: Integrated Access and Backhaul Routing

5.1 Introduction

Let us now turn to the optimization of the Radio Access Network (RAN). Compared to the previous chapter, the work presented here shall model the traffic of each individual users and abstract services and network functions away. Indeed, here the goal with be to guarantee the throughput, assuming the rest of the network is managed by techniques such as the ones presented in Chapters 1 and 2. In particular, the problem considered here is the optimization of energy in an ultra-dense urban scenario.

Ultra-dense deployment and millimeter wave (mmWave) have been portrayed as the solution to meet the stringent requirements standardized with 5G in terms of data rates [132]. As shown by many studies, mmWave is capable of providing multi-gigabit connectivity to User Equipments (UEs) [133, 134], and Integrated Access and Backhaul (IAB) has been shown to be an effective way to reduce the deployment costs [135]. This technology, introduced in 3GPP Release 16, allows, to connect only a subset of the Next Generation Node Bases (gNBs), called IAB-donors, to the fiber backhaul while the rest of the gNBs, called IAB-node, rely on in-band wireless communication to reach one of the donors, forming a multihop wireless network.

By dynamically activating and deactivating gNBs with respect to the current load of the network, it is possible to reduce the energy footprint of the system, switching off IAB-nodes that are not strictly necessary to match the requested level of service. This is of critical importance since energy consumption accounts for up to 60% of the Operational Expenditure (OpEx) [136]. Research on energy optimization techniques for traditional

networks usually assumes that all gNBs are connected to the fiber backhaul. The presence of wireless-only IAB-nodes, however, increases the complexity of the scenario. In fact, the deactivation of an IAB-node might disrupt the service of another IAB-node that is relying on it. Moreover, IAB-nodes need to support periodic wake-up to update their radio state in order to dynamically change the topology when needed.

Thanks to the effort led by the Open Radio Access Network (O-RAN) Alliance, which has opened the Radio Access Network (RAN) architecture by introducing interfaces such as the E2 and O1 and the concept of RAN intelligent controller (RAN Intelligent Controller (RAN)), which enables operators to modify the parameters and the resource allocation of the RAN while it is running [137], it is now possible to integrate custom closed-loop control logic in the RAN. This has been already studied for several applications, such as network slicing [138], and, more recently, an extension of the O-RAN architecture has been proposed for IAB [139].

Our optimization approach takes advantage of this closed-loop control framework to overcome the limitations discussed above and dynamically minimize the number of active IAB-nodes, while maintaining a minimum capacity per UE. The optimization—based on input data that can be obtained through O-RAN interfaces—generates a topology tree over which we route the traffic from each UE to the IAB-donor, deactivating IAB-nodes that are not needed and distributing UEs across the available gNBs.

Hence, in this chapter, we fill this gap by formulating the problem as a binary nonlinear program. Similar to the previous chapter, the continuous relaxation is non-convex, hence it is not solvable using off-the-shelf solvers, therefore we show it can be transformed into an equivalent binary linear program, which we then solve using the Gurobi solver. The approach is evaluated on a scenario built upon open data of two months of traffic collected by network operators in the city of Milan, Italy, [140] together with detailed morphological data of the same area. Our optimization model manages to perfectly tune the number of active gNBs with the number of UE, reducing by 47% the total number of hours the gNBs (i.e., IAB-nodes) had been active, while maintaining a minimum downlink capacity for each UE equals to 80Mb/s. This shows how introducing dynamic optimizations enabled by the O-RAN architecture can effectively target improvements in energy efficiency.

5.2 Background on IAB networks

To the best of our knowledge, joint routing and energy optimization on multi-hop IAB topologies has never been studied before. Most studies focus on optimizing the IAB topology with different constraints, such as in

Notation	Description
$\mathcal{G} = (\mathcal{V}, \mathcal{E})$	Measurements graph
$\mathcal{U} \subset \mathcal{V}$	Set of UEs
$t \in V$	IAB donor
BW_{v_i,v_j}	Bandwidth of wireless link (v_i, v_j)
$\mathcal{T}$	Tree to find in $\mathcal{G}$ (output of the algorithm)
$\mathcal{K}$	Set of commodities to route
s_k	Source of commodity k
t_k	Destination of commodity k
BW_k^d	Bandwidth demanded by commodity k
$\mathcal{N}_{out}(v)$	Outer neighbors of v
$\mathcal{N}_{in}(v)$	Inner neighbors of v
$out(v)$	Cardinal of $\mathcal{N}_{out}(v)$
$in(v)$	Cardinal of $\mathcal{N}_{in}(v)$
$deg(v)$	Degree of v
$a(v)$	Binary variable indicating whether v is turned on or off
$f_k(v_i, v_j)$	Binary variable indicating whether commodity k uses edge (v_i, v_j)
$x_m(v_i)$	Binary variable in the linearized model. Indicates if at least m of the inner edges incident to v_i are activated

Table 5.1: Notation

[141] where the fiber-deployment cost is minimized or in [142] where the UE data rates are maximized. Other studies, more focused on power and energy-related optimization of IAB networks exist, but they either optimize the energy consumption after the topology and routing have been chosen, such as in [143], or they are restricted to a single-hop architecture, such as in [144], where a low energy multiple access scheme is designed, or [145], where a coordination mechanism between the donor and nodes is designed.

Other related problems also exist in Wireless Sensor Networks (WSN), where it is required to route data on mesh networks [146]. However, while multicommodity-flow with energy optimization has been studied in this context, the problems are quite different: in WSN, the nodes typically are powered by a battery, hence the goal of an energy optimization typically is to maximize the lifetime of the network, *e.g.* to maximize amount of time the network can function using the available batteries or energy sources [147, 148]. On the other hand, in our case there is no notion of lifetime and the nodes are powered by the electrical grid, which means, since we do not consider node failure in this study, that the only reason nodes can turn off is because the network operator tells them to do so.

5.3 Contribution

5.3.1 System Model and Optimization

5.3.1.1 Graph formulation

Let us introduce the problem formally. Note all the notations are summed up in Table 5.1 We start from a weighted directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, called

measurements graph, whose nodes can be either IAB-nodes (donors or other IAB-nodes) or UEs. We denote the set of UEs as $\mathcal{U} \subset \mathcal{V}$ and the IAB-donor as $t \in V$. Note we call the graph we work on a measurements graph because, contrary to the previous chapters, we are working on wireless links, hence, the bandwidth of the links can vary and depend on measurements of the quality of the channel. Each edge (v_i, v_j) of this graph represents a potentially usable wireless link between each node and it is weighted by its available capacity (BW_{v_i,v_j}), which depends on the channel quality. Since the goal is to find a tree representing the routing from each UE to towards the donor t, the edges of the graph will be directed accordingly. Access links (originating from the UEs) will always have UEs as source and IAB-nodes as destination. Backhaul links involving t will always point towards it, as it is always the destination to reach the core. The links between IAB-nodes instead can be used in one or the other direction to build the IAB-tree, so the measurements graph contains a couple of links per each neighbor IAB-node pair. fig. 5.1a and fig. 5.1b report an example of a measurements graph and a possible IAB-tree.

Local detailed information on the feasibility of wireless links between UEs and IAB-nodes is available on each gNBs. The O-RAN architecture allows extensions to standard interfaces so that we can assume that the local information can be collected by an rApp[1], running on the non-real-time RIC, which reconstructs the measurements graph we mentioned above. Then, the optimization algorithm periodically runs and pushes the optimized topology to the RAN through the O1 interface. Note that we take into consideration periodic updates of the topology with a period in the order of minutes, so we assume that disabled nodes wake up to receive an updated topology with a similar schedule. Without loss of generality in the following model we will assume the optimization of a single tree, but the proposed optimization model can be trivially adapted to optimize multiple trees (*e.g.* multiple donors), by linking all the donors to a meta-donor which of which all foreign links have infinite capacity.

5.3.1.2 Optimization problem

The goal of our optimization model is to find a tree $\mathcal{T}$ such that $\mathcal{T}$ is a subgraph of $\mathcal{G}$ rooted in the IAB-donor and whose leaves are all the UEs. $\mathcal{T}$ should minimize energy consumption. Similar to the previous chapter, this metric can be defined as the sum of the static energy, consumed even when a IAB-node is idle and of the dynamic energy, which depends on the number of radio resources the IAB-node has to serve. However, it has been shown [149] that the static energy accounts for more than 70% of

[1] *e.g.* a controller that runs on the non-real-time portion of the RIC, which is supposed to perform management tasks that are not too sensitive

(a) (b)

Figure 5.1: Example of a measurements graph $\mathcal{G}$ (a) and a possible IAB Tree $\mathcal{T}$ (b). IAB-donors are depicted in red, IAB-nodes that are not donors in blue, and UEs in black.

the maximum energy consumed by a gNB. Hence, since the total energy is largely dominated by static energy, we restrict our study to a simple model where only static energy is considered. This yields an objective which is to find the tree $\mathcal{T}$ that minimizes the number of activated IAB-nodes.

Additionally, since the whole network operates using the same spectrum, we assume that each node has a Time Division Multiple Access (TDMA) scheduler that operates using a round-robin policy to schedule the inbound traffic and a dedicated radio device to relay the outbound traffic. This additional constraint—which follows guidance from 3GPP technical documents [150]—differentiates our model from a classical multicommodity flow problem, where adjacent edges do not have to share the same time resources as in a wireless network.

We begin the formulation of the problem as a binary multicommodity flow problem. In such a problem, we have to route a set $\mathcal{K}$ of commodities on the graph, each using a single path. A commodity $k \in \mathcal{K}$ is defined as a triplet s_k, t_k, BW_k^d where s_k is the source node (in our case, a UE), t_k is the destination node (in our case, the IAB-donor t) and $BW_k^d \in \mathbb{R}$ is the bandwidth to reserve on the path from s_k to t_k, in order to serve the traffic by the UE. Note that, in IAB, the donor has a CU, which controls the DUs of the other nodes. This split of the CU and the DU induces overhead traffic. The exact amount of this traffic depends on the exact configuration of the network, but it is a static amount for each configuration (capacity served to the UE, use of MIMO, etc). After trying several different configurations with a dedicated calculator such as [151][2], we observed that the overhead typically corresponds to 10 to 15% of the bandwidth served to the UE. Since the amount is static, we assume it is built into the commodity demands, *i.e.* that BW_k^d is in fact the sum of the capacity demanded by the UE and of the overhead traffic necessary to serve it. These commodities are decided by the Mobile Network Operator (MNO) beforehand, depending on the minimum capacity it wants to guarantee to its customers, and might be differentiated by different classes. The MNO can feed this information

[2]In this calculator, the overhead can be calculated as the difference between the values DL Split 2 BW and DL User BW.

to the rApp running the optimization problem. We denote by $\mathcal{N}_{out}(v)$ the outer neighbors of node v and by $\mathcal{N}_{in}(v)$ its inner neighbors. The cardinality of these sets (*e.g.,* the outer and inner degrees) are denoted by $out(v)$ and $in(v)$, and their sum (the degree of the node) $deg(v) = out(v) + in(v)$. Let us introduce the binary variables $a(v)\ \forall v \in \mathcal{V}$ which indicate whether node v is turned on or sleeping, binary variables $f_k(v_i, v_j)$ which indicate whether commodity k uses edge $(v_i, v_j) \in \mathcal{E}$, and binary variables $f(v_i, v_j)$ which indicate whether (v_i, v_j) is used by any commodity. We define the problem as the following binary non-linear programming problem:

$$\min \quad \sum_{v_i \in \mathcal{V}} a(v_i) \tag{5.1}$$

$$\text{s.t.} \quad \sum_{k \in \mathcal{K}} f_k(v_i, v_j) \cdot BW_k^d \leq BW_{v_i,v_j} \times \frac{1}{\sum\limits_{v_l \in \mathcal{N}_{in}(v_i)} f(v_l, v_i)} \qquad \forall (v_i, v_j) \in \mathcal{E}, \forall k \in \mathcal{K} \tag{5.2}$$

$$\sum_{v_j \in \mathcal{V}} f_k(v_i, v_j) - \sum_{v_i \in \mathcal{V}} f_k(v_i, v_j) = 0 \qquad \forall v_i \in \mathcal{V}, \forall k \in \mathcal{K} \tag{5.3}$$

$$\sum_{v_i \in \mathcal{V}} f_k(s_k, v_i) - \sum_{v_i \in \mathcal{V}} f_k(v_i, s_k) = 1 \qquad \forall k \in \mathcal{K} \tag{5.4}$$

$$\sum_{v_i \in \mathcal{V}} f_k(v_i, t_k) - \sum_{v_i \in \mathcal{V}} f_k(t_k, v_i) = -1 \qquad \forall k \in \mathcal{K} \tag{5.5}$$

$$a(v_i) \geq \frac{1}{deg(v_i)} \left[\sum_{\forall v_j \in \mathcal{N}_{in}(v_i)} f(v_j, v_i) + \sum_{\forall v_j \in \mathcal{N}_{out}(v_i)} f(v_i, v_j) \right] \qquad \forall v_i \in \mathcal{V} \tag{5.6}$$

$$f(v_i, v_j) \geq f_k(v_i, v_j) \qquad \forall (v_i, v_j) \in \mathcal{E}, k \in \mathcal{K} \tag{5.7}$$

$$\sum_{v_j \in \mathcal{N}_{out}(v_i)} f(v_i, v_j) \leq 1 \qquad \forall v_i \in \mathcal{V} \tag{5.8}$$

$$a(v_i), f(v_i, v_j), f_k(v_i, v_j) \in \{0, 1\} \tag{5.9}$$

Our objective in eq. (5.1) is to minimize the number of nodes that are turned on, *e.g.,* the energy consumption of the network. In eq. (5.6), the value of variable $a(v_i)$ is enforced to be 1 if any flow uses node v_i. eqs. (5.2) to (5.5) are multi-commodity flow constraints, where eqs. (5.3) to (5.5) enforce the equilibrium of the flow and eq. (5.2) ensures the capacity constraints are respected. This constraint is different from the classic multicommodity flow problem, in which it would be

$$f_k(v_i, v_j) \cdot BW_k^d \leq BW_{v_i,v_j} \qquad \forall (v_i, v_j) \in \mathcal{E}, \forall k \in \mathcal{K}.$$

In fact, as mentioned above, in a wireless network the edges adjacent to the same node need to share the spectrum, typically by using TDMA with a specific scheduler. In our case, we have assumed that a Round Robin scheduler allocates equal resources to all the adjacent edges. Finally, the constraint in Equation (5.7) ensures that an edge is activated if any commodity uses it and eq. (5.8) makes sure all activated nodes have outer degree 1, which implies the network is a tree.

This model is non-linear because of the inverse function in Equation eq. (5.2). We now propose an equivalent linearized version of the previous model. We prove the equivalence in Theorem 4.

In the linearized model below, we introduce binary variables $x_m(v_i) \quad \forall v_i \in \mathcal{V}$. These variables are equal to 1 iff at least m of the inner edges incident to v_i are activated. This enables us to linearize the inverse function in eq. (5.2) and to replace it with a weighted sum of those binary variables.

$$\min \quad \sum_{v_i \in \mathcal{V}} \sum_{m=1}^{in(v_i)} x_m(v_i) \tag{5.10}$$

$$\text{s.t.} \quad f(v_i, v_j) \cdot BW_k^d \leq BW_{v_i,v_j} \cdot \left(x_1(v_j) - \sum_{m=2}^{in(v_j)} \frac{x_m(v_j)}{(m-1)m} \right) \quad \forall (v_i, v_j) \in \mathcal{E} \tag{5.11}$$

$$x_m(v_j) \geq \left(\sum_{v_i \in \mathcal{N}_{in}(v_j)} f(v_i, v_j) - (m-1) \right) / in(v_j) \quad \forall v_j \in \mathcal{V}, \forall 1 \leq m \leq deg(v_j) \tag{5.12}$$

$$x_m(v_i) \in \{0,1\} \quad \forall v_i \in \mathcal{V}, \forall 1 \leq m \leq deg(v_i) \tag{5.13}$$

(5.3), (5.4), (5.5), (5.7), (5.8), (5.9)

Theorem 4. *The BNLP (5.1) - (5.9) has the same optimal solution as the BLP (5.3) - (5.13).*

Proof. Let us first observe that $\left(\sum_{v_i \in \mathcal{N}_{in}(v_j)} f(v_i, v_j) - (m-1) \right)$ is always positive if at least m inner edges of v_j are activated, and is nonpositive otherwise. also note that this sum is always lower or equal to $in(v_j)$. Hence, the right-hand side of eq. (5.12) is between -1 and 1, and its value is positive if m inner edges are used. This, combined with the fact we are minimizing the sum of variables $x_m(v_j)$ means that in an optimal solution to problem (5.3) – (5.13), $x_m(v_j)$ will be equal to 1 if at least m inner edges of v_j are

activated and 0 otherwise.
Let us now observe that in eq. (5.11), if n inner edges of v_j are activated, then $x_1(v_j), x_2(v_j), ...x_n(v_j)$ will be equal to 1. It follows that the sum $x_1(v_j) - \sum_{m=2}^{in(v_j)} \frac{x_m(v_j)}{(m-1)m}$ will be equal to $\frac{1}{n}$, *e.g.* the constraint is equivalent to constraint (5.2). Finally, observe that since we are building a tree, minimizing its number of edges is equivalent to minimizing its number of nodes, as a tree of n nodes always has exactly $n-1$ edges, meaning the objective function, Equation (5.10), is equivalent to the objective in Equation (5.1). □

5.3.2 Performance Evaluation Setup

This section presents the techniques used to synthetically generate the set of measurements graphs $\mathcal{G}(\mathcal{V}, \mathcal{E})$, needed to evaluate the feasibility and effectiveness of our optimization model. In particular, we will be using datasets representing an area of 0.092km^2 in the center of Milan, Italy. To do so, in the first subsection, we will describe the state-of-the-art techniques used to place the IAB-nodes [152], and the UEs [153]. Then, in the second subsection, we present our channel model, based off 3GPP specifications combined with ray tracing. Finally, the third section presents our data-driven time-varying UE density model, which enables us to generate different instances depending on the time of the day and the day of the week.

5.3.2.1 Placement of gNBs and UEs

The set of nodes of our graph $\mathcal{V}$ is comprised of both IAB-nodes, and UEs, whose placement is done separately using two different techniques. IAB-nodes are placed on building facades with a given density λ_{gNB}. The exact position is computed by using a state-of-the-art placement heuristic [152] that exploits highly precise 3D models to place the gNB such that the number of potential UEs' location in line of sight is maximized. UEs are then randomly distributed both in public areas, such as streets, and inside buildings. Specifically, given a density of λ_{UE}, indoor UEs are uniformly randomly distributed inside buildings with a density equal to $r_{i/o} \cdot \lambda_{\text{UE}}$ and outdoor UEs are uniformly randomly distributed inside buildings with density $(1 - r_{i/o}) \cdot \lambda_{\text{UE}}$, where $r_{i/o}$ is a commonly used ratio of indoor to outdoor UE equal to 0.8 taken from 3rd Generation Partnership Project (3GPP) technical report [153]. In short, we consider that in our simulations 80% of the UEs are placed indoors. fig. 5.2 shows a deployment with $\lambda_{\text{gNB}} = 45$ and $\lambda_{\text{UE}} = 900$ UE/km^2.

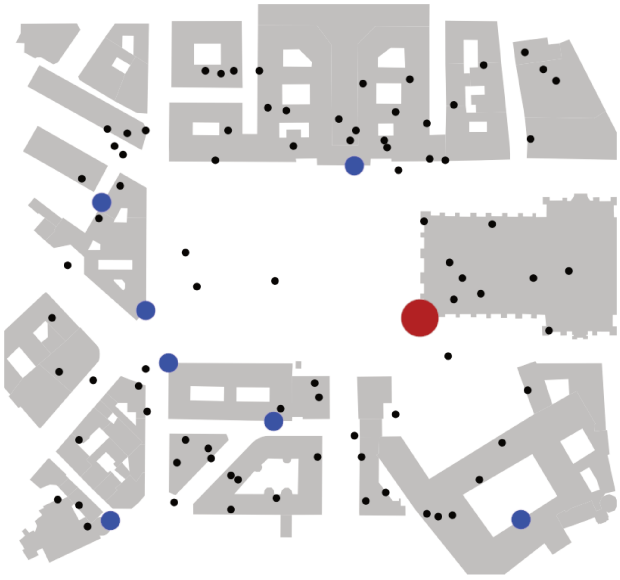


Figure 5.2: Sample deployment of a network in the center of Milan, with 1 IAB-donor (in red), 7 IAB-nodes (in blue), and 83 UEs (in black). It corresponds to $\lambda_{\text{UE}}(9) = 900$ UE/km^2 (Mon 9am).

5.3.2.2 Access and Backhaul channel models

Once the location of both UEs and IAB-nodes have been determined, we evaluate the path loss by applying the 3GPP Urban-Micro (UMi) stochastic channel model [154]. However, instead of using the stochastic Line of Sight (LoS) probability model provided by the same UMi model, we deterministically evaluate the LoS by employing ray tracing analysis on the same 3D models used to find the optimal locations, obtaining a more accurate estimation [155]. For indoor UEs, we always consider them to be Non-LoS (NLoS) and we add the additional Outdoor to Indoor (O2I) penetration loss. Since the buildings in the area we consider are mostly made out of concrete, we use the high-loss O2I model [154].

Finally, we compute the Signal-to-Noise-Ratio (SNR) using the thermal noise and by adding the receiver noise figure, then we calculate the Shannon capacity, which gives us the bandwdith of each potential link. Both access and backhaul are assumed to be using the same frequencies, but different values of antenna gain and numbers of MIMO layers are used. table 5.2 details all the values used in our simulations, which are aligned with typical literature and 3GPP studies on this topic.

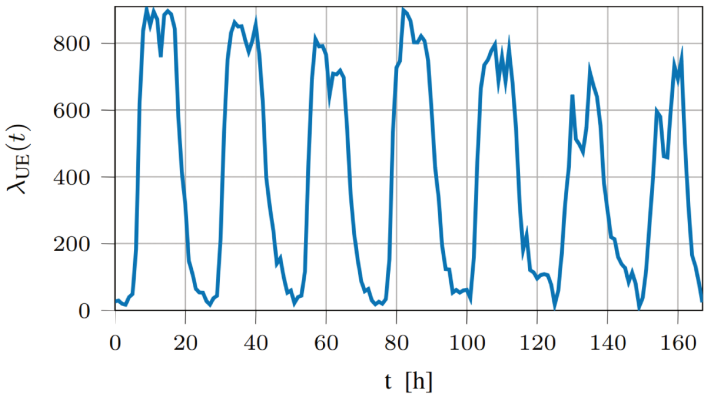


Figure 5.3: Weekly profile for the UE density in central Milan.

5.3.2.3 Time-varying UE density model

As explained in more detail in Section 5.1, most studies dealing with topology optimization focus their analysis on a single, or a handful, value of λ_{UE}. Since the energy optimization technique we devise tunes the IAB-node activation on the basis of the number of UEs and their load, we need to evaluate our model on a large number of values of UE density, ideally following a realistic trend. Therefore, we employ a technique used in similar research [156] to devise a time-varying UE density model. First, we extract the cell load profile $p(t)$ related to our analysis area, in Milan, from openly available datasets [140]. We then normalize it in the range $(0, 1]$, and we model the UE density as a function of time $\lambda_{UE}(t) = p(t)l\lambda_{gNB}$, where $l = 10$ is the number of UEs per gNBs taken from the 3GPP technical report [153]. Finally, we generate a set of 168 graphs spanning an average week with a one-hour granularity.

fig. 5.3 reports the hourly trend of $\lambda_{UE}(t)$ corresponding to the area of our analysis, showing how for several hours every night the network has to serve almost no UEs and how in the weekends, even at peak hours, the density of UEs never exceed 80% of the weekday peak hours.

5.3.3 Results

As already mentioned in section 5.1, we evaluate our model on a 0,092km^2 area in the center of the city of Milan (Italy), for which we computed the UE density trend λ_{UE} of an average week. For each hour of the week (168 in total), we generate the measurements graph as described in the previous section and then we run our optimization algorithm on it. We compare the trees found by our solution with 4 strategies:

- **All donors**, a dense deployment without IAB, where all the gNB are wired. It is an upper bound in terms of energy consumption and

Parameter	Value
Area size	0.092 km^2
UE density range	[0-900] UE/km^2
Indoor/Outdoor UE ratio	80/20
Carrier frequency	28 GHz
Bandwidth	100 MHz
Noise Figure	5 dB
O2I Loss	14.15 dB
Reception gain (Access/Backhaul)	3 / 10 dBi
MIMO layers (Access/Backhaul)	2 / 4
Backhaul transmission power	30 dBm
Minimum Capacity per UE	80Mb/s
Number of independent simulation runs	10

Table 5.2: Simulation Parameters

capacity. Additionally, no re-distribution of the UEs is performed as they are always attached to the gNB with the lowest SNR.

- **No relays**, a deployment where all the IAB-nodes are not active. It is a lower bound in terms of energy and capacity.
- **Widest Tree**, a strategy that employs the well-known widest path algorithm to find the path of maximum capacity (*e.g.,* with the largest bottleneck in terms of capacity) from each UE towards the donor and deactivates all the IAB-nodes that are not part of any path.
- **Optimized Tree**, our optimization model.

In the first part of this section, we compare the energy consumption (both in terms of the number of nodes activated and of the overall number of gNB-hours) of the different algorithms. Then, in the second part, we evaluated the topologies in terms of bottlenecks of the downlink capacity, and finally, we compare the runtimes of the algorithms.

5.3.3.1 Energy Consumption

To evaluate the energy consumption of the IAB networks we first show the hourly number of active IAB-nodes, then we introduce a metric that measures the total number of hours each IAB-node has been active. The IAB-donor is not taken into consideration as we always need at least one node to be active to provide a minimum service to the users.

Figure 5.4 shows the number of IAB-nodes activated, on the left axis, and the number of UEs connected to the network, on the right axis. To improve the readability only the values for the first day of the week have been reported. **Optimized Tree**, shows that it is possible to fully deactivate the IAB-nodes at nighttime (from 12 pm to 4 am) and that also during daytime several IAB-nodes can be deactivated. By comparing its

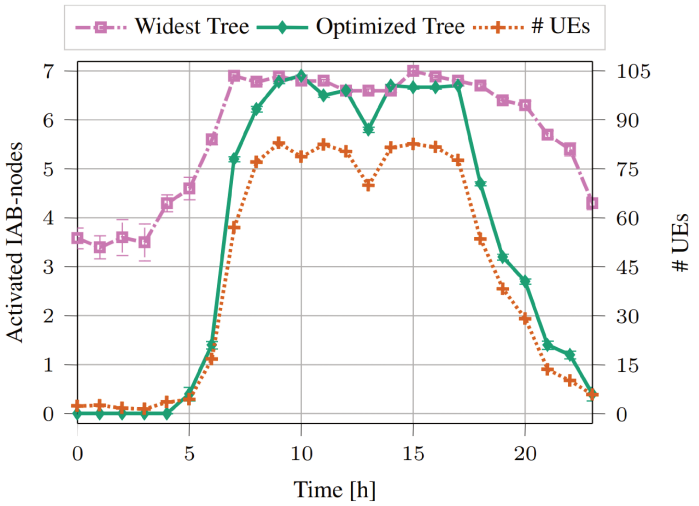


Figure 5.4: Number of IAB-nodes activated in the over an average Monday.

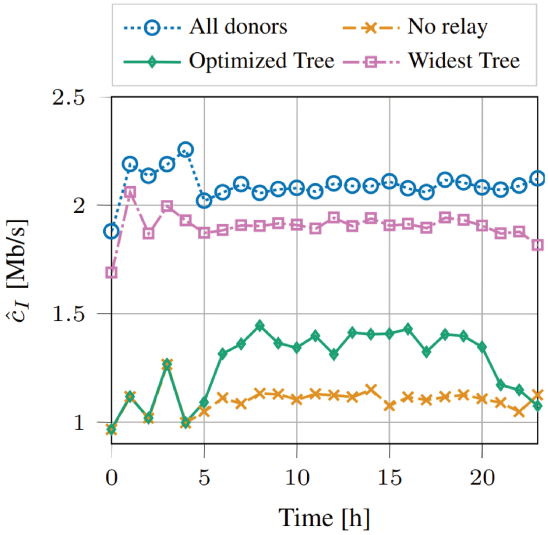
trend with the number of UEs, we can also see that it gets perfectly followed, highlighting the effectiveness of our optimization model. **Widest Tree**, on the other hand, never manages to deactivate more than 3 IAB-nodes, highlighting that a specific algorithm is needed to fully implement energy-saving policies.

Additionally, by integrating the number of activated gNB at each hour for the span of the week we obtain the total number of gNB-hours for each strategy. For **No relays**, the value is $168h$, as only one gNB is always active. For **All donors**, on the other hand, the total number of gNB-hours is equals to $168h \cdot 8 = 1344h$, since 8 IAB-donors are active at all times. More interestingly, the values for **Widest Tree** and **Optimized Tree** respectively activate the RAN for $1141h$ and $709h$, which means our method improves the power consumption of 47% over **All Donors** and 38% over **Widest Tree**.

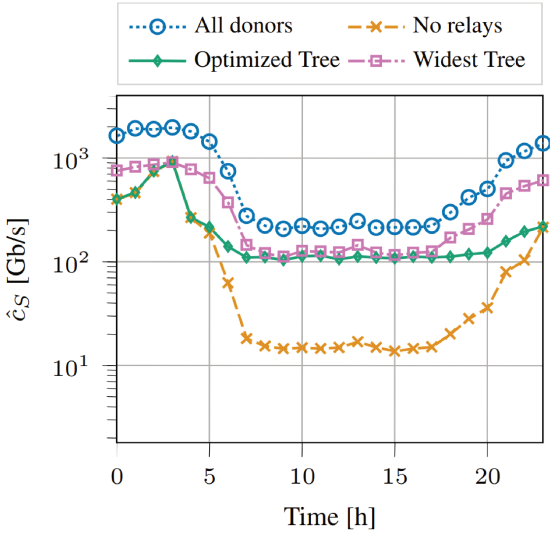
5.3.3.2 Capacity

To evaluate the performance of the topology we analyze the capacity served to each UEs with three different capacity metrics, which are shown in fig. 5.5 and detailed below. First, let us define some functions used throughout the section. Let $\mathcal{P}(u,t)$ be the function returning the set of edges forming the path from u to t over our topology tree. Let $N_{in}(t)$ the number of edges directed towards t in the tree and BW_{v_i,v_j} the capacity of the edge (i,j).

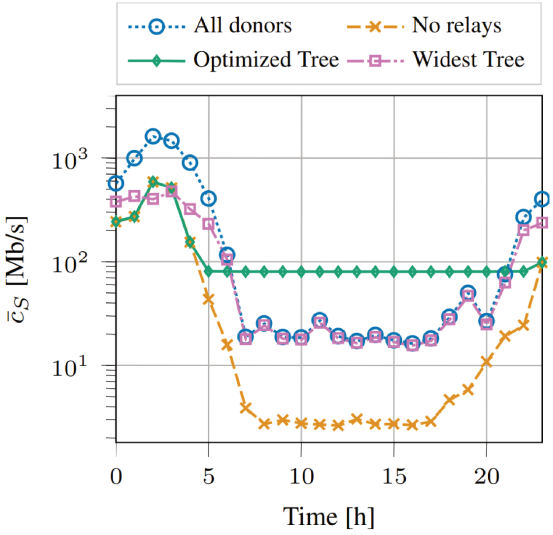
The first metric, called Average Idle Capacity measures the average theoretical capacity of UEs, *e.g.* the capacity that would be attainable if the network resources were completely unused. As detailed in eq. (5.14) below, it is computed as the minimum capacity (the bottleneck) of the



(a) Average idle capacity.



(b) Average saturation capacity.



(c) Minimum saturation capacity.

Figure 5.5: Capacity Metrics for the first 24h of the week (Monday).

edges over the path between each UE (u) and the donor (t). Which is then averaged across all the UEs $u \in \mathcal{U}$. This metric represents an upper bound on the capacity per UE.

$$\hat{c}_I = \frac{1}{|\mathcal{U}|} \sum_{v_i \in \mathcal{U}} \min_{(v_j, v_l) \in \mathcal{P}(v_i, t)} BW_{v_j, v_l} \tag{5.14}$$

Figure 5.5a shows the Average Idle Capacity for the four different strategies on the first 24 hours of our week. The first insight provided by this figure is that the maximum capacity per UE does not depend on the load of the network. Moreover, as we were expecting **All donors** and **No relay** are respectively the upper and lower bounds in terms of capacity. **Widest Tree**, the strategy that maximizes the bottleneck between each UE and the donor, manages to achieve a capacity very close to the upper bound (8% lower). **Optimized Tree** instead shows a more significant drop with a loss of 35%. The drop can be explained by the minimum capacity constraint that, instead of letting each UEs reach the IAB-donor through the widest path, in certain cases picks paths worse in terms of maximum capacity that instead guarantee the minimum capacity.

The second metric, called Average Saturation Capacity and detailed in Equation (5.15), is formulated in a very similar way as Equation (5.14). However, here we assume that all the UEs try to access the network at the same time, thus we divide the capacity of each edge $BW_{s,t}$ by the number of inner neighbors of the node t, since those edges share the same resources through the scheduler.

$$\hat{c}_S = \frac{1}{|\mathcal{U}|} \sum_{v_i \in \mathcal{U}} \min_{(v_j, v_l) \in \mathcal{P}(v_i, t)} \frac{BW_{v_j, v_l}}{N_{in}(v_l)} \tag{5.15}$$

As in the previous metric, also here **All donors** and **No relays** behaves respectively as upper and lower bound. The difference between the two other strategies, and their distance from the upper bound drops sharply. In fact, at peak time **All donors** is capable of delivering roughly 200Mb/s per UE, while **Optimized Tree** and **Widest Tree** respectively deliver 115 and 130 Mb/s per UE.

The third metric, called Minimum Saturation Capacity and detailed in Equation (5.16), measures the capacity delivered to worst UE while the network is under saturation by all the UEs, *e.g.* it defines the minimum level of Quality-of-Service provided by the topology. It is defined similarly to the previous one, but instead of averaging over the UEs we take the worst value.

$$\bar{c}_S = \min_{v_i \in \mathcal{U}} \min_{(v_j, v_l) \in \mathcal{P}(v_i, t)} \frac{BW_{v_j, v_l}}{N_{in}(v_l)} \tag{5.16}$$

fig. 5.5c shows that **Optimized Tree** is the only strategy that manages to guarantee the minimum level of service, equal to 80Mb/s during peak hours (7 am-16 pm), while also minimizing the excessive capacity at night time. Indeed, as shown in fig. 5.5b, some of the other algorithms give a better average capacity, but don't manage to provide the required minimum. This means those schemes are highly unfair, and that some users obtain a huge portion of the bandwidth, at the cost of leaving little capacity to the other users. In comparison, with **No relays** we measure a minimum level of service that, at peak time, is one order of magnitude lower than the minimum level of service (between 2 and 7 Mb/s) while **Widest Tree** and **All donors** behave similarly in terms of minimum capacity, as they can both take advantage of all the IAB-nodes available. However, since the UEs are not load-balanced across all the available IAB nodes, the minimum level of service is not met. We also note that with **All donors** there is also an excess of capacity at night time; when energy-saving policies could deactivate several IAB-nodes, moreover despite being in a more favorable position where no routing is to be performed beyond the first link between the UEs and the BS, **All donors** still has less capacity than the Optimized Tree. This emphasizes the importance of balancing the load of UEs between base-stations

5.3.3.3 Runtime

Finally, we compare the algorithms in terms of runtime. The runtimes are depicted in Figure 5.6. What this plot shows is that the energy optimization problem becomes much harder when the UE density increases, as, during the peak hour, the optimization problem becomes much harder (we reach the time limit of 10 minutes). The other algorithms also become slightly slower, but the slowdown is negligible.

5.4 Conclusions

In this chapter, we have modeled and solved the problem of finding an IAB topology optimized for energy efficiency. We find optimal solutions to the problem, which enables us to save up to 47% of energy compared to the baseline, while still respecting capacity constraints, which other approaches cannot do. Our results hence show the importance of considering energy-efficiency as a core feature of IAB topology design. However, the main limit of this work is that we have to re-run the algorithm regularly as the user demands change. We have several ideas to solve this flaw in future works. One is to improve the speed by finding a better formulation, and to find fast heuristics or Reinforcement Learning algorithm. Those solutions could leverage the fact that in practice, the changes in demand are often very

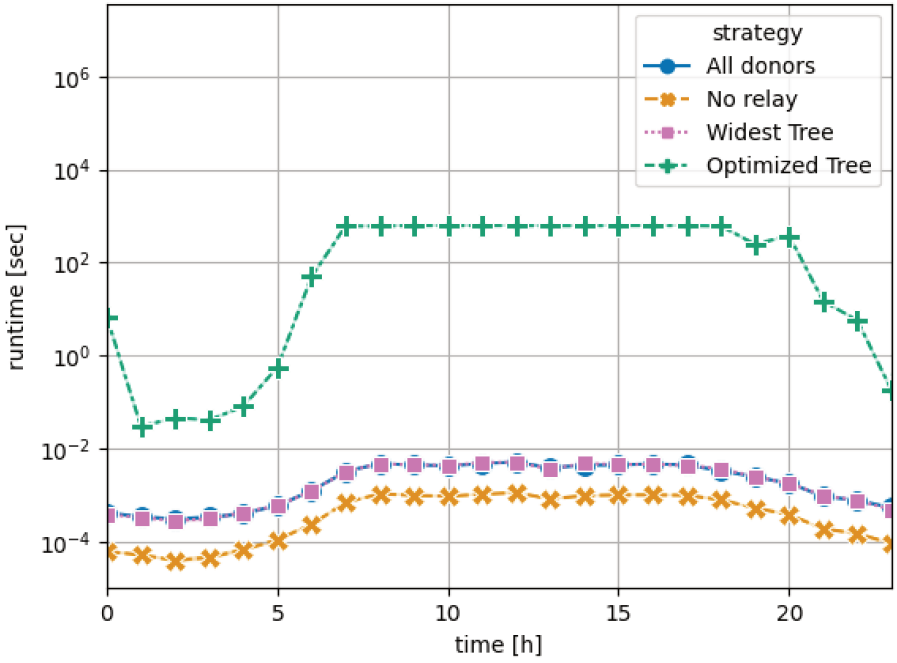


Figure 5.6: Runtime of the algorithms for the first 24 hours of the week

progressive, hence the solution to the previous set of demands could inform the solution to the current demands. Another avenue for research is to formulate the problem directly as an online optimization problem. A second way to improve upon this work is to make the energy model more precise. In that regard, this chapter only considered the static energy consumption, because, as explained before, it is the main part of the energy consumed by the RAN. However, it would be interesting to also consider the energy consumed due to using more radio resources. In that regard, it is currently hard to find fine-grained models for the 5G network. Furthermore, another question we did not consider is that of the "turn-on time", *e.g.*, in practice, the gNB might not turn on instantaneously, which can be problematic if the IAB topology has to change very often. Another limitation of this work is that it is mostly focused on the downlink, while the uplink is also an important part of the network, especially in 5G. We argue that, since we considered a TDMA scheduling, reserving bandwidth on a link would be implemented as reserving timeslots. Hence, the bandwidth reserved could either be used for the uplink or the downlink. However, evaluating the robustness of our algorithm with both uplink and downlink traffic is still to be done (especially, it has to be taken into account in the choice of the channel model), and we plan to do it in the extended journal version.

Chapter 6

General Conclusion and Perspectives

6.1 Conclusion

In conclusion, this PhD thesis has addressed critical challenges in energy and resource optimization within 5G sliced networks, presenting algorithms that leverage techniques from the fields of graph theory (Chapters 3, 4, 5), optimization (Chapters 4, 5), reinforcement learning (Chapter 3), and network design (Chapters 4, 5). The overarching goal of achieving energy and resource efficiency in 5G networks, while catering to diverse service requirements through network slicing, has been systematically addressed through a multi-faceted approach.

One of the primary contributions of this research lies in designing a new method for minimizing the resource usage of the core network of sliced 5G networks (Chapter 3). This was done by applying Nested Rollout Policy Adaptation (NRPA), a Monte Carlo Search Reinforcement Learning (RL) methodology in conjunction with Neighborhood Search, which to our knowledge had never been done for any problem. This approach has demonstrated its efficacy in solving the core network slice placement problem, formulated as a Virtual Network Embedding problem, with a better number of accepted slices than state-of-the-art approaches. This chapter led to one conference [7] and one journal publication [8]. There also is an ongoing effort to implement the algorithm in practice, using a platform from the lab based on the core network of OpenAirInterface [13].

Then, we focused on minimizing the energy consumed by the edge network (Chapter 4) with an exact method. This has been achieved by the integration of Mixed-Integer Linear Programming (MILP) and column generation techniques. Particularly, the technical challenge which

we adressed was to finely model the quality-of-service constraints in probabilistic terms, leading to a non-linear problem which we showed how to linearize. Since the problem was very slow to solve in practice, we also proved theroretically that considering energy optimization is inherently harder than other classic objectives such as minimizing resource consumption. This was achieved by proving that a simplified version of the problem which is polynomial with a classical objective becomes NP-hard with energy minimization. This chapter led to one conference publication [9], while another one is currently being prepared (see preprint [125]) along with a future journal extension where we will consider how the algorithm can be adapted to the online case where edge slices arrive in the system over time.

In Chapter 5, we optimized the energy for the integrated access and backhaul network, also utilizing MILP, which can further contribute to paving the way to greener 5G networks. This work enables to guarantee per-user bandwidth in highly dense networking environment while ensuring the energy consumption of the network remains low. This is done by seeing the user bandwidth as hard constraints and the energy as an objective, contrary to the existing methods which were based on heuristics. This paper led to one conference publication [157] and a journal extension, where more precise energy and wireless channel models will be considered is currently in preparation. Similar to the previous chapter, this journal extension will also consider online optimization so the solution can evolve as users come and go, without recomputing the full solution each time.

6.2 Future Directions

Overall, the findings presented in this thesis underline the feasibility and benefits of employing advanced optimization techniques and intelligent learning algorithms to enhance energy efficiency in 5G sliced networks. The methodologies applied to the core network, edge network and integrated access and backhaul optimization collectively contribute to a comprehensive energy optimization strategy. This thesis also underlines the main challenge of designing energy-efficient routing strategies: the energy optimization objective makes the problem much harder than classical objectives such resource-efficiency. We observed this insight both theoretically and empirically.

Leveraging the structure of the Network

Hence, there still is a lot of work towards greener 5G networks, as the problems we solve typically are slow to solve optimally. Several solutions could exist to solve this challenge. The obvious one would be to design heuristics and approximation algorithms, but another key, unexplored possibility would be to leverage the structure of the practical networks: we have noticed that, in the Topology Zoo dataset, a large proportion of the networks are planar and have low treewidth[1]. It could be interesting to leverage these property in order to accelerate our algorithms and devise some approximation algorithms with guaranteed performance, enabling us to solve the problem faster without sacrificing too much of the theoretical guarantees coming with exact algorithms.

End-to-end optimization

Second, in this thesis, we have designed solutions for each part of the network in isolation. Now that we have a better understanding of each of those parts, a key future direction is to instead model and optimize these aspects in an end-to-end manner, as the interaction between the optimization of the different parts is, to our knowledge, an unexplored challenge. Of course, we have seen in this thesis that each problem taken is isolation is itself hard, hence taking all the aspects into account will probably require to design new techniques that are able to scale with the size of the network while not sacrificing accuracy.

However, in order to model the end-to-end optimization, one has to study the interaction of the different components in practice. This can only be done by collecting data on a real system, which to our knowledge, does not exist as a research platform. Hence, in the future, it would be desirable to implement those end-to-end solutions in a practical manner. For the edge and core parts, this could be done using SDN routers to perform the link placement, combined with Kubernetes-based core network functions (such as Kube5G [160]) as well as implementing tenant applications (video, AR, AI, ...) as Kubernetes containers. This part of the network could be connected to one of the existing RAN experimentation platforms such as OpenAirInterface [13] or SRSRAN [12]. In addition to that, while some standardization effort have been carried out separately (*e.g.* with the OpenRAN for the RAN part or with OpenFlow for the SDN part), to our knowledge standardizing the interfaces that would apply the end-to-end control of mobile networks is still to be done.

[1] see [158] for a description of treewidth and how it is usually feasible to design fast algorithms for low-treewidth graphs, and [159] for measurements of TopologyZoo treewidth

Due to the complexity of such a system and the necessary reactivity, in an end-to-end context, we believe the approaches presented in this thesis should be combined with some data-driven system, able to orchestrate them online. Hence, we believe it would be key to combine the methods presented in the thesis with reinforcement learning algorithms, *i.e.* to keep going with the design philosophy of the NEPA algorithm presented in the second chapter, which combines Reinforcement Learning and Neighborhood Search. The possibilities to do so are numerous:

- Modeling the other presented problems as MDPs, which would then enable us to apply an NRPA-based algorithm, and possibly to improve the parts of the solution for each region of the network with a different heuristic tailored to each case and sub-objective. Another avenue for optimizing such a system with an extremely large number of states would be to use options reinforcement learning [161], in which actions can be aggregated as options (i.e. a chain of actions), which implicitly reduces the number of states.

- Leveraging the learn-to-branch [162] techniques, which use a Neural Network to guide a branching algorithm (such as the one of Chapter 4). To our knowledge there is a dearth of works on this topic in the contexts of multicommodity flows or of column generation.

- Using differentiable convex optimization [163] [164], which offers the possibility to use a mathematical program as part of a Neural Network

RAN optimization

Finally, another key aspect that we want to explore more in the future is the RAN. While this thesis does not present work on RAN optimization, some preliminary work has been carried on the topic towards the end of the PhD, mainly consisting in learning to use a private version of OpenAirInterface which includes radio slicing (*i.e.* where it is possible to allocate radio resources to different groups of UEs) and modifying it to measure the application delay between the UEs and the gNB. This lead us to discover several challenges in RAN energy optimization. The first finding is that performing accurate measurements of the QoS metrics (mainly the delay) is in itself a hard problem in practice, and that modeling the delay as a function of the allocated resources, while taking into account things like interferences and channel quality is also a key unsolved challenge. On top of this, we discovered that the RAN energy models are mostly coarse-grained and hence imprecise. For example, the main model in the litterature is [165] (it also has a few extensions) but it still does not fully model aspects such as 5G sleep modes, base-station

parameters or radio scheduling. A sign of such a lack is for example that Huawei recently launched a public challenge to try to model the energy consumption given the parameters of real base-stations [166].

Regarding the RAN, experimenting with the OAI platform enabled us to realize that the radio scheduling part is an aspect that is not well adressed by the current OpenRAN standardization effort: the existing platforms and specifications only let the tenants demand more or less radio resources for their slices. However, the way those radio resources are scheduled is still something that remains hardcoded in the platform. We argue it would be of key importance for future networks to enable slice tenants to inject more logic in the slice scheduling policy, depending on their usecase, metrics and the knowledge they have of their user traffics. This of course entails numerous challenges, especially in terms of security, but would also open the possibility to design a lot more innovative scheduling algorithms and test them on real mobile networks.

Furthermore, the possibility for customizing more the scheduling opens several theoretical directions. First, it would be possible to evaluate the wealth of schedulers that have been proposed in the litterature, and to evaluate them while having to deal with real channels, data and signaling traffic. Second, it would enable us to include the scheduling of the radio resources of the slice in the end-to-end slice optimization, as the scheduling has an influence on the amount of resources required in other parts of the network, as well as on the various metrics such as the delay. Third, enabling the slice tenant to inject their scheduling logic opens a large field of research, where a scheduler can be customized to a specific traffic or set of clients. For example, it would be possible to train reinforcement learning scheduling agents which could leverage some complex traffic patterns and specific objectives of some given slice and then use them only for that slice.

Appendix A

Large version of figures

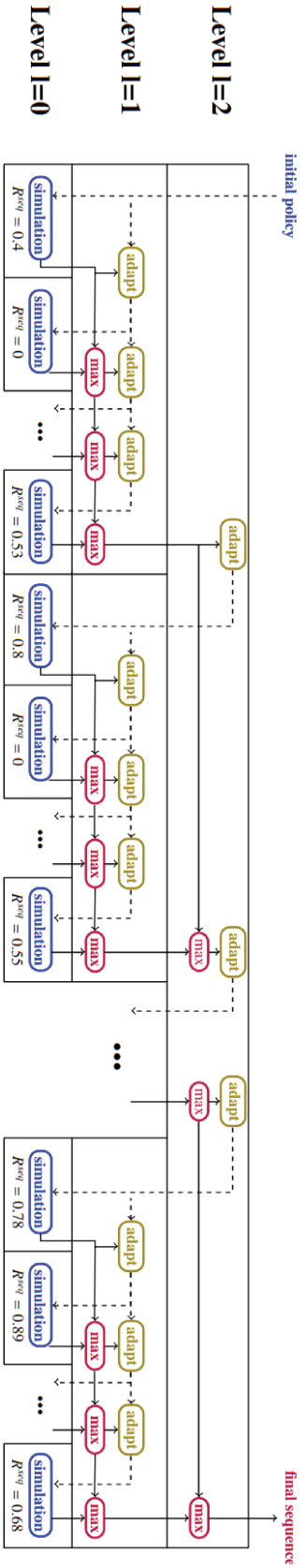


Figure A.1: Example execution of NRPA for l=2. Dashed arrows represent the policy going from one function to the other, while plain arrows represent sequences returned between functions. A function needs to get values from all its predecessor before executing.

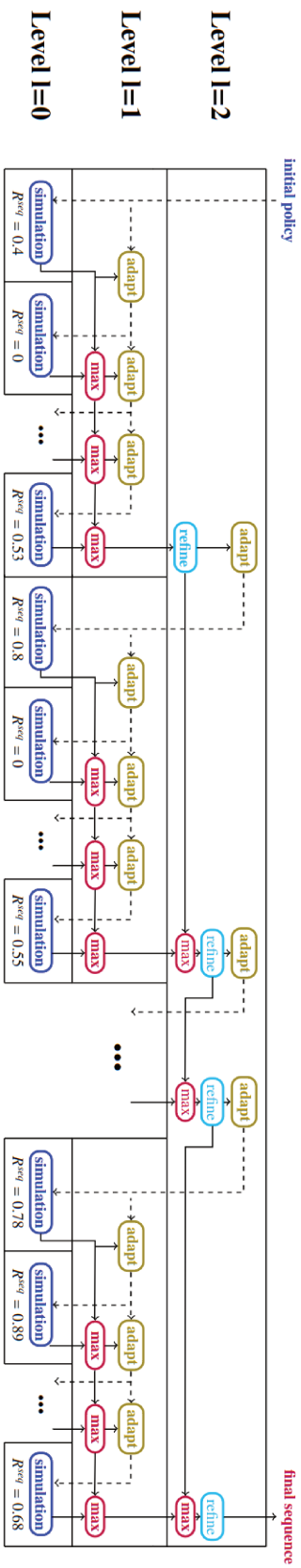


Figure A.2: Example execution of NEPA for l=2, l'=2. Dashed arrows represent the policy going from one function to the other, while plain arrows represent sequences returned between functions. A function needs to get values from all its predecessor before executing.

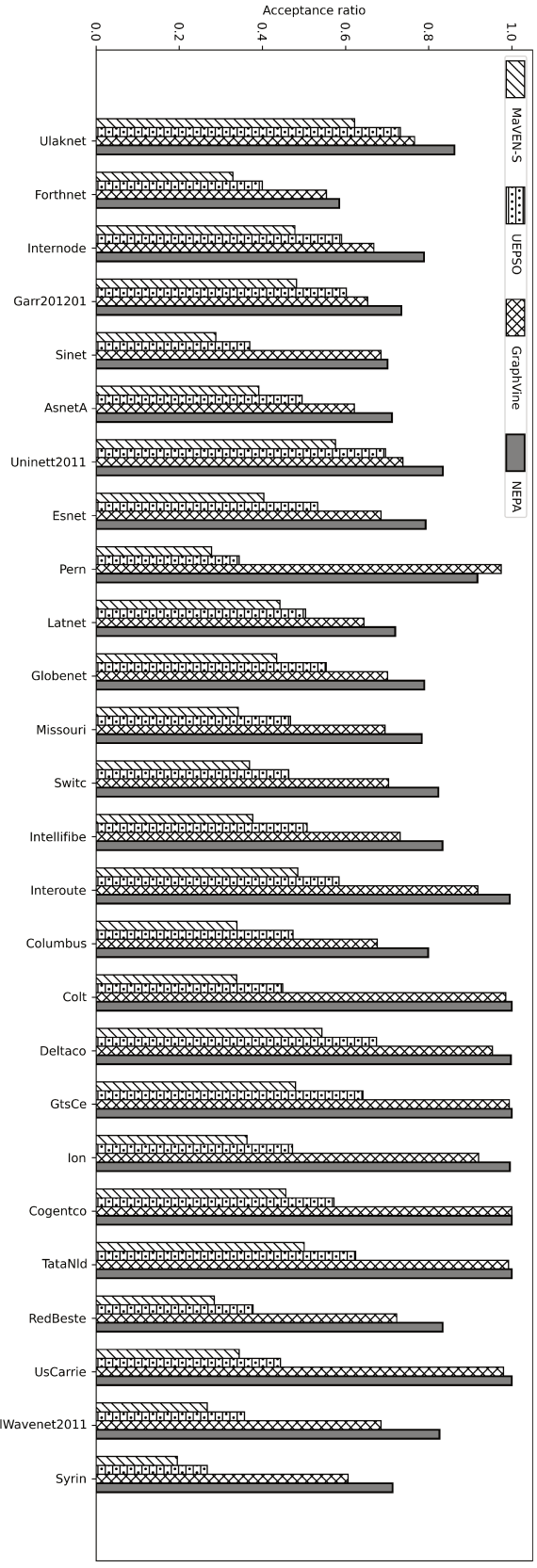


Figure A.3: Acceptance on real physical networks. Results are ordered by increasing shortest-path length variance.

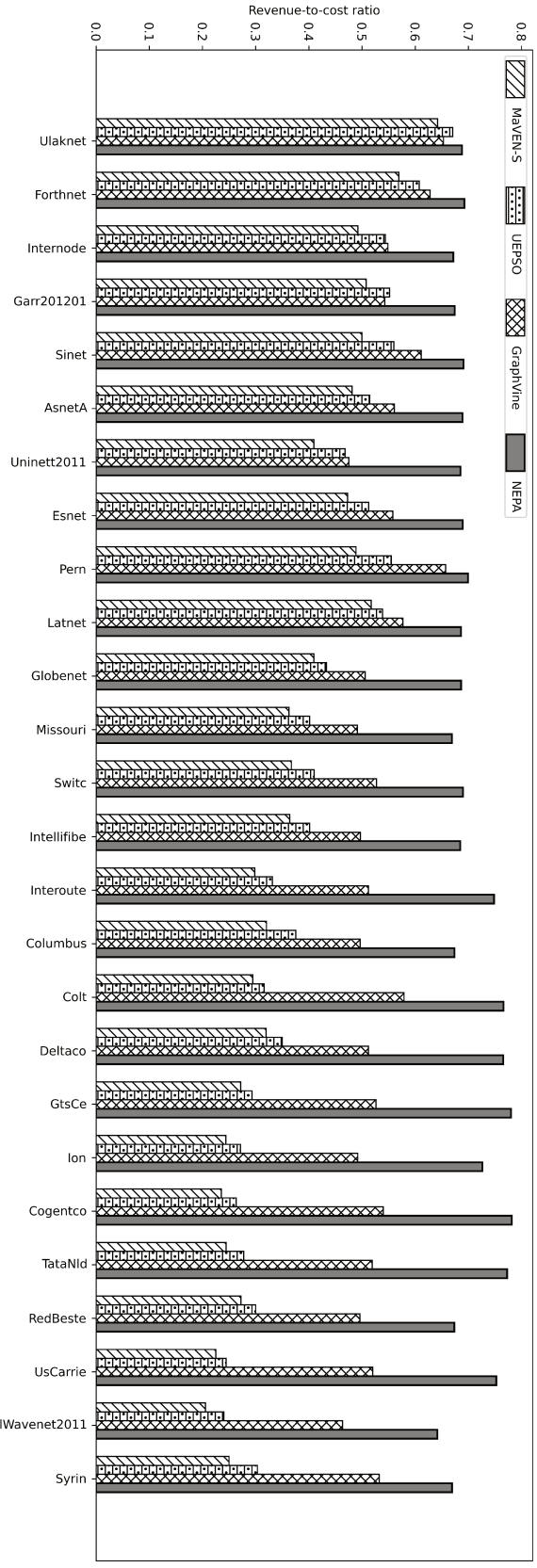


Figure A.4: Revenue-to-cost ratios on real physical networks.

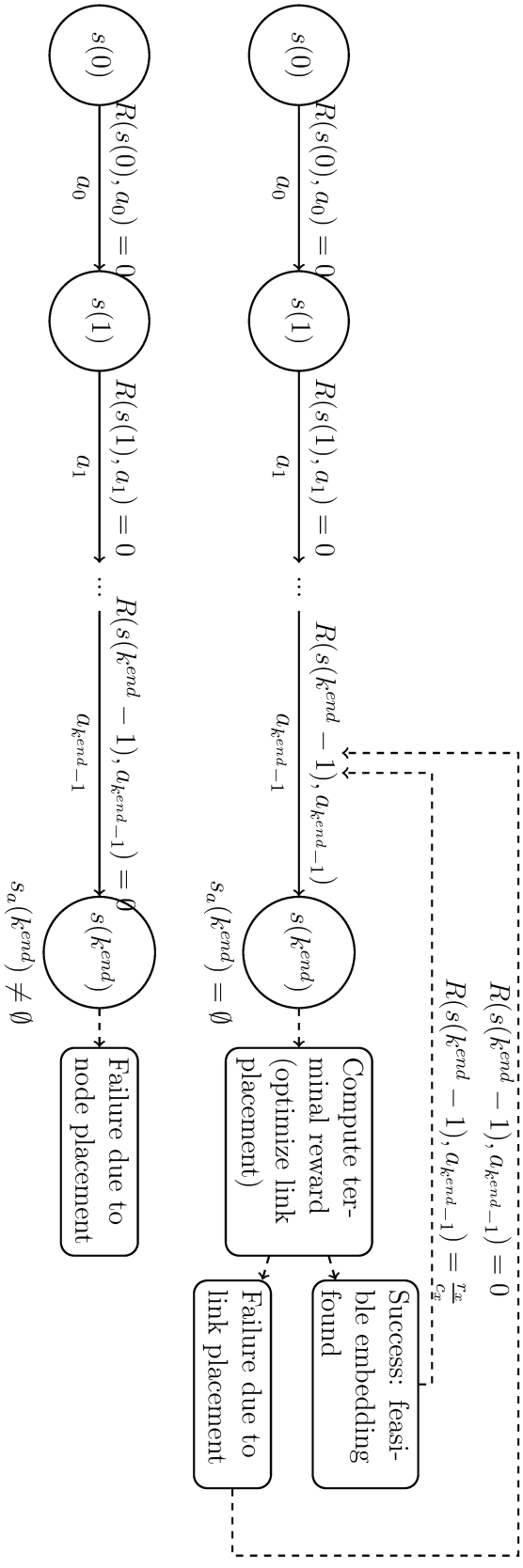


Figure A.5: Example sequences of actions in the MDP. Dashed arrows are transitions not occurring in the MDP (no action choice).

Appendix B

Comparison of results for NEPA with and without the Alternative Reward Function Based on Degrees

In this Appendix, we expose our simulation results on all tested instances when we use the AFBD (alternative function based on degrees) proposed in [43] in the reward function. This function uses a combination of the sum of Bandwidth used and the degrees of the used nodes as a cost function. The formula for the AFBD cost function for virtual network $\mathcal{H}^x$ placed on $\mathcal{G}$ is the following :

$$AFBD(\mathcal{G}, \mathcal{H}^x) = \sum_{\forall (v_i, v_j) \in \mathcal{E}} B\bar{W}^x_{v_i, v_j} + \sum_{v_i^x \in \mathcal{V}^x} deg(host(v_i^x)) - deg(v_i^x)$$

Where $deg(v_i)$ is the degree of node v_i and $host(v_i^x)$ is the physical node hosting virtual node v_i^x.
The idea behind that choice is to keep minimizing the length of the used paths, but while preserving resources on high degree nodes when possible. This revolves around the intuition that higher degree nodes tend to offer more link embedding possibilities, hence, if a virtual network can be placed by using more constraining physical nodes, it should be done, since some future virtual networks might require less constrained ones in order to be placed. In [43], the authors claim to achieve improvements (in the order of a several percents of acceptance) that could be transferred to other meta-heuristic algorithms.
We try to find out if this is the case with NEPA, since after our investigations it is the best performing algorithm at our disposal. Note that the reference meta-heuristic used in [43] is Harmony Search, which has comparable performances to UEPSO, as shown by the same authors in [167] (in

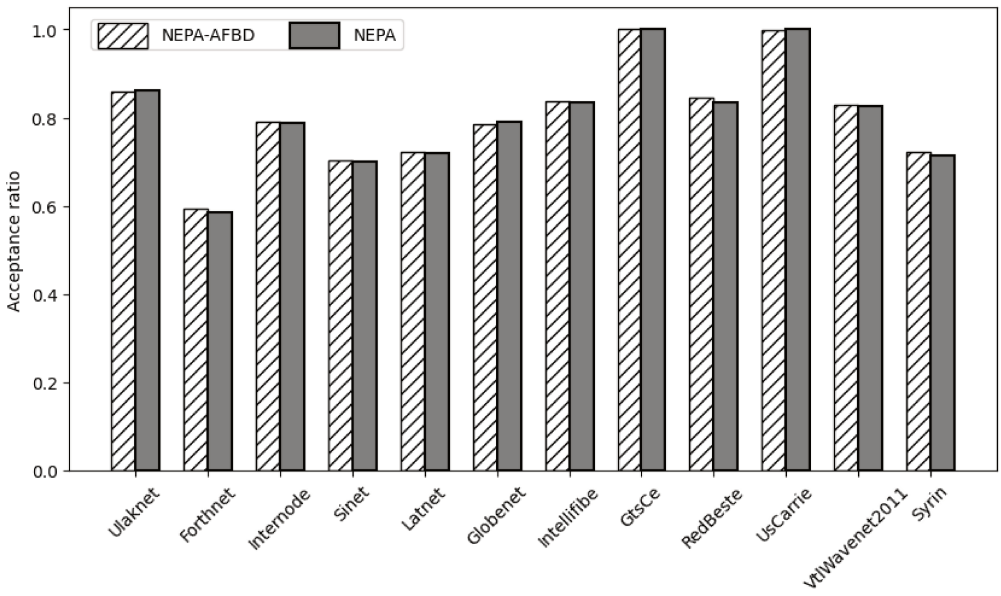
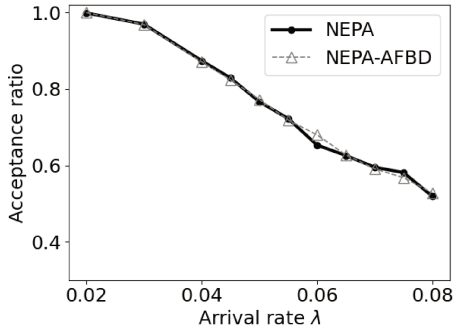


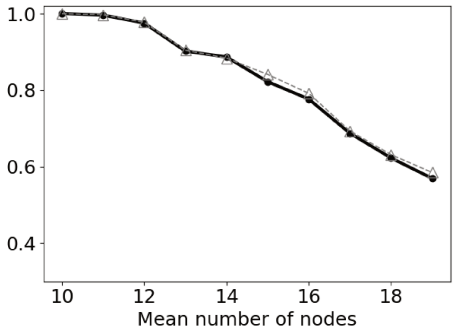
Figure B.1: Acceptance for several real topologies

that article UEPSO is referred to as PSOI and shows very close acceptance ratio with Harmony Search based methods).

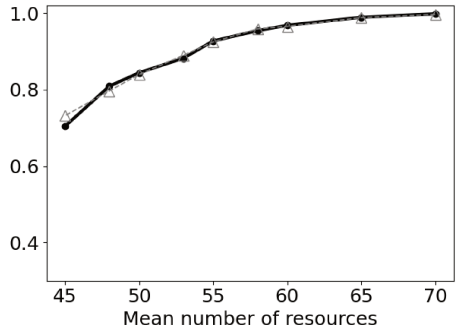
Since NEPA tries to maximize a reward function, and the AFBD function is a cost function, which should be minimized, we take $\frac{1}{AFBD(\mathcal{G},\mathcal{H}^x)}$ as the reward function in this appendix. We run NEPA on all the cases of section 5.2 (synthetic networks) and section 5.3 (real topologies), but only depict a subset of those instances due for brevity. Our results (figs. B.1 B.2 B.3) show there is no significative difference when using AFBD with NEPA.



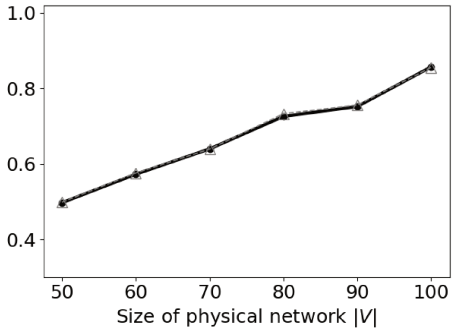
B.2.1 Acceptances for varying arrival rate (λ)



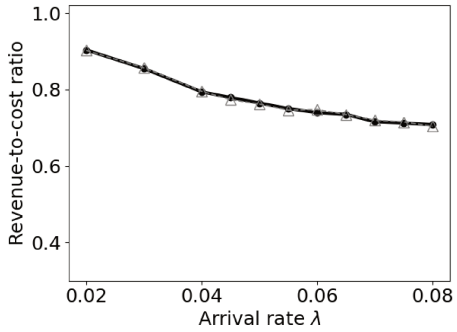
B.2.2 Acceptances for varying CNS size



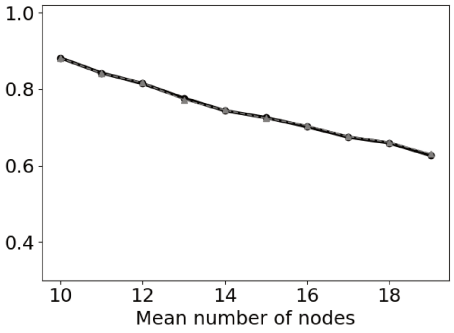
B.2.3 Acceptances (varying CPU & BW capacities)



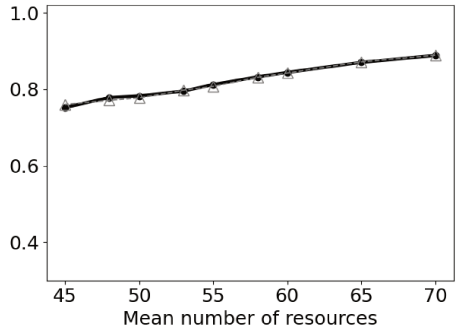
B.2.4 Mean acceptance ratios for varying physical network sizes



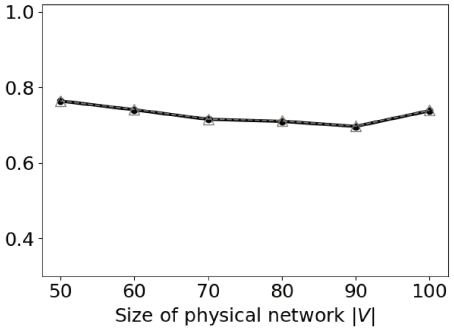
B.2.5 Revenue-to-cost ratio for varying λ



B.2.6 Revenue-to-cost ratio for varying CNS size



B.2.7 Revenue-to-cost ratio (varying CPU & BW capacities)



B.2.8 Revenue-to-cost ratio (varying network sizes)

Figure B.2: Results for sensitivity analysis experiments

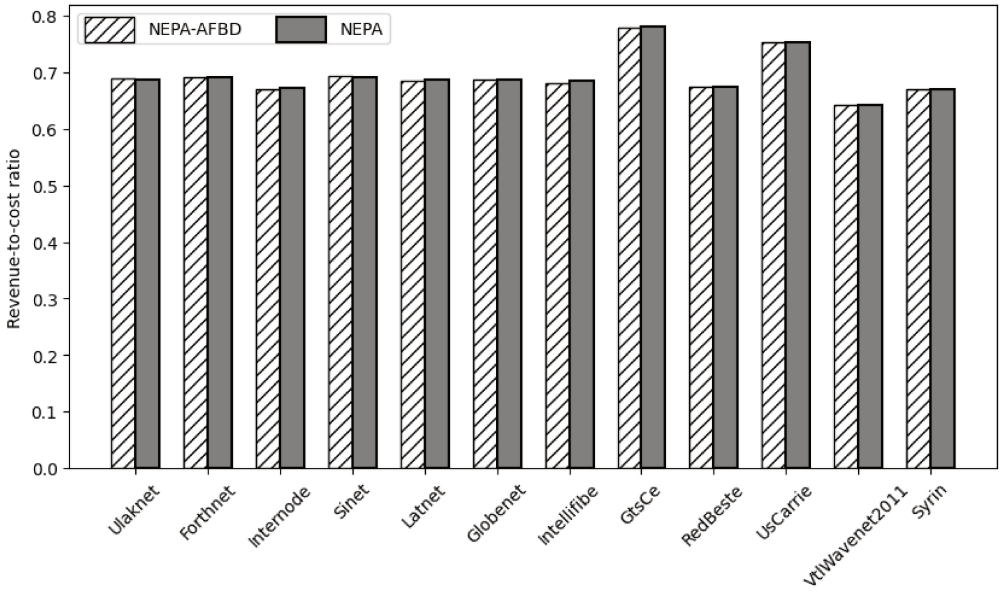


Figure B.3: Revenue-to-cost ratio for several real topologies

Appendix C

Ablation study of NEPA components

In this appendix, we compare NEPA and NRPA with their counterpart that do not use weight initialization. The presented experiments were executed using the parameters and instance generation described in the main article. However, due to space constraints, we only present a representative subset (10 topologies) of the ablation study experiments for real networks. The rest can be found on [68]. We call NRPA-W and NEPA-W the versions of our algorithms that do not use our weight initialization function (but initialize all weights to 0)

Note that we chose parameters $l = 3$, $N = 7$ for NRPA and NRPA-W, as we observed that this resulted in the same runtime for NEPA/NEPA-W (with parameters $l = 3, N = 5$) and these approches.

Our results from figure C.2 show that NRPA outperforms standard NRPA-W by a thin margin on random topologies, hinting that the weight heuristic helps slightly for improving the results. For NEPA-W, there is no significative difference in results for randomly generated topologies compared to NEPA, meaning that the refining operation already exploits well the shortest path information. However, on real topologies (figure C.1), the difference is more significative. NEPA outperforms NEPA-W by a large acceptance margin on several topologies, such as Pern and UsCarrie. In terms of revenue-to-cost ratios (figure C.3, NEPA also outperforms NEPA-W on real topologies. It means on some real topologies, the extra exploitation of shortest path information is of importance. This makes sense intuitively as, as we have seen before in the paper, real topologies typically have larger diameters, mean distances and distance standard deviation, meaning an error of placement related to distances can be much more costly that on randomly generated topologies.

When we compare NRPA and NRPA-W on real topology, we also observe that on most of the cases, NRPA beats NRPA-W both in terms of

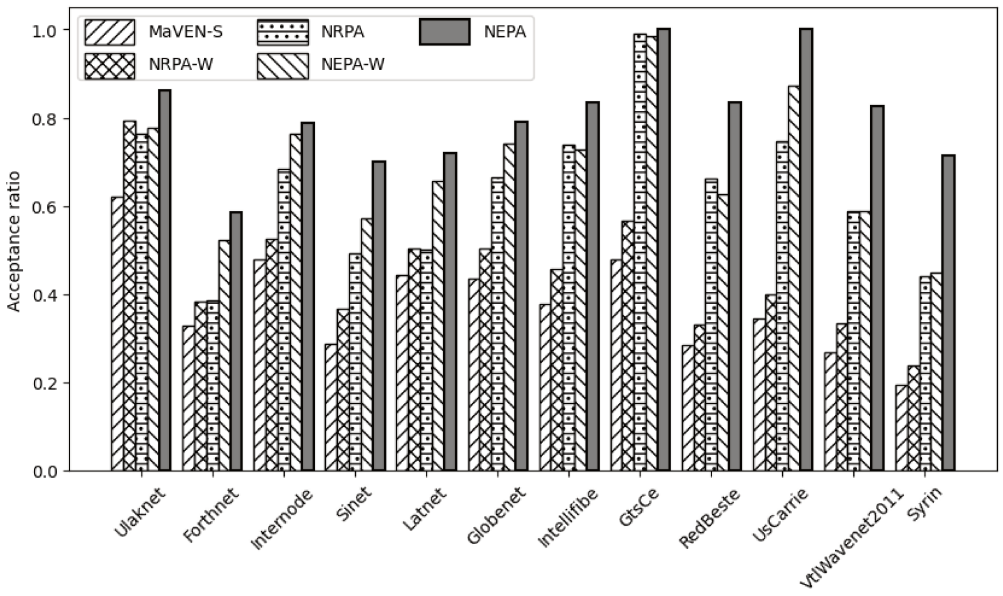
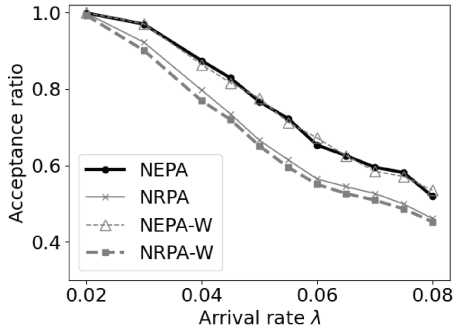


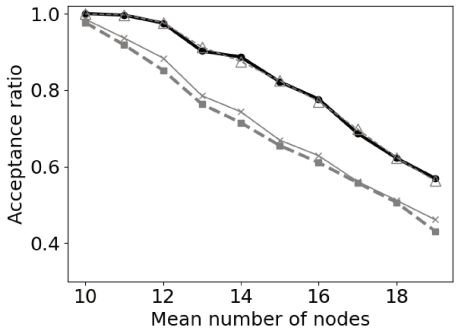
Figure C.1: Acceptance for several real topologies

acceptance and revenue-to-cost ratio by a larger margin than what was observed with random topologies. Similarly here, we believe this is largely due to less errors caused by choosing distant nodes when other less costly solutions existed.

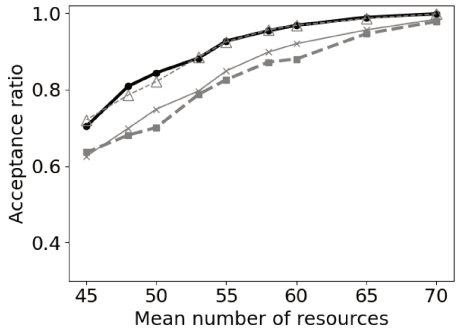
Finally, we observe that on all cases, NEPA significantly outperforms NRPA, meaning that neigborhood-based enhancement is effective at finding better solutions.



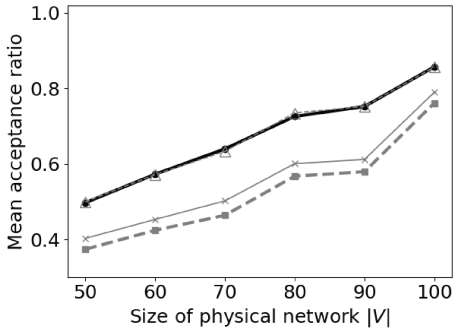
C.2.1 Acceptances for varying arrival rate (λ)



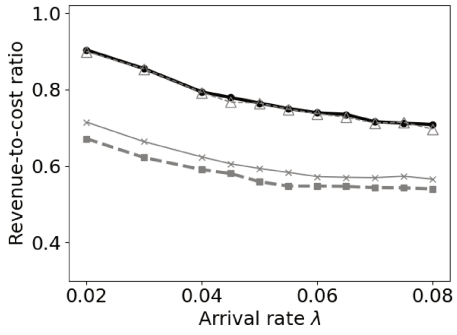
C.2.2 Acceptances for varying CNS size



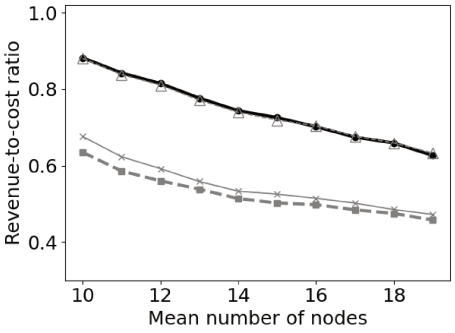
C.2.3 Acceptances (varying CPU & BW capacities)



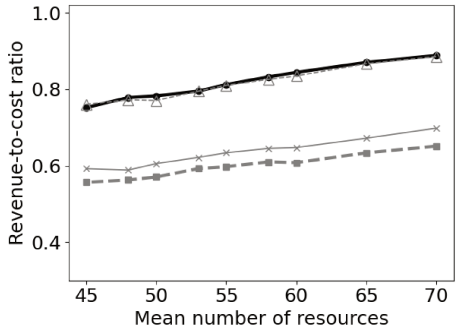
C.2.4 Mean acceptance ratios for varying physical network sizes



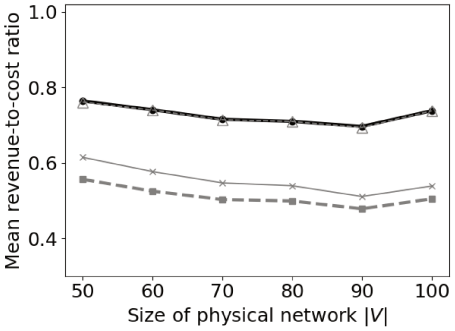
C.2.5 Revenue-to-cost ratio for varying λ



C.2.6 Revenue-to-cost ratio for varying CNS size



C.2.7 Revenue-to-cost ratio (varying CPU & BW capacities)



C.2.8 Revenue-to-cost ratio (varying network sizes)

Figure C.2: Results for sensitivity analysis experiments

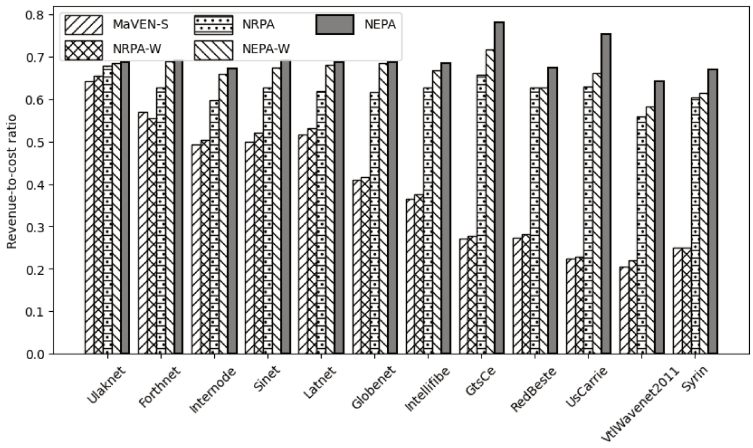


Figure C.3: Revenue-to-cost ratio for several real topologies

Appendix D

Statistics on network topologies

Instance	Mean distance	Diameter	Distance Standard Deviation	Clustering Coefficient
Intellifibe	6.33	15	2.99	0.088
Latnet	3.96	12	2.17	0.058
VtlWavenet2011	13.07	31	6.50	0.000
Syrin	11.95	31	6.77	0.000
Globenet	5.23	15	2.46	0.081
Forthnet	3.28	7	1.06	0.016
GtsCe	9.83	21	4.18	0.082
Ulaknet	2.44	4	0.59	0.000
Sinet	3.93	7	1.41	0.000
Internode	3.60	6	1.14	0.013
UsCarrie	12.09	35	6.46	0.058
RedBeste	10.59	28	5.66	0.009
Missouri	6.23	14	2.73	0.025
Interoute	7.62	17	3.39	0.105
Columbus	7.24	18	3.62	0.045
Garr201201	3.62	8	1.28	0.054
Cogentco	10.51	28	5.10	0.012
Deltaco	7.16	23	3.80	0.107
AsnetA	3.78	8	1.43	0.127
Switc	6.09	13	2.80	0.110
Uninett2011	4.25	9	1.62	0.018
Ion	10.14	25	4.79	0.011
Pern	4.55	8	1.89	0.004
Esnet	4.32	9	1.68	0.034
Colt	9.35	20	3.70	0.040
TataNld	9.85	28	5.17	0.065

Table D.1: Statistics on each real topology

Instance	Mean Distance	Diameter	Distance standard deviation	Clustering coefficient
Waxman 50	2.58	6	0.88	0.169
Waxman 60	2.45	4	0.76	0.108
Waxman 70	2.50	5	0.77	0.152
Waxman 80	2.41	5	0.73	0.142
Waxman 90	2.44	4	0.70	0.127
Waxman 100	2.27	4	0.66	0.160
Erdos-Renyi 0.03	4.33	9	1.60	0.032
Erdos-Renyi 0.04	3.66	8	1.27	0.042
Erdos-Renyi 0.06	3.27	7	1.09	0.019
Erdos-Renyi 0.08	2.74	6	0.87	0.054
Erdos-Renyi 0.11	2.34	4	0.68	0.109
Erdos-Renyi 0.16	2.01	3	0.55	0.142
Erdos-Renyi 0.2	1.89	3	0.49	0.182
PSS 1	1.93	4	0.62	0.385
PSS 2	1.82	4	0.61	0.476
PSS 3	1.91	4	0.59	0.376
PSS 4	1.94	4	0.59	0.358
PSS 5	1.83	4	0.56	0.425
PSS 6	1.98	4	0.62	0.413
PSS 7	1.92	4	0.60	0.398
PSS 8	1.91	4	0.58	0.360

Table D.2: Statistics on example simulated topologies

Bibliography

[1] *Study on physical layer enhancements for NR ultra-reliable and low latency case (URLLC).* 3GPP, 2019.

[2] GSMA. 5G Energy efficiency: Green is the new black. *https://www.zte.com.cn/global/about/news/202011241629.html.*

[3] Luís Carlos Gonçalves, Pedro Sebastião, Nuno Souto, and Américo Correia. One step greener: reducing 5g and beyond networks' carbon footprint by 2-tiering energy efficiency with co2 offsetting. *Electronics*, 9(3):464, 2020.

[4] Berge. *The theory of graphs.* Courier Corporation, 2001.

[5] Jeff Kennington and Mohamed Shalaby. An effective subgradient procedure for minimal cost multicommodity flow problems. *Management Science*, 23(9):994–1004, 1977.

[6] Andrew V Goldberg, Jeffrey D Oldham, Serge Plotkin, and Cliff Stein. An implementation of a combinatorial approximation algorithm for minimum-cost multicommodity flow. In *International conference on integer programming and combinatorial optimization*, pages 338–352. Springer, 1998.

[7] Maxime Elkael, Hind Castel-Taleb, Badii Jouaber, Andrea Araldo, and Massinissa Ait Aba. Improved monte carlo tree search for virtual network embedding. In *2021 IEEE 46th Conference on Local Computer Networks (LCN)*, pages 605–612. IEEE, 2021.

[8] Maxime Elkael, Massinissa Ait Aba, Andrea Araldo, Hind Castel-Taleb, and Badii Jouaber. Monkey business: Reinforcement learning meets neighborhood search for virtual network embedding. *Computer Networks*, 216:109204, 2022.

[9] Maxime Elkael, Andrea Aralda, Salvatore D'Oro, Hind Castel-Taleb, Massinissa Ait Aba, and Badii Jouaber. Joint placement, routing and dimensioning at the network edge for energy minimization. *Globecom*, 2023.

[10] Gabriele Gemmi, Maxime Elkael, Michele Polese, Leonardo Maccari, Hind Castel-Taleb, and Tommaso Melodia. Joint routing and energy optimization for integrated access and backhaul with open ran. *Globecom*, 2023.

[11] Eleonora Cau, Marius Corici, Paolo Bellavista, Luca Foschini, Giuseppe Carella, Andy Edmonds, and Thomas Michael Bohnert. Efficient exploitation of mobile edge computing for virtualized 5g in epc architectures. In *2016 4th IEEE international conference on mobile cloud computing, services, and engineering (MobileCloud)*, pages 100–109. IEEE, 2016.

[12] Ismael Gomez-Miguelez, Andres Garcia-Saavedra, Paul D Sutton, Pablo Serrano, Cristina Cano, and Doug J Leith. srslte: An open-source platform for lte evolution and experimentation. In *Proceedings of the Tenth ACM International Workshop on Wireless Network Testbeds, Experimental Evaluation, and Characterization*, pages 25–32, 2016.

[13] Navid Nikaein, Mahesh K Marina, Saravana Manickam, Alex Dawson, Raymond Knopp, and Christian Bonnet. Openairinterface: A flexible platform for 5g research. *ACM SIGCOMM Computer Communication Review*, 44(5):33–38, 2014.

[14] 3GPP. Ts 23.501 v15.0.0 (2017-12) system architecture for the 5g system (stage 2).

[15] Techplayon. https://www.techplayon.com/open-ran-o-ran-reference-architecture/.

[16] Wenfeng Xia, Yonggang Wen, Chuan Heng Foh, Dusit Niyato, and Haiyong Xie. A survey on software-defined networking. *IEEE Communications Surveys & Tutorials*, 17(1):27–51, 2014.

[17] John T Moy. *OSPF: anatomy of an Internet routing protocol*. Addison-Wesley Professional, 1998.

[18] Vishal Sharma, Rajneesh Narula, and Sumeer Khullar. Performance analysis of ieee 802.3 using igrp and eigrp routing protocols. *International Journal of Computer Applications*, 975:8887, 2012.

[19] Adrian Lara, Anisha Kolasani, and Byrav Ramamurthy. Network innovation using openflow: A survey. *IEEE communications surveys & tutorials*, 16(1):493–512, 2013.

[20] Gigabyte. https://www.gigabyte.com/solutions/embb.

[21] Ning Zhang, Peng Yang, Shan Zhang, Dajiang Chen, Weihua Zhuang, Ben Liang, and Xuemin Sherman Shen. Software defined networking enabled wireless network virtualization: Challenges and solutions. *IEEE network*, 31(5):42–49, 2017.

[22] Brendan Burns, Joe Beda, Kelsey Hightower, and Lachlan Evenson. *Kubernetes: up and running.* " O'Reilly Media, Inc.", 2022.

[23] Leonardo Bonati, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. Scope: An open and softwarized prototyping platform for nextg systems. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services*, pages 415–426, 2021.

[24] Daniela Raddino. https://www.linkedin.com/pulse/what-5g-integrated-access-backhaul-iab-daniela-raddino/.

[25] Simon Knight, Hung X Nguyen, Nickolas Falkner, Rhys Bowden, and Matthew Roughan. The internet topology zoo. *IEEE Journal on Selected Areas in Communications*, 29(9):1765–1775, 2011.

[26] Rhys Bowden, Hung X Nguyen, Nickolas Falkner, Simon Knight, and Matthew Roughan. Planarity of data networks. In *2011 23rd International Teletraffic Congress (ITC)*, pages 254–261. IEEE, 2011.

[27] Monika R Henzinger, Philip Klein, Satish Rao, and Sairam Subramanian. Faster shortest-path algorithms for planar graphs. *journal of computer and system sciences*, 55(1):3–23, 1997.

[28] Neil Robertson, Daniel P Sanders, Paul Seymour, and Robin Thomas. Efficiently four-coloring planar graphs. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*, pages 571–575, 1996.

[29] Haruko Okamura and Paul D Seymour. Multicommodity flows in planar graphs. *Journal of Combinatorial Theory, Series B*, 31(1): 75–81, 1981.

[30] Omar Houidi. *Algorithms for virtual network functions chaining.* PhD thesis, Institut polytechnique de Paris, 2020.

[31] Mosharaf Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Transactions on networking*, 20(1):206–219, 2011.

[32] Xiang Cheng, Sen Su, Zhongbao Zhang, Hanchi Wang, Fangchun Yang, Yan Luo, and Jie Wang. Virtual network embedding through topology-aware node ranking. *ACM SIGCOMM Computer Communication Review*, 41(2):38–47, 2011.

[33] Andreas Fischer, Juan Felipe Botero, Michael Till Beck, Hermann De Meer, and Xavier Hesselbach. Virtual network embedding: A survey. *IEEE Communications Surveys & Tutorials*, 15(4):1888–1906, 2013.

[34] Matthias Rost and Stefan Schmid. Np-completeness and inapproximability of the virtual network embedding problem and its variants. *arXiv preprint arXiv:1801.03162*, 2018.

[35] Matthias Rost and Stefan Schmid. On the hardness and inapproximability of virtual network embeddings. *IEEE/ACM Transactions on Networking*, 28(2):791–803, 2020.

[36] Soroush Haeri and Ljiljana Trajković. Virtual network embedding via monte carlo tree search. *IEEE transactions on cybernetics*, 48(2): 510–521, 2017.

[37] Zhongxia Yan, Jingguo Ge, Yulei Wu, Liangxiong Li, and Tong Li. Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks. *IEEE Journal on Selected Areas in Communications*, 38(6):1040–1057, 2020.

[38] Mahdi Dolati, Seyedeh Bahereh Hassanpour, Majid Ghaderi, and Ahmad Khonsari. Deepvine: Virtual network embedding with deep reinforcement learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 879–885. IEEE, 2019.

[39] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[40] Zhongbao Zhang, Xiang Cheng, Sen Su, Yiwen Wang, Kai Shuang, and Yan Luo. A unified enhanced particle swarm optimization-based virtual network embedding algorithm. *International Journal of Communication Systems*, 26(8):1054–1073, 2013.

[41] Tristan Cazenave and Fabien Teytaud. Application of the nested rollout policy adaptation algorithm to the traveling salesman problem

with time windows. In *International Conference on Learning and Intelligent Optimization*, pages 42–54. Springer, 2012.

[42] Tristan Cazenave, Jean-Yves Lucas, Hyoseok Kim, and Thomas Triboulet. Monte carlo vehicle routing. In *ATT at ECAI 2020*, 2020.

[43] Christian Aguilar-Fuster and Javier Rubio-Loyola. A novel evaluation function for higher acceptance rates and more profitable metaheuristic-based online virtual network embedding. *Computer Networks*, 195:108191, 2021.

[44] Marcio Melo, Susana Sargento, Ulrich Killat, Andreas Timm-Giel, and Jorge Carapinha. Optimal virtual network embedding: Node-link formulation. *IEEE Transactions on Network and Service Management*, 10(4):356–368, 2013.

[45] Massinissa Ait Aba, Maxime Elkael, Badii Jouaber, Hind Castel-Taleb, Andrea Araldo, and David Olivier. A two-stage algorithm for the Virtual Network Embedding problem. In *LCN 2021: 46th conference on Local Computer Networks*, pages 395–398, Edmonton (online), Canada, October 2021. IEEE. doi: 10.1109/LCN52139.2021.9524968. URL https://hal.archives-ouvertes.fr/hal-03524809.

[46] Farzad Habibi, Mahdi Dolati, Ahmad Khonsari, and Majid Ghaderi. Accelerating virtual network embedding with graph neural networks. In *2020 16th International Conference on Network and Service Management (CNSM)*, pages 1–9. IEEE, 2020.

[47] Andreas Blenk, Patrick Kalmbach, Johannes Zerwas, Michael Jarschel, Stefan Schmid, and Wolfgang Kellerer. Neurovine: A neural preprocessor for your virtual network embedding algorithm. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 405–413. IEEE, 2018.

[48] Xiuming Mi, Xiaolin Chang, Jiqiang Liu, Longmei Sun, and Bin Xing. Embedding virtual infrastructure based on genetic algorithm. In *2012 13th International Conference on Parallel and Distributed Computing, Applications and Technologies*, pages 239–244. IEEE, 2012.

[49] Ilhem Fajjari, Nadjib Aitsaadi, Guy Pujolle, and Hubert Zimmermann. Vne-ac: Virtual network embedding algorithm based on ant colony metaheuristic. In *2011 IEEE international conference on communications (ICC)*, pages 1–6. IEEE, 2011.

[50] Xiang Cheng, Sen Su, Zhongbao Zhang, Kai Shuang, Fangchun Yang, Yan Luo, and Jie Wang. Virtual network embedding through topology awareness and optimization. *Computer Networks*, 56(6):1797–1813, 2012.

[51] Ashraf A Shahin. Memetic multi-objective particle swarm optimization-based energy-aware virtual network embedding. *arXiv preprint arXiv:1504.06855*, 2015.

[52] Long Gong, Yonggang Wen, Zuqing Zhu, and Tony Lee. Toward profit-seeking virtual network embedding algorithm via global resource capacity. In *IEEE INFOCOM 2014-IEEE Conference on Computer Communications*, pages 1–9. IEEE, 2014.

[53] Jiuyue Cao, Yan Zhang, Wei An, Xin Chen, Jiyan Sun, and Yanni Han. Vnf-fg design and vnf placement for 5g mobile networks. *Science China Information Sciences*, 60:1–15, 2017.

[54] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[55] Tristan Cazenave, Jean-Baptiste Sevestre, and Matthieu Toulemont. Stabilized nested rollout policy adaptation. In *Monte Carlo Search: First Workshop, MCS 2020, Held in Conjunction with IJCAI 2020, Virtual Event, January 7, 2021, Proceedings 1*, pages 17–30. Springer, 2021.

[56] Christopher D Rosin. Nested rollout policy adaptation for monte carlo tree search. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011.

[57] Matthew Andrews and Lisa Zhang. Hardness of the undirected edge-disjoint paths problem. In *Proceedings of the thirty-seventh annual ACM symposium on Theory of computing*, pages 276–283, 2005.

[58] Minlan Yu, Yung Yi, Jennifer Rexford, and Mung Chiang. Rethinking virtual network embedding: Substrate support for path splitting and migration. *ACM SIGCOMM Computer Communication Review*, 38 (2):17–29, 2008.

[59] Saman Ahmadi, Guido Tack, Daniel Harabor, and Philip Kilby. Enhanced methods for the weight constrained shortest path problem: Constrained path finding meets bi-objective search. *arXiv preprint arXiv:2207.14744*, 2022.

[60] Johnson and Garey. Computers and intractability: a guide to the theory of np-completeness. 1980.

[61] Jiawei Zhang, Yuefeng Ji, Mei Song, Hui Li, Rentao Gu, Yongli Zhao, and Jie Zhang. Dynamic virtual network embedding over multilayer optical networks. *Journal of Optical Communications and Networking*, 7(9):918–927, 2015.

[62] Bernard Fortz, Luís Gouveia, and Martim Joyce-Moniz. Models for the piecewise linear unsplittable multicommodity flow problems. *European Journal of Operational Research*, 261(1):30–42, 2017.

[63] Fatemeh Hosseini, Alexander James, and Majid Ghaderi. Probabilistic virtual link embedding under demand uncertainty. *IEEE Transactions on Network and Service Management*, 16(4):1552–1566, 2019.

[64] Walid Ben-Ameur and Adam Ouorou. Mathematical models of the delay constrained routing problem. *Algorithmic Operations Research*, 1(2):94–103, 2006.

[65] Muntasir Raihan Rahman, Issam Aib, and Raouf Boutaba. Survivable virtual network embedding. In *International Conference on Research in Networking*, pages 40–52. Springer, 2010.

[66] Andreas Fischer. *An evaluation methodology for virtual network embeddING*. PhD thesis, Universität Passau, 2017.

[67] Ieee standard for ethernet - amendment 3: Media access control parameters for 50 gb/s and physical layers and management parameters for 50 gb/s, 100 gb/s, and 200 gb/s operation. *IEEE Std 802.3cd-2018 (Amendment to IEEE Std 802.3-2018 as amended by IEEE Std 802.3cb-2018 and IEEE Std 802.3bt-2018)*, pages 1–401, 2019. doi: 10.1109/IEEESTD.2019.8649797.

[68] Maxime Elkael. vne.jl. `https://github.com/melkael/vne`, 2021.

[69] Jose Jurandir Alves Esteves, Amina Boubendir, Fabrice Guillemin, and Pierre Sens. Heuristic for edge-enabled network slicing optimization using the "power of two choices". In *2020 16th International Conference on Network and Service Management (CNSM)*, pages 1–9. IEEE, 2020.

[70] Juan Felipe Botero, Xavier Hesselbach, Michael Duelli, Daniel Schlosser, Andreas Fischer, and Hermann De Meer. Energy efficient virtual network embedding. *IEEE Communications Letters*, 16(5): 756–759, 2012.

[71] Li and Wang. An energy-aware edge server placement algorithm in mobile edge computing. In *EDGE 2018*, pages 66–73. IEEE, 2018.

[72] Cynthia Barnhart, Christopher A Hane, and Pamela H Vance. Using branch-and-price-and-cut to solve origin-destination integer multicommodity flow problems. *Operations Research*, 48(2):318–326, 2000.

[73] Christophe Duhamel and Philippe Mahey. Multicommodity flow problems with a bounded number of paths: A flow deviation approach. *Networks: An International Journal*, 49(1):80–89, 2007.

[74] Wei Huang, Andrea Araldo, Hind Castel-Taleb, and Badii Jouaber. Dimensioning resources of network slices for energy-performance trade-off. In *2022 IEEE Symposium on Computers and Communications (ISCC)*, pages 1–6. IEEE, 2022.

[75] Nafiseh Sharghivand, Farnaz Derakhshan, Lena Mashayekhy, and Leyli Mohammadkhanli. An edge computing matching framework with guaranteed quality of service. *IEEE Transactions on Cloud Computing*, 10(3):1557–1570, 2020.

[76] Rahul Yadav, Weizhe Zhang, Omprakash Kaiwartya, Houbing Song, and Shui Yu. Energy-latency tradeoff for dynamic computation offloading in vehicular fog computing. *IEEE Transactions on Vehicular Technology*, 69(12):14198–14211, 2020.

[77] Hossein Badri, Tayebeh Bahreini, Daniel Grosu, and Kai Yang. Energy-aware application placement in mobile edge computing: A stochastic optimization approach. *IEEE Transactions on Parallel and Distributed Systems*, 31(4):909–922, 2019.

[78] Yongchao Zhang, Xin Chen, Ying Chen, Zhuo Li, and Jiwei Huang. Cost efficient scheduling for delay-sensitive tasks in edge computing system. In *2018 IEEE International Conference on Services Computing (SCC)*, pages 73–80. IEEE, 2018.

[79] Pengchao Han, Lei Guo, Yejun Liu, Xuetao Wei, Jian Hou, and Xu Han. A new virtual network embedding framework based on qos satisfaction and network reconfiguration for fiber-wireless access network. In *2016 IEEE International Conference on Communications (ICC)*, pages 1–7. IEEE, 2016.

[80] Weiwei Fang, Zishuo Wang, Jaime LIoret, Daqiang Zhang, and Zhijun Yang. Optimising data placement and traffic routing for energy saving in backbone networks. *Transactions on Emerging Telecommunications Technologies*, 25(9):914–925, 2014.

[81] Ravindra K Ahuja, Thomas L Magnanti, and James B Orlin. Network flows. 1988.

[82] Bernard Fortz and Mikkel Thorup. Internet traffic engineering by optimizing ospf weights. In *Proceedings IEEE INFOCOM 2000. conference on computer communications. Nineteenth annual joint conference of the IEEE computer and communications societies (Cat. No. 00CH37064)*, volume 2, pages 519–528. IEEE, 2000.

[83] Chen Dang, Cristina Bazgan, Tristan Cazenave, Morgan Chopin, and Pierre-Henri Wuillemin. Monte carlo search algorithms for network traffic engineering. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 486–501. Springer, 2021.

[84] Prabhakar Raghavan and Clark D Tompson. Randomized rounding: a technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, 1987.

[85] Farah Ait Salaht, Frédéric Desprez, and Adrien Lebre. An overview of service placement problem in fog and edge computing. *ACM Computing Surveys (CSUR)*, 53(3):1–35, 2020.

[86] George Dantzig and Thapa TN. *Linear programming: Theory and extensions*. 2003.

[87] Eunjeong Choi and Dong-Wan Tcha. A column generation approach to the heterogeneous fleet vehicle routing problem. *Computers & Operations Research*, 34(7):2080–2095, 2007.

[88] Carlos Saldarriaga-Cortés, Harold Salazar, Rodrigo Moreno, and Guillermo Jiménez-Estévez. Stochastic planning of electricity and gas networks: An asynchronous column generation approach. *Applied energy*, 233:1065–1077, 2019.

[89] Patrik Bjorklund, Peter Varbrand, and Di Yuan. Resource optimization of spatial tdma in ad hoc radio networks: A column generation approach. In *IEEE INFOCOM 2003. Twenty-second Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE Cat. No. 03CH37428)*, volume 2, pages 818–824. IEEE, 2003.

[90] Stephen Boyd and Jacob Mattingley. Branch and bound methods. *Notes for EE364b, Stanford University*, 2006:07, 2007.

[91] Edwin KP Chong, Wu-Sheng Lu, and Stanislaw H Żak. *An introduction to optimization*. John Wiley and Sons, 2023.

[92] Donald B Johnson. A note on dijkstra's shortest path algorithm. *Journal of the ACM (JACM)*, 20(3):385–388, 1973.

[93] Thomas H Cormen, Charles E Leiserson, Ronald L Rivest, and Clifford Stein. *Introduction to algorithms*. MIT press, 2022.

[94] John E Beasley and Nicos Christofides. An algorithm for the resource constrained shortest path problem. *Networks*, 19(4):379–394, 1989.

[95] Luigi Di Puglia Pugliese and Francesca Guerriero. A survey of resource constrained shortest path problems: Exact solution approaches. *Networks*, 62(3):183–200, 2013.

[96] Raith and Ehrgott. A comparison of solution strategies for biobjective shortest path problems. *Computers & Operations Research*, 36(4): 1299–1331, 2009.

[97] Daniel Duque, Leonardo Lozano, and Andrés L Medaglia. An exact method for the biobjective shortest path problem for large-scale road networks. *European Journal of Operational Research*, 242(3):788–797, 2015.

[98] Daniel Villeneuve and Guy Desaulniers. The shortest path problem with forbidden paths. *European Journal of Operational Research*, 165 (1):97–107, 2005.

[99] Luigi Di Puglia Pugliese and Francesca Guerriero. Shortest path problem with forbidden paths: The elementary version. *European Journal of Operational Research*, 227(2):254–267, 2013.

[100] Dominique Feillet, Pierre Dejax, Michel Gendreau, and Cyrille Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks: An International Journal*, 44(3):216–229, 2004.

[101] Jacques Desrosiers and Marco E Lübbecke. A primer in column generation. In *Column generation*, pages 1–32. Springer, 2005.

[102] Mustaq Ahmed and Anna Lubiw. Shortest paths avoiding forbidden subpaths. *arXiv preprint arXiv:0807.0807*, 2008.

[103] Kevin Limonier, Frédérick Douzet, Louis Pétiniaud, Loqman Salamatian, and Kave Salamatian. Mapping the routes of the internet for geopolitics: The case of eastern ukraine. *First Monday*, 2021.

[104] Pierre Hansen. Bicriterion path problems. In *Multiple Criteria Decision Making Theory and Application: Proceedings of the Third Conference Hagen/Königswinter, West Germany, August 20–24, 1979*, pages 109–127. Springer, 1980.

[105] Joao Carlos Namorado Climaco and Ernesto Queiros Vieira Martins. A bicriterion shortest path algorithm. *European Journal of Operational Research*, 11(4):399–404, 1982.

[106] Pedro Maristany de las Casas, Antonio Sedeno-Noda, and Ralf Borndörfer. An improved multiobjective shortest path algorithm. *Computers & Operations Research*, 135:105424, 2021.

[107] Sofie Demeyer, Jan Goedgebeur, Pieter Audenaert, Mario Pickavet, and Piet Demeester. Speeding up martins' algorithm for multiple objective shortest path problems. *4or*, 11:323–348, 2013.

[108] James Brumbaugh-Smith and Donald Shier. An empirical investigation of some bicriterion shortest path algorithms. *European Journal of Operational Research*, 43(2):216–224, 1989.

[109] Lawrence Mandow, JL Pérez De la Cruz, et al. A new approach to multiobjective a* search. In *IJCAI*, volume 8, 2005.

[110] Saman Ahmadi, Guido Tack, Daniel D Harabor, and Philip Kilby. A fast exact algorithm for the resource constrained shortest path problem. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 12217–12224, 2021.

[111] Leonardo Lozano and Andrés L Medaglia. On an exact method for the constrained shortest path problem. *Computers & operations research*, 40(1):378–384, 2013.

[112] Ernesto de Queiros Vieira Martins. An algorithm for ranking paths that may contain cycles. *European Journal of Operational Research*, 18(1):123–130, 1984.

[113] Alfred V Aho and Margaret J Corasick. Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, 18(6): 333–340, 1975.

[114] Esfahbod Behnam. https://commons.wikimedia.org/w/index.php?curid=3532181.

[115] Stephen A Cook. The complexity of theorem-proving procedures. In *Logic, Automata, and Computational Complexity: The Works of Stephen A. Cook*, pages 143–152. 2023.

[116] Richard M Karp. *Reducibility among combinatorial problems*. Springer, 2010.

[117] Anders JV Skriver and Kim Allan Andersen. A label correcting approach for solving bicriterion shortest-path problems. *Computers & Operations Research*, 27(6):507–524, 2000.

[118] Mosharaf Chowdhury, Muntasir Raihan Rahman, and Raouf Boutaba. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. *IEEE/ACM Transactions on networking*, 20(1):206–219, 2011.

[119] Arnaud Adelin, Philippe Owezarski, and Thierry Gayraud. On the impact of monitoring router energy consumption for greening the internet. In *2010 11th IEEE/ACM International Conference on Grid Computing*, pages 298–304. IEEE, 2010.

[120] Vijay Sivaraman, Arun Vishwanath, Zhi Zhao, and Craig Russell. Profiling per-packet and per-byte energy consumption in the netfpga gigabit router. In *2011 IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 331–336. IEEE, 2011.

[121] Guohan Lu, Chuanxiong Guo, Yulong Li, Zhiqiang Zhou, Tong Yuan, Haitao Wu, Yongqiang Xiong, Rui Gao, and Yongguang Zhang. {ServerSwitch}: A programmable and high performance platform for data center networks. In *8th USENIX Symposium on Networked Systems Design and Implementation (NSDI 11)*, 2011.

[122] Xiaobo Fan, Wolf-Dietrich Weber, and Luiz Andre Barroso. Power provisioning for a warehouse-sized computer. *ACM SIGARCH computer architecture news*, 35(2):13–23, 2007.

[123] Marcelo Amaral, Huamin Chen, Tatsuhiro Chiba, Rina Nakazawa, Sunyanan Choochotkaew, Eun Kyung Lee, and Tamar Eilam. Kepler: A framework to calculate the energy consumption of containerized applications. In *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*, pages 69–71. IEEE, 2023.

[124] Asmussen. *Applied probability and queues*, volume 2. Springer, 2003.

[125] Maxime Elkael, Andrea Araldo, Hind Castel-Taleb, and Badii Jouaber. An exact algorithm to solve multi-objective, multi-constrained shortest path problems with forbidden paths. *Multi-Constrained Shortest Path Problems with Forbidden Paths (April 10, 2023)*, 2023.

[126] https://github.com/melkael/mcf-energy.

[127] Boyd and Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

[128] Richard L Burden, J Douglas Faires, and Annette M Burden. *Numerical analysis*. Cengage learning, 2015.

[129] Narahari Umanath Prabhu. *Stochastic storage processes: queues, insurance risk, and dams, and data communication.* Number 15. Springer Science & Business Media, 1998.

[130] Grimmett and Stirzaker. *Probability and random processes.* Oxford university press, 2020.

[131] Shimon Even, Alon Itai, and Adi Shamir. On the complexity of time table and multi-commodity flow problems. In *16th annual symposium on foundations of computer science (sfcs 1975)*, pages 184–193. IEEE, 1975.

[132] David López-Pérez, Ming Ding, Holger Claussen, and Amir H. Jafari. Towards 1 Gbps/UE in Cellular Systems: Understanding Ultra-Dense Small Cell Deployments. *IEEE Communications Surveys & Tutorials*, (4):2078–2101, 2015. doi: 10.1109/COMST.2015.2439636.

[133] Theodore S Rappaport, James N Murdock, Felix Gutierrez, Erez Ben-Dor, Jonathan Tamir, Yu Qiao, Osman Gupta, Andrew R Nix, and Morteza Samimi. Millimeter wave mobile communications for 5G cellular: It will work! *IEEE Access*, pages 335–349, 2013.

[134] Mustafa Reza Akdeniz, Yan Liu, Morteza Samimi, Shu Sun, Sundeep Rangan, Taeyoung Yang, Elza Erkip, Ahmed Sulyman, Ali Sadri, Stephanie Peters, et al. Millimeter wave channel modeling and cellular capacity evaluation. *IEEE Journal on Selected Areas in Communications*, (6):1164–1179, 2014.

[135] Michele Polese, Marco Giordani, Tommaso Zugno, Arnab Roy, Sanjay Goyal, Douglas Castor, and Michele Zorzi. Integrated access and backhaul in 5G mmWave networks: Potential and challenges. *IEEE Communications Magazine*, (3):62–68, 2020.

[136] GSMA. 5G Energy efficiencies: Green is the new black, Nov 2020.

[137] Michele Polese, Leonardo Bonati, Salvatore D'Oro, Stefano Basagni, and Tommaso Melodia. Understanding O-RAN: Architecture, interfaces, algorithms, security, and research challenges. *IEEE Communications Surveys & Tutorials*, 2023.

[138] Salvatore D'Oro, Leonardo Bonati, Michele Polese, and Tommaso Melodia. Orchestran: Network automation through orchestrated intelligence in the open ran. In *IEEE INFOCOM 2022-IEEE Conference on Computer Communications*, pages 270–279. IEEE, 2022.

[139] Eugenio Moro, Gabriele Gemmi, Michele Polese, Leonardo Maccari, Antonio Capone, and Tommaso Melodia. Toward Open Integrated Access and Backhaul
with O-RAN. In *21st Mediterranean Communication and Computer Networking Conference (MedComNet 2023)*. IEEE, 2023.

[140] Gianni Barlacchi, Marco De Nadai, Roberto Larcher, Antonio Casella, Cristiana Chitic, Giovanni Torrisi, Fabrizio Antonelli, Alessandro Vespignani, Alex Pentland, and Bruno Lepri. A multi-source dataset of urban life in the city of Milan and the Province of Trentino. *Scientific data*, (1):1–15, 2015.

[141] Muhammad Nazmul Islam, Sundar Subramanian, and Ashwin Sampath. Integrated access backhaul in millimeter wave networks. In *2017 IEEE Wireless Communications and Networking Conference (WCNC)*, pages 1–6, 2017. doi: 10.1109/WCNC.2017.7925837.

[142] Muhammad Nazmul Islam, Navid Abedini, Georg Hampel, Sundar Subramanian, and Junyi Li. Investigation of performance in integrated access and backhaul networks. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 597–602, 2018. doi: 10.1109/INFCOMW.2018.8406872.

[143] Danfeng Meng, Xiaohui Li, Wenjuan Pu, Xu Yang, and Dantao Li. An energy-saving scheme with multi-hop transmission for mmwave backhaul networks. In *2018 IEEE 87th Vehicular Technology Conference (VTC Spring)*, pages 1–5. IEEE, 2018.

[144] Lei Lei, Eva Lagunas, Symeon Chatzinotas, and Björn Ottersten. Noma aided interference management for full-duplex self-backhauling hetnets. *IEEE Communications Letters*, (8):1696–1699, 2018.

[145] Athul Prasad, Mikko A Uusitalo, and Andreas Maeder. Energy efficient coordinated self-backhauling for ultra-dense 5G networks. In *2017 IEEE 85th Vehicular Technology Conference (VTC Spring)*, pages 1–5. IEEE, 2017.

[146] Cauligi S Raghavendra, Krishna M Sivalingam, and Taieb Znati. *Wireless sensor networks*. Springer, 2006.

[147] M Ezhilarasi and V Krishnaveni. A survey on wireless sensor network: energy and lifetime perspective. *Taga Journal*, 14(6):41–48, 2018.

[148] Tifenn Rault, Abdelmadjid Bouabdallah, and Yacine Challal. Energy efficiency in wireless sensor networks: A top-down survey. *Computer networks*, 67:104–122, 2014.

[149] Nicola Piovesan, David López-Pérez, Antonio De Domenico, Xinli Geng, Harvey Bao, and Mérouane Debbah. Machine learning and analytical power consumption models for 5G Base Stations. *IEEE Communications Magazine*, (10):56–62, 2022.

[150] 3GPP. Study on integrated access and backhaul. Technical Report (TR) 38.874, 3rd Generation Partnership Project (3GPP), 01 2019. Version 16.0.0.

[151] Smallcell Forum. https://www.smallcellforum.org/darts-tool/.

[152] Gabriele Gemmi, Renato Lo Cigno, and Leonardo Maccari. On cost-effective, reliable coverage for los communications in urban areas. *IEEE Transactions on Network and Service Management*, 2022. doi: 10.1109/TNSM.2022.3190634.

[153] 3GPP. Study on Scenarios and Requirements for Next Generation Access Technologies. Technical Report (TR) 38.913, 3rd Generation Partnership Project (3GPP), 04 2022. Version 17.0.0.

[154] 3GPP. Study on channel model for frequencies from 0.5 to 100 GHz. Technical Report (TR) 38.901, 3rd Generation Partnership Project (3GPP), 01 2020. Version 16.1.0.

[155] Gabriele Gemmi, Renato Lo Cigno, and Leonardo Maccari. On the properties of next generation wireless backhaul. *IEEE Transactions on Network Science and Engineering*, 2022.

[156] Andrea Baiocchi, Luca Chiaraviglio, Francesca Cuomo, and Valentina Salvatore. Joint management of energy consumption, maintenance costs, and user revenues in cellular networks with sleep modes. *IEEE Transactions on Green Communications and Networking*, (2):167–181, 2017. doi: 10.1109/TGCN.2017.2686598.

[157] Gabriele Gemmi, Maxime Elkael, Michele Polese, Leonardo Maccari, and Tommaso Melodia. Joint routing and energy optimization for integrated access and backhaul networks.

[158] Bruce A Reed. Algorithmic aspects of tree width. *Recent advances in algorithms and combinatorics*, pages 85–107, 2003.

[159] Mathias Rost. https://github.com/matthiasrost/topologyzoo-treewidth-analysis.

[160] Osama Arouk and Navid Nikaein. Kube5g: A cloud-native 5g service platform. In *GLOBECOM 2020-2020 IEEE Global Communications Conference*, pages 1–6. IEEE, 2020.

[161] Martin Stolle and Doina Precup. Learning options in reinforcement learning. In *Abstraction, Reformulation, and Approximation: 5th International Symposium, SARA 2002 Kananaskis, Alberta, Canada August 2–4, 2002 Proceedings 5*, pages 212–223. Springer, 2002.

[162] Maria-Florina Balcan, Travis Dick, Tuomas Sandholm, and Ellen Vitercik. Learning to branch. In *International conference on machine learning*, pages 344–353. PMLR, 2018.

[163] Brandon Amos and J Zico Kolter. Optnet: Differentiable optimization as a layer in neural networks. In *International Conference on Machine Learning*, pages 136–145. PMLR, 2017.

[164] Akshay Agrawal, Brandon Amos, Shane Barratt, Stephen Boyd, Steven Diamond, and J Zico Kolter. Differentiable convex optimization layers. *Advances in neural information processing systems*, 32, 2019.

[165] Gunther Auer, Vito Giannini, Claude Desset, Istvan Godor, Per Skillermark, Magnus Olsson, Muhammad Ali Imran, Dario Sabella, Manuel J Gonzalez, Oliver Blume, et al. How much energy is needed to run a wireless network? *IEEE wireless communications*, 18(5): 40–49, 2011.

[166] Huawei. Challenge dataset. *https://challenge.aiforgood.itu.int/match/matchitem/83*, 2023.

[167] Javier Rubio-Loyola, Christian Aguilar-Fuster, Gregorio Toscano-Pulido, Rashid Mijumbi, and Joan Serrat-Fernández. Enhancing metaheuristic-based online embedding in network virtualization environments. *IEEE Transactions on Network and Service Management*, 15(1):200–216, 2017.

Titre: Apprentissage par renforcement et optimisation pour un slicing 5G économe en ressources de calcul et en énergie

Résumé: Cette thèse traite des problèmes d'allocation des ressources dans les réseaux 5G. Notre objectif est d'exploiter le slicing du réseau (c'est-à-dire un corpus de techniques basées sur la virtualisation et la softwarisation du réseau qui permettent à l'opérateur de fournir différentes quantités de ressources à différents clients) afin d'améliorer l'efficacité énergétique et la consommation de ressources des réseaux 5G, tout en respectant des contraintes de Qualité de Service. Pour ce faire, nous formulons et résolvons des problèmes d'optimisation dans les différents domaines du réseau : nous nous intéressons tout d'abord au placement des slices dans le réseau coeur. Pour résoudre le problème, une nouvelle approche combinant la recherche Monte Carlo et la recherche par voisinage est formulée. Nous montrons qu'elle permet d'accepter plus de slices que les techniques de l'état de l'art pour le problème de placement du réseau coeur. Ensuite, nous mettons l'accent sur l'efficacité énergétique en proposant un framework pour l'allocation de ressources dans les réseaux 5G partagés entre les opérateurs de réseaux physiques (PNO) et les opérateurs de réseaux mobiles virtuels (MVNO). Ce framework tient compte à la fois du placement des composants logiciels, du routage des demandes des utilisateurs et du dimensionnement des ressources, tout en respectant les accords de niveau de service (SLA) basés sur des contraintes de latence et de fiabilité. Grâce à la génération de colonnes, nous obtenons des solutions exactes, démontrant des économies d'énergie allant jusqu'à 50% dans des réseaux réels, par rapport aux algorithmes de placement ou de minimisation des ressources existants. Enfin, nous abordons le problème de l'optimisation de l'énergie dans les réseaux Integrated Access and Backhaul (IAB), un élément clé des déploiements denses de la 5G. S'appuyant sur le framework du réseau d'accès ouvert Open RAN (O-RAN), notre modèle minimise les noeuds IAB actifs tout en garantissant une capacité minimale pour l'équipement de l'utilisateur (UE). Formulée comme un programme non linéaire binaire, cette approche réduit la consommation d'énergie du RAN de 47%, tout en maintenant la qualité de service pour les UEs. Dans l'ensemble, cette thèse propose de nouveaux algorithmes pour améliorer la l'efficacité en ressources et en énergie du réseau 5G slicé. Ces améliorations sont étudiées dans différentes parties du réseau, du coeur au réseau d'accès.

Title: Reinforcement Learning and optimization for an energy and resource efficient 5G slicing

Abstract: This thesis addresses resource allocation problems in 5G networks. Our objective is to leverage network slicing (e.g. the set of techniques based on virtualization and network softwarization which allows the network operator to provide different amounts of resources to different tenants) in order to to improve the energy-efficiency and resource consumption of 5G networks, while guaranteeing Quality of Service constraints. To do so, we formulate and solve optimization problems at the different domains of the network: We are first concerned with the placement of slices in the core network. To solve the problem, a new approach combining Monte Carlo Search and Neighborhood Search is formulated. We show it accepts more core slices than state-of-the-art approaches for the core network placement problem. Then we shift the focus to energy efficiency in resource allocation in 5G networks shared between Physical Network Operators (PNOs) and Mobile Virtual Network Operators (MVNOs). This framework jointly considers software component placement, user request routing, and resource dimensioning while meeting Service Level Agreements (SLAs) based on latency and reliability constraints. Through Column Generation, we obtain exact solutions, demonstrating energy savings of up to 50% in real networks compared to existing placement or resource minimization algorithms. Finally, we delve into the realm of energy optimization in Integrated Access and Backhaul (IAB) networks, a key component of dense 5G deployments. Leveraging the Open Radio Access Network (O-RAN) framework, our model minimizes active IAB nodes while ensuring a minimum capacity for User Equipment (UE). Formulated as a binary nonlinear program, this approach reduces RAN energy consumption by 47%, while maintaining Quality-Of-Service for UEs. Overall, this thesis provides novel algorithms for improving resource and energy efficiency of 5G network slicing. Such improvement is studied in different parts of the network, from the core up to the access network.