Optimization of Network Slice Placement in Distributed Large Scale Infrastructures From Heuristics to Controlled Deep Reinforcement Learning

Jose Jurandir Alves Esteves





SORBONNE UNIVERSITÉ

ORANGE

Doctoral School **Ecole Doctorale Informatique, Télécommunications et Electronique**

University Department **Laboratoire d'Informatique de Paris 6**

Thesis defended by **ALVES ESTEVES JOSÉ JURANDIR**

Defended on **December 13, 2021**

In order to become Doctor from Sorbonne Université

Speciality **Computer Science**

Optimization of Network Slice Placement in Distributed Large Scale Infrastructures

From Heuristics to Controlled Deep Reinforcement Learning

Thesis supervised by	Pierre SENS	Supervisor
	Amina BOUBENDIR	Co-Supervisor
	Fabrice GUILLEMIN	Co-Supervisor

Committee members

Referees	Stefano SECCI	Professor at Conservatoire National des Arts et Métiers
	Yassine HADJADJ-AOUL	HDR Associate Professor at Université de Rennes 1
Examiners	Adlen KSENTINI	Professor at Eurecom
	Anne FLADENMULLLER	Professor at Sorbonne Université
	Sylvaine KERBOEUF	Senior Researcher at Nokia Bell Labs
Supervisors	Pierre SENS	Professor at Sorbonne Université
	Amina BOUBENDIR	Research Project Manager at Orange Labs
	Fabrice GUILLEMIN	HDR Research Program Manager at Orange Labs

Acknowledgements

Undertaking a PhD thesis requires considerable personal effort, but the road to obtaining a doctorate is certainly not one that is traveled alone. That is why I would like to dedicate the first section of this manuscript to those who have contributed directly and indirectly to this achievement.

First of all, I would like to thank my thesis supervisors, Amina Boubendir, Fabrice Guillemin and Pierre Sens. I am really honored to have been accepted as a PhD student by such great researchers and very grateful for their advice, availability, attention and trust during these last three years. Knowing them and working as a team with them has been an enlightening experience in many ways and has strongly contributed to my personal and professional development during the last three years.

Next, I would like to thank Prof. Stefano Secci and Prof. Yassine Hadjaj-Aoul for the time and effort invested in the review and evaluation of this manuscript. I would also like to thank Prof. Adlen Ksentini, Prof. Anne Fladenmuller and Dr. Sylvaine Kerboeuf for agreeing to be part of my evaluation committee as examiners.

I express my gratitude to Jean-François Maudoux, head of the TEAM team at Orange, Nabil Charkani El Hassani, director of the Network Architecture & Automation department at Orange and Brigitte Cardinael, director of the Software Infrastructure research domain at Orange for receiving me so warmly from my first day and providing me with all the support needed.

I am also very grateful to all my colleagues at Orange and LIP6 for having contributed to make my PhD a very memorable experience, specially to Alessandro Aimi, Abderaouf Kichaine, Christian Destre, David Mathieu, Ilhem Fajjari, Michel Ribeyron, Samuel Bertrand Dalechamps for their strong support and all the moments spent together. I am also particularly grateful to Salah Bin Ruba for the collaborative work done during his stay in Orange; and to Bruno de Moura Donassolo and Nicolas Sivan for all the moments spent together and for helping me with my integration at the beginning of my PhD.

I had the great opportunity to work on three collaborative research projects and to be part of a fruitful research environment during my PhD at Orange and LIP6. I would like to give special thanks to all the researchers of the European project MON-B5G, of the INFLUENCE project in collaboration with Nokia Bell Labs, and of the NASA/EASI project for their collaboration and the opportunity to learn with them during the last three years.

I thank my father, José Jurandir Alves Esteves, my mother Ana Maria Esteves and my brother João Victor Alves Esteves for their love and endless support in my life pursuits.

I also thank my friends, specially Olivier Cleuziou, Wesley da Silva Coelho, Paloma Ruiz, Hanny Boukerras, Zità Razafiarisson and Aurélien Brooke, for their support, confidence and for always being there for me when I needed.

Last but not least, I thank my partner Vitoria Rodrigues França for her love, her patience, her dedication and for having made the choice to share with me—every day—the happy and the difficult moments of these last three years.

Abstract

Network Slicing is a major stake in 5G networks and beyond, which is mainly enabled by Network Function Virtualization (NFV) and Software Defined Networks (SDN). These two paradigms enable telcos to offer virtual networks meeting specific needs of the vertical markets on the top on the same shared physical infrastructure. An important challenge in the implementation at scale of Network Slicing is Network Slice Placement and the optimal allocation of virtualized resources. This can be formulated as a multi-objective Integer Linear Programming (ILP) problem. However, ILP suffers from combinatorial explosion when solving $\mathcal{NP}$-hard problem, especially for large-scale scenarios. Heuristics and Machine Learning (ML) algorithms have been investigated as efficient means of tackling this kind of problem.

This PhD thesis investigates how to optimize Network Slice Placement in distributed large-scale infrastructures focusing on online heuristic and Deep Reinforcement Learning (DRL) based approaches. First, we rely on ILP to propose a data model for enabling on-Edge and on-Network Slice Placement. In contrary to most studies related to placement in the NFV context, the proposed ILP model considers complex Network Slice topologies and pays special attention to the geographic location of Network Slice Users and its impact on the End-to-End (E2E) latency. Extensive numerical experiments show the relevance of taking into account the user location constraints.

Then, we rely on an approach called the "Power of Two Choices"(P2C) to propose an online heuristic algorithm for the problem which is adapted to support placement on large-scale distributed infrastructures while integrating Edge-specific constraints. The evaluation results show the good performance of the heuristic that solves the problem in few seconds under a large-scale scenario. The heuristic also improves the acceptance ratio of Network Slice Placement Requests when compared against a deterministic online ILP-based solution.

Finally, we investigate the use of ML methods, more specifically DRL, for increasing scalability and automation of Network Slice Placement considering a multi-objective optimization approach to the problem. We first propose a DRL algorithm for Network Slice Placement which relies on the Advantage Actor Critic algorithm for fast learning, and Graph Convolutional Networks for feature extraction automation. Then, we propose an approach we call Heuristically Assisted Deep Reinforcement Learning (HA-DRL), which uses heuristics to control the learning and execution of the DRL agent. We evaluate this solution trough simulations under stationary, cycle-stationary and non-stationary network load conditions. The evaluation results show that heuristic control is an efficient way of speeding up the learning process of DRL, achieving a substantial gain in resource utilization, reducing performance degradation, and is more reliable under unpredictable changes in network load than non-controlled DRL algorithms.

Key-words: Network Functions Virtualization, Network Slicing, Placement, Large-scale infrastructures, Optimization, Automation, Deep Reinforcement Learning, Heuristics.

Résumé

Le découpage du réseau est un enjeu majeur des réseaux 5G et au-delà, ce qui est principalement permis par la virtualisation des fonctions réseau (NFV) et les réseaux définis par logiciel (SDN). Ces deux paradigmes permettent aux opérateurs de télécommunications de proposer des réseaux virtuels répondant aux besoins spécifiques des marchés verticaux en utilisant une même infrastructure physique partagée. Un défi important dans la mise en œuvre à l'échelle du découpage du réseau est le placement des tranches de réseau et l'allocation optimale des ressources virtualisées. Ce problème peut être formulé comme un problème d'optimisation multi-objectifs via la programmation linéaire en nombres entiers (ILP). Cependant, l'ILP souffre d'une explosion combinatoire lors de la résolution de problèmes $\mathcal{NP}$-difficiles, en particulier pour les scénarios à grande échelle. Les heuristiques et les algorithmes d'apprentissage automatique (ML) sont envisagés comme des moyens efficaces pour résoudre ce type de problème.

Cette thèse examine comment optimiser le placement de tranches (slices) de réseau dans les infrastructures distribuées à grande échelle en se concentrant sur des approches heuristiques en ligne et basées sur l'apprentissage par renforcement profond (DRL). Tout d'abord, nous nous appuyons sur l'ILP pour proposer un modèle de données permettant le placement de tranches de réseau sur le bord et le coeur du réseau. Contrairement à la plupart des études relatives au placement de fonctions réseau virtualisées, le modèle ILP proposé prend en compte les topologies complexes des tranches de réseau et accorde une attention particulière à l'emplacement géographique des utilisateurs des tranches réseau et à son impact sur le calcul de la latence de bout en bout. Des expérimentations numériques nous ont permis de montrer la pertinence de la prise en compte des contraintes de localisation des utilisateurs.

Ensuite, nous nous appuyons sur une approche appelée "Power of Two Choices" pour proposer un algorithme heuristique en ligne qui est adapté à supporter le placement sur des infrastructures distribuées à grande échelle tout en intégrant des contraintes spécifiques au bord du réseau. Les résultats de l'évaluation montrent la bonne performance de l'heuristique qui résout le problème en quelques secondes dans un scénario à grande échelle. L'heuristique améliore également le taux d'acceptation des demandes de placement de tranches de réseau par rapport à une solution déterministe en ligne en utilisant l'ILP.

Enfin, nous étudions l'utilisation de méthodes de ML, et plus particulièrement de DRL, pour améliorer l'extensibilité et l'automatisation du placement de tranches réseau en considérant une version multi-objectif du problème. Nous proposons d'abord un algorithme DRL pour le placement de tranches réseau qui s'appuie sur l'algorithme "Advantage Actor Critic" pour un apprentissage rapide, et sur les réseaux convolutionels de graphes pour l'extraction de propriétés. Ensuite, nous proposons une approche que nous appelons "Heuristically Assisted DRL" (HA-DRL), qui utilise des heuristiques pour contrôler l'apprentissage et l'exécution de l'agent DRL. Nous évaluons cette solution par des simulations dans des conditions de charge de réseau stationnaire, ensuite cyclique et enfin non-stationnaire. Les résultats de l'évaluation montrent que le contrôle par heuristique est un moyen efficace d'accélérer le processus d'apprentissage du DRL, et permet d'obtenir un gain substantiel dans l'utilisation des ressources, de réduire la dégradation des performances et d'être plus fiable en cas de changements imprévisibles de la charge du réseau que les algorithmes DRL non contrôlés.

Mots-clés: Virtualisation des Fonctions Réseau, Découpage du réseau, infrastructures à large échelle, optimisation, automatisation, apprentissage par renforcement profond, heuristiques.

Contents

List of acronyms

A3C	Assyncronous Advantage Actor Critic
AI	Artificial Intelligence
BEF	Best Effort
CCP	Central Cloud Platform
CDC	Core Data Center
DC	Data Center
DNN	Deep Neural Network
DRL	Deep Reinforcement Learning
E2E	End-to-End
EDC	Edge Data Center
EMBB	Enhanced Mobile Broad Band
EPC	Evolved Packet Core
ETSI	European Telecommunication Standards Institute
FCAPS	Fault, Configuration, Accounting, Performance and Security
GAR	Global Acceptance Ratio
GCN	Graph Convolutional Networks
HA-DRL	Heuristically Assisted Deep Reinforcement Learning
ILP	Integer Linear Programming
ML	Machine Learning
NF	Network Function
NFV	Network Functions Virtualization
NSP	Network Slice Placement
NSPR	Network Slice Placement Request

NSU . . . . . Network Slice User

P2C . . . . . Power of Two Choices

PNF . . . . . Physical Network Function

PSN . . . . . Physical Substrate Network

QoE . . . . . Quality-of-Experience

QoS . . . . . Quality-of-Service

RAN . . . . . Radio Access Network

RGCN . . . . . Relational Graph Convolutional Network

RL . . . . . Reinforcement Learning

SDN . . . . . Software Defined Networking

SFC . . . . . Service Functions Chain

SFC-P . . . . . Service Functions Chain Placement

SLA . . . . . Service Level Agreement

TAR . . . . . Acceptance Ratio per training phase

UAP . . . . . User Access Point

URLLC . . . . . Ultra Reliable Low Latency Communications

VL . . . . . Virtual Link

VM . . . . . Virtual Machine

VNE . . . . . Virtual Network Embedding

VNF . . . . . Virtualized Network Function

VNF-FG . . . . . VNF Fowarding Graph

VNF-FGC . . . . . VNF Fowarding Graph Composition

VNF-FGE . . . . . VNF Forwarding Graph Embedding

VNF-PC . . . . . VNF Placement and Chaining

Chapter 1

General Introduction

Contents

We present in this Chapter the context and motivations of this PhD thesis (Section 1.1), the challenges and problem statement (Section 1.2), the thesis scope and research questions (Section 1.3), the summary of contributions (Section 1.4), the publications and reports (Section 1.5), and lastly, the organization of this manuscript (Section 1.6).

1.1 Context and motivations

Network Functions Virtualization (NFV) is a relatively new paradigm for Network Function (NF) implementation based on the decoupling between NF's logic and its hosting hardware. Hence, NFs that exist today as dedicated middleboxes may become software suites called Virtualized Network Functions (VNFs) designed to be deployed on commercial off-the-shelf servers. The main goal of NFV is to introduce flexibility in the deployment of NFs allowing the management of their life-cycle independently from the one of the underlying Physical Substrate Network (PSN) [1].

The flexibility introduced by NFV is an enabler for Network Slicing. The Network Slicing concept can be seen as deploying multiple logical networks (as a series of interconnected VNFs) over the same PSN shared between the latter. This would be achieved through logical isolation of physical network resources. The main objective of Network Slicing is to allow Network Operators to accurately satisfy a wide range of user's needs providing fully customized services [2].

To achieve this goal, proper management and orchestration of Network Slices are essential. Network Slice orchestration and management processes include life-cycle management of Network Slices, VNFs and their associated Virtual Links (VLs). They also include Fault, Configuration, Accounting, Performance and Security (FCAPS) management [3].

Standardization bodies have proposed functional and resource-centric views of Network Slicing [4]–[6]. Multiple studies related to Network Slice orchestration have been carried out during the last few years [2]. However, automated and optimized orchestration of Network Slices remains a challenge. This PhD thesis is motivated by this challenge.

From a Network Operator's point of view we address the question of how to coordinate the use of different network resources to host Network Slices in an optimized way. More specifically, our objective is to design Network Slice orchestration mechanisms based on relatively simple algorithms to optimize the placement and chaining of the VNFs composing the Network Slices.

1.2 Challenges and problem statement

We consider the VNF placement and chaining problem for Network Slice orchestration, which we name Network Slice Placement (NSP). This problem consists in choosing the servers of the PSN in which the VNFs composing Network Slices are to be deployed and which physical links to use in order to steer traffic between these servers.

This problem contains a specific optimization objective, e.g., minimizing resource consumption, optimizing a specific Quality-of-Service (QoS) or Quality-of-Experience (QoE) metric, that must be satisfied [7]. A VNF placement and chaining decision is needed prior to the instantiation of a Network Slice but also in the other phases of the Network Slice life-cycle management, for instance, when the scaling of a VNF is done and a migration is needed.

When considering this problem, the main concern of Network Operators is to ensure the strict resource and QoS requirements, e.g., computing, storage, bandwidth, End-to-End(E2E) latency, service availability. These requirements are formally defined in the Service Level Agreement (SLA) of a Network Slice. For achieving this, Network Operators have to properly manage resources of a large-scale PSN.

This large-scale PSN has a complex topology with limited amount of heterogeneous resources on its nodes (e.g., computing, storage, networking) and on its links (e.g., bandwidth, radio frequencies). In this PhD thesis, we assume that there are a huge number of Network Slices with volatile demands that share the PSN's resources. To fulfil the requirements of this wide variety of Network Slices, management actions and decisions such as VNF placement and chaining need to be triggered automatically and be performed efficiently to prevent from costly SLA violations.

In spite of the huge attention dedicated to Network Slice orchestration in the recent few years, researchers have not completely dealt with the inherent complexity. Most of works do not take into account the QoS metrics existing on the SLA and focus only on resource requirements. Also, they are mostly based on centralized classical optimization approaches such as Integer Linear Programming (ILP). Centralized classical optimization is very effective with regard to the use of resources and compliance with constraints, but might be too complex for a large-scale network. Also, it does not provide the flexibility to allow the autonomic orchestration of Network Slices based on fast and automatic decisions. Besides, the dynamic nature of network traffic, and the assumption of volatility and non-stationarity of network loads in large-scale networks with limited storage and computing capacities at the Edge are rarely considered.

Hence, the design of algorithms aimed at optimizing the placement and chaining of VNFs for Network Slice orchestration is still an important topic. These algorithms must take into account not only the VNF and Virtual Links (VLs) constraints (e.g., latency, necessary resources) but also the dimension and the topology of the network greately imposed by the technological constraints of deployment. Simple algorithms, possibly based on Machine Learning (ML), to dynamically allocate the resources orchestrated by the Network Operator are seen as an alternative to the centralized classic optimization approaches and may contribute towards autonomic Network Slice orchestration.

1.3 Thesis scope and research questions

In Figure 1.1, we present the scope of this PhD thesis. The core of the thesis is at the crossroad of three domains represented by the three circles: Management and Orchestration; Distributed infrastructures, Edge/MEC, Large-scale networks; ML, Artificial Intelligence (AI). The intersections between these three domains are the more precise aspects that we investigated in the state-of-the-art analysis to obtain precise research questions. After analyzing the state-of-the-art regarding placement optimization algorithms and models, orchestration automation, we formulate the three research questions investigated in this PhD thesis as described in Sections 1.3.1, 1.3.2, and 1.3.3.

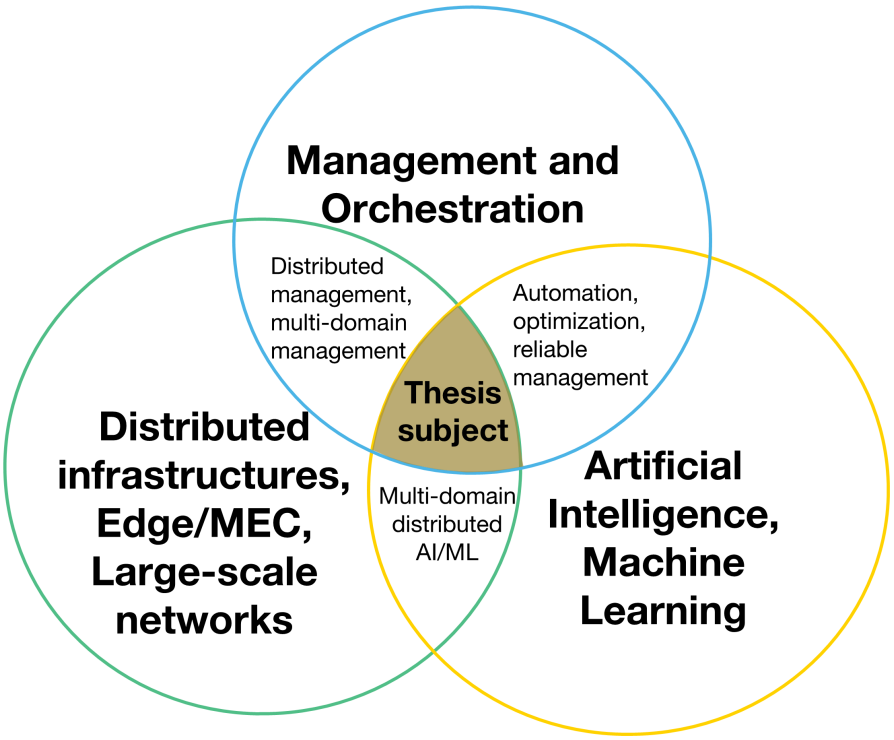


Figure 1.1: Thesis scope

1.3.1 Modeling and definition of large-scale network optimization algorithms

The first question we investigate is:

What are the accurate optimization models and algorithms for Edge-enabled Network Slice placement in large-scale networks?

We define a large-scale network considering two levels. The first level is the PSN and the second level is Network Slice. We assume that a large-scale the Network Slice can be a complex meshed topology comprising hundreds of nodes depending on the implementation model chosen for the VNFs (e.g., monolithic, micro-services, etc.). A large-scale PSN contains thousands of nodes and links. Our goal is to define models and algorithms to solve NSP problem in this large-scale network scenario. Moreover, efficient strategies for modeling and evaluating the performance of the proposed algorithms are needed.

1.3.2 Orchestration mechanisms for Edge-specific constraints

The second question is:

How can we ensure QoS/QoE for Network Slice Users in a converged network-Edge/MEC context?

We study different Network Slicing use-cases to understand the application constraints that need to be respected to ensure QoS/QoE for the Network Slice Users (NSUs). In this context, we focus on Edge-specific constraints and address the specific challenge of how to ensure E2E latency requirements from the NSUs perspective. We investigate appropriated management tools and mechanisms and treat user location as an important constraint to be included in the placement algorithm.

1.3.3 AI/ML for automated and reliable network management

The third question is:

How do we build automated and scalable algorithms to compute optimal placement decisions that are robust to changes in network behaviour?

The evolution of the network state in time is treated in this question. Network Slice Placement Requests (NSPRs) arrive over time and stay in the system during a given service duration. But, in real networks, the rate of arrivals of slices varies over time. Hence, in the context of large-scale networks, we investigate to which extent automatic placement algorithms such as Deep Reinforcement Learning (DRL) are robust and reliable when used under realistic network load conditions. As a complement of the placement algorithms and models, monitoring and telemetry for autonomic management and automated orchestration of Network Slices are also investigated in this question.

1.4 Summary of contributions

In this section we summarize the three contributions of this PhD thesis—related with the three research questions we investigate—on Sections 1.4.1, 1.4.2 and 1.4.3, respectively.

1.4.1 Enabling on-Edge and on-Network Slice Placement: an ILP-based Solution

In spite of the numerous papers about placement in virtual networks, most of them frequently ignore the geographic dimension of placement decision. Existing studies often do not take into account neither the user's location when solving the problem nor user location implications, especially in the E2E latency calculation. These aspects are utmost important to enable on-Edge and on-Network slice placement and we specifically address this challenge.

In this context, the first contribution of this PhD thesis, which is presented in **Chapter 4**, gathers three main contributions as follows:

1. Propose an E2E latency model that integrates the user location as an end-point of the Network Slice, and improves scalability by exploiting the possibility of grouping NSUs instead of considering them individually;

2. Deal with complex Network Slice topologies, going beyond the currently studied Service Functions Chain (SFC) concept;

3. Set no restrictions on the placement location of two VNFs of the same Network Slice.

The proposed model is formalized mathematically using ILP and implemented in a latency-aware NSP solution [8]. We use CPLEX solver to assess the performance of the model [9].

1.4.2 Optimizing Large Scale Network Slice Placement: a Heuristic using the Power of Two Choices

Numerous papers about placement in virtual networks use heuristic-based approaches to solve associated optimization problems. However, most of them do not jointly address the large-scale network aspect and the Edge-specific constraints and thus the direct impact on QoS metrics (notably, E2E latency).

The second contribution of this PhD thesis, which is presented in **Chapter 5**, gathers two main contributions:

1. An original method of placing network slices through a heuristic based on the Power of Two Choices (P2C) algorithm [10] which is adapted to large-scale network scenarios and integrate both Edge-specific constraints related to user location and strict E2E latency requirements;

2. A policy for selecting servers for VNF placement that offloads Edge Data Centers (EDCs) and improves Network Slice acceptance ratio.

We implement the heuristic inside an NSP solution [11] and compare it to ILP-based placement algorithms [12].

1.4.3 Automating Multi-objective Network Slice Placement with Machine Learning: a Heuristically Assisted Deep Reinforcement Learning solution

From an operational perspective, heuristic approaches are more suitable than ILP as they yield faster placement results. The drawback of heuristic approaches is that they give sub-optimal solutions. To address this issue, ML offer a corpus of methods, such as DRL, which are able to overcome the convergence issues of ILPs while being more accurate than heuristics.

However, from a practical point of view, ensuring that a DRL agent converges to an optimal policy is still a challenge. A first important drawback is that DRL agents act as self-controlled black boxes. In addition, there are a large number of hyper-parameters to fine-tune in order to ensure an adequate equilibrium between exploring solutions and exploiting the knowledge acquired via training. While there are techniques to improve the efficiency of the solution exploration process (e.g., ϵ-greedy, entropy regularization), their use may also lead to situations of instability, where the algorithm may diverge from the optimal point.

Another issue with DRL is its use in non-stationary environments. As a matter of fact, when the environment is continually changing the rules, the algorithm has trouble in using the acquired knowledge to find optimal solutions. The usage of the DRL algorithm in a online fashion can then become impractical. Most of the existing works applying DRL to placement in virtual networks assume a stationary environment, i.e., with static network load. However, traffic conditions in networks are basically non-stationary with daily and weekly variations and subject to drastic changes (e.g., traffic storm due to an unpredictable event).

To overcome this unsuitable behaviour of DRL agents, based on the concept of Heuristically Accelerated Reinforcement Learning [13], we introduce the concept of Heuristically Assisted Deep Reinforcement Learning (HA-DRL) and we apply it in a fully online learning scenario with time-varying network loads to show how this strategy can accelerate and stabilize the convergence of DRL techniques when applied to the NSP.

This contribution, which is described in **Chapter 6**, gathers three main contributions:

1. Combines Graph Convolutional Network (GCN)—to automatically extract PSN related features—and Advantage Actor Critic algorithm—for optimal policy learning—to solve multi-objective NSP optimization problem [14];
2. Provides a network load model to network slice infrastructure conditions with time-varying network loads;
3. Reinforces the DRL learning process by using the P2C based heuristic we propose in [12] to control the DRL convergence.

We implement the proposed algorithms inside an NSP solution [15] and evaluate them in static [14], cycle-stationary [16] and non-stationary [17] network load scenarios.

1.4.4 Integration to MON-B5G research project

The contributions of this PhD thesis were carried out in the framework of the MON-B5G European research project [18]. MON-B5G proposes a novel autonomic management and orchestration framework, heavily leveraging distribution of operations together with state-of-the-art data-driven AI-based mechanisms. MON-B5G is designed around a hierarchical approach that allows the flexible and efficient management of network tasks, while introducing a diverse set of centralization levels through an optimal adaptive assignment of monitoring, analysis, and decision-making tasks.

The MON-B5G vision is depicted in Figure 1.2. MON-B5G proposes to split the centralized management system into several management sub-systems, distributing both the intelligence as well as the decision making across various components. Each technological domain may have one or several distributed management elements while each distributed management element includes enhanced data-driven Monitoring System (MS block on Figure 1.2), Analytics Engine (AE block on Figure 1.2) and Decision Engine (DE block on Figure 1.2) components. The project is organized in seven work packages and this PhD thesis have contributed to all of them except packages 1 and 5. The studies and algorithms designed in this PhD thesis were presented in project meetings and served as inputs for the projects' technical deliverables. We also contribute to the integration of the proposed algorithms with other MON-B5G components to be trialed over the projects' 5G test-bed and to the preparation of Proofs-of-Concept planned to be performed at the end of the project to showcase MON-B5G concept using two use cases.

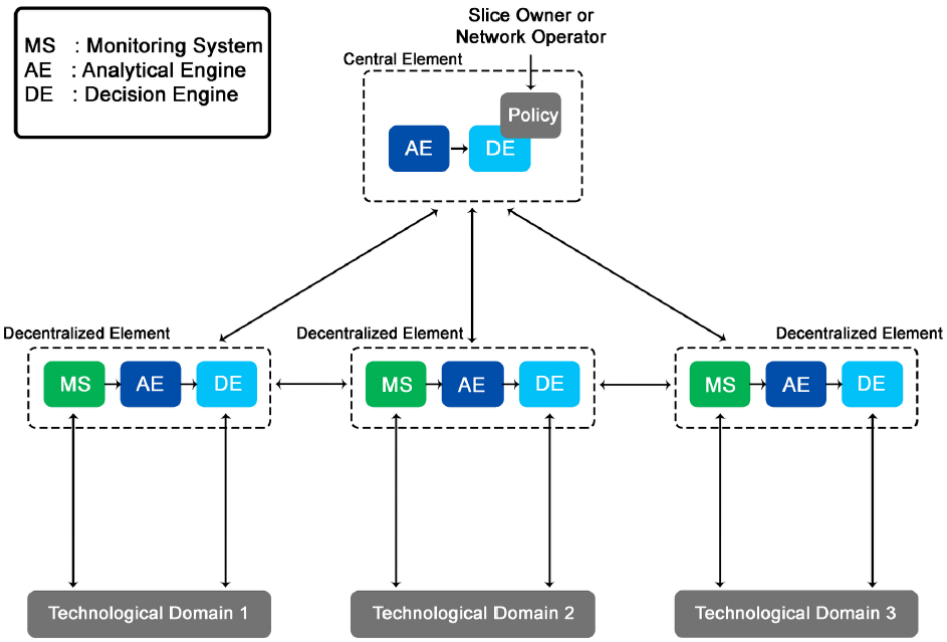


Figure 1.2: The MON-B5G vision of a decentralized management system for network slicing

1.5 Publications and reports

Table 1.1 presents the publications for this PhD thesis describing the publication title (with citation), the publication venue, the publication type and the current status.

The work performed during this PhD thesis have also served as contribution to the following MON-B5G technical deliverables:

- Deliverable 2.1, 1st release of the MON-B5G architecture;
- Deliverable 2.2, Techno-economic analysis of the beyond 5G environment use case requirements and KPIs;
- Deliverable 3.1, Initial report on AI driven techniques for the MON-B5G AE/MS;
- Deliverable 4.1, Initial report on AI driven techniques for the MON-B5G DE.

1.6 Organization of the manuscript

The rest of the manuscript is organized as follows: Chapter 2, introduces the background; Chapter 3, presents the state-of-the-art analysis; Chapter 4, describes the first contribution of this PhD thesis, namely, *Enabling on-Edge and on-Network Slice Placement: an ILP-based Solution*; Chapter 5, describes the second contribution of this PhD thesis, namely, *Optimizing Large Scale Network Slice Placement: a Heuristic using the Power of Two Choices*; Chapter 6, describes the third and final contribution of this PhD thesis, namely, *Automating Multi-objective Network Slice Placement with Machine Learning: a Heuristically Assisted Deep Reinforcement Learning solution*; and Chapter 7, concludes the manuscript and present the perspectives.

#	**Publication title**	**Publication venue**	**Publication type**	**Status**
1	Location-based Data Model for Optimized Network Slice Placement [9]	6th IEEE International Conference on Network Softwarization (NetSoft 2020)	Full paper	Published
2	Heuristic for Edge-enabled Network Slicing Optimization using the "Power of Two Choices" [12]	16th International Conference on Network and Service Management (CNSM 2020)	Full paper	Published
3	Optimized Network Slicing Proof-of-Concept with Interactive Gaming Use Case [8]	23rd Conference on Innovation in Clouds, Internet and Workshops (ICIN 2020)	Demo paper	Published
4	Edge-enabled Optimized Network Slicing in Large Scale Networks [11]	11th International Conference on Network of the Future (NoF 2020)	Demo paper	Published
5	Controlled Deep Reinforcement Learning for Optimized Slice Placement [19]	IEEE International Mediterranean Conference on Communications and Networking (MeditCom 2021)	Short paper	Presented
6	DRL-based Slice Placement Under Non-Stationary Conditions [16]	17th International Conference on Network and Service Management (CNSM 2021)	Full paper	Presented
7	DRL-based Slice Placement Under Realistic Network Load Conditions [15]	17th International Conference on Network and Service Management (CNSM 2021)	Demo paper	Presented
8	A Heuristically Assisted Deep Reinforcement Learning Approach for Network Slice Placement [14]	IEEE Transactions on Network and Service Management Special Issue on Embracing Artificial Intelligence	Journal paper	Accepted
9	On the Robustness of Controlled Deep Reinforcement Learning for Slice Placement [17]	Journal of Network and Systems Management Special Issue on Network management for beyond 5G systems	Journal paper	Accepted

Table 1.1: Summary of publications

Chapter 2

Background

Contents

Introduction

In this chapter, we introduce some background information for a better understanding of this PhD thesis. In Section 2.1, we present useful concepts needed to understand the scope of this work. In Section 2.2, we focus on the approaches and theoretical methods adopted during the research.

2.1 Concepts

In this section, we present the two main concepts studied during this PhD thesis: the NFV paradigm, in Section 2.1.1, and the Network Slicing concept, in Section 2.1.2.

2.1.1 The Network Functions Virtualization paradigm

This section presents the NFV paradigm. We first give a definition of NFV, then we present the goals of NFV. Finally, we give an overview of the NFV architectural framework.

Definition of NFV

To understand the NFV paridigm, first, it is necessary to explain the Network Function (NF) definition. According to the European Telecommunication Standards Institute (ETSI), an NF is a functional building block within a network infrastructure which has well-defined external interfaces and a well-defined functional behaviour [20]. In practical terms, an NF is today often a network node or physical appliance [20] like a router or a gateway.

The NFV paradigm is then a relatively new paradigm for NFs implementation based on the decoupling between NF's logic and its hosting hardware. Hence, NFs that exist today as dedicated middleboxes called Physical Network Functions (PNFs), may become software components called Virtual Network Functions (VNFs) designed to be deployed on commercial off-the-shelf servers instead of proprietary hardware appliances.

NFV goals

The main goal of NFV is to introduce flexibility in the deployment of NFs allowing the management of their life-cycle independently from the one of the underlying PSN. NFV also aims to improve [1]:

- **capital efficiencies**, by using general purpose servers as hardware shared between VNFs, which improves the usage of resources and also reduce the number of different hardware architectures available;
- **service scalability**, by decoupling functionality from location allowing software to be located at the most appropriate places, e.g., customers' premises, network exchange points, central offices, data centers (DCs), etc, enabling time of day reuse, and facilitating resource sharing;
- **service innovation rapidity**, by enabling software-based service deployment;
- **operational efficiencies**, since the NFV framework allows common automation and operating procedures;
- **power usage**, achieved by migrating workloads and powering down unused hardware;
- **standardization and openness of interfaces between VNFs and the infrastructure and associated management entities**, so that such decoupled elements can be provided by different vendors.

NFV architectural framework

The NFV concept is realized by the NFV architectural framework depicted in Figure 2.1. The main layers and functional blocks of the NFV architecture are summarized in the following paragraphs according to standard [1]. The interested reader can obtain more information, especially about NFV reference points and interfaces, on [1].

Operations and Business Support Systems layer The Operations and Business Support Systems (OSS/BSS block in Figure 2.1) layer includes the collection of systems and management applications that service providers use to operate their business.

VNF layer The VNF layer contains the NFs after being passed from PNF to VNF and the Entity Managers (EM blocks on Figure 2.1) that perform typical management functionality for one or more VNFs. Functional behaviour and external operations interfaces are expected to be the same in a VNF and in a PNF. A VNF can be composed of multiple components called VNF Components. Each VNF Component of a VNF can be instantiated in a different virtual machine (VM) or container. All VNF Components of a VNF can be instantiated in the same VM or container.

NFV Infrastructure layer The NFV Infrastructure (NFVI block on Figure 2.1) represents the environment in which VNFs will be executed (computing, storage and network) providing processing, storage and connectivity between VNFs through the virtualisation layer. The NFV Infrastructure is distributed in Points-of-Presence in many locations. Hardware and virtualisation layer are seen as a single entity by the VNFs. The virtualisation layer serves to:

- Abstracting and logically partitioning physical resources, commonly as a hardware abstraction layer;
- Enabling the software that implements the VNF to use the underlying virtualized infrastructure;
- Providing virtualized resources to the VNFs.

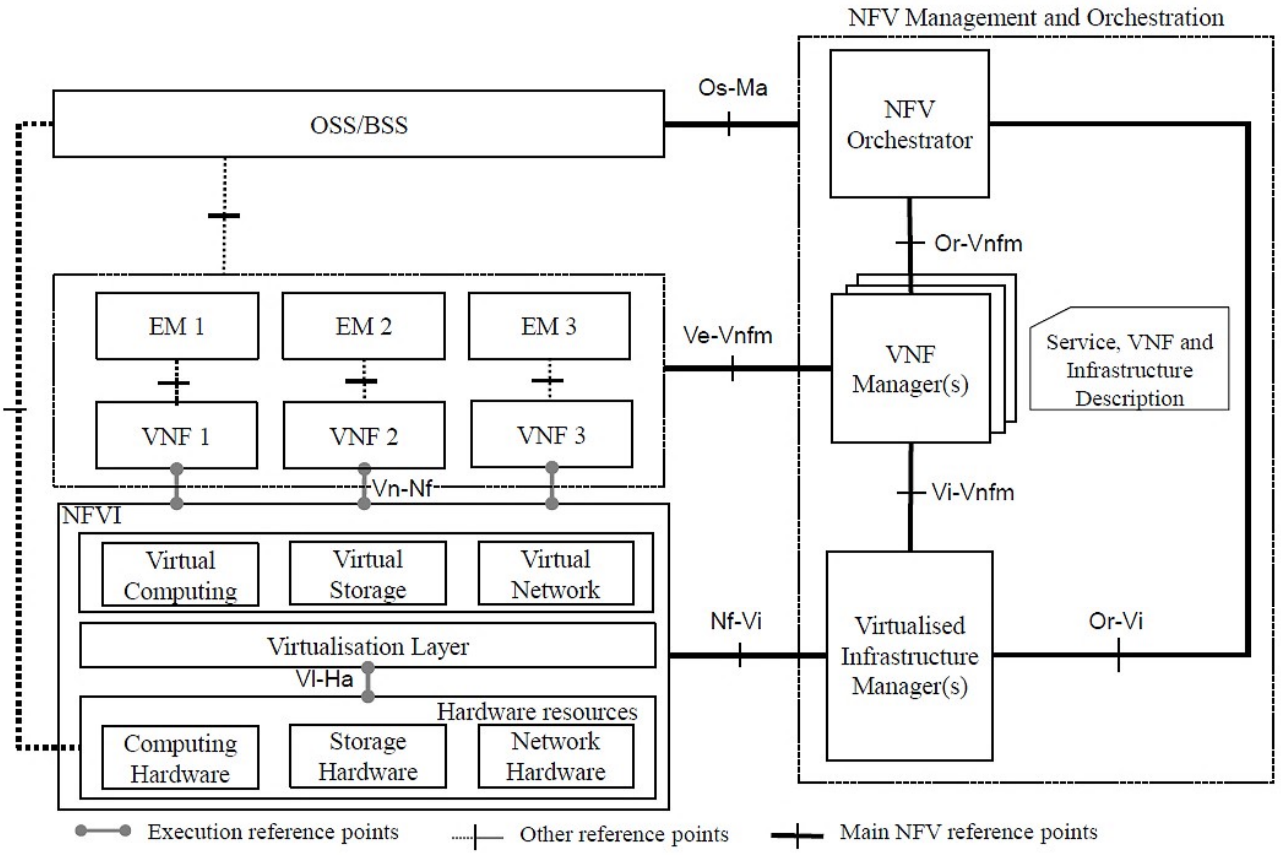


Figure 2.1: NFV Architectural Functional Blocks. Source: [1].

NFV Management and Orchestration layer The NFV Management and Orchestration layer covers orchestration and life-cycle management of physical and software resources to support infrastructure virtualization and life-cycle of VNFs [1]. The NFV Management and Orchestration layer comprises four main building blocks described in the following:

- **NFV Orchestrator:** is the interface between the Operations and Business Support Systems with other layers passing by the VNF Manager and Virtualized Infrastructure Manager. It is the main responsible for orchestration and management of the NFV Infrastructure hardware and software resources and of realizing network services on the NFV Infrastructure;

- **VNF Manager:** is responsible for performing VNF life-cycle management, that is, performing instantiation, update, query, scaling and termination of VNFs. Multiple VNF Managers may be deployed. A VNF Manager may be deployed for each VNF, or a VNF Manager may serve multiple VNFs;

- **Virtualized Infrastructure Manager:** manages the interactions between the VNFs and the infrastructure. It is responsible for resource allocation to the VMs or containers exiting into the NFV Infrastructure (computing, storage and network connectivity) being also responsible for inventory of software on NFV Infrastructure;

- **Service, VNF and Infrastructure description:** contains information about Network Services, infrastructure and VNFs used internally within NFV Management and Orchestration. The NFV Management and Orchestration handles this information and exposes parts of it applicable to functional blocks as needed.

2.1.2 The Network Slicing concept

In this section, we present the Network Slicing concept. First, we discuss the Network Slicing definitions and then, we present Network Slice life-cycle management and orchestration.

Network Slicing definitions

The flexibility introduced by NFV is an enabler for Network Slicing. Standardization bodies in telecommunications have proposed different architectures and views of Network Slicing that have evolved differently across time [4]–[6]. A more resource oriented view of the the Network Slicing concept can be seen as deploying multiple logical networks (as a series of interconnected VNFs) over the same PSN [6].

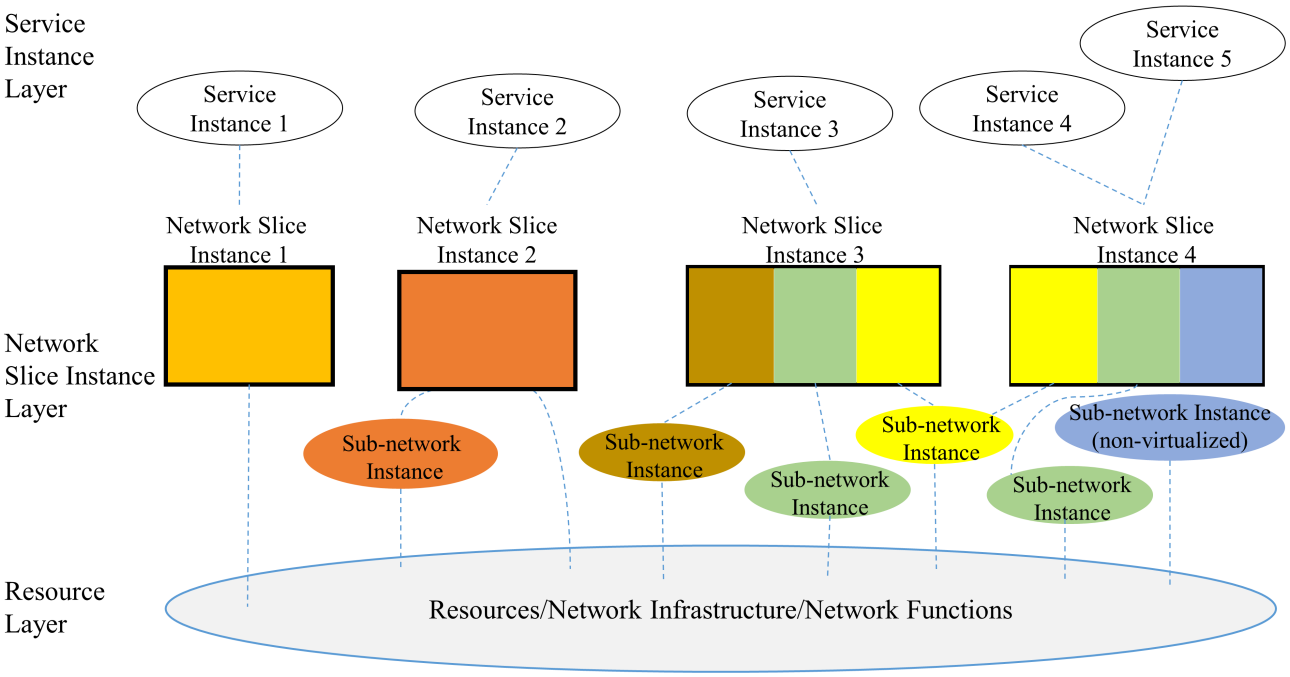


Figure 2.2: Network Slicing conceptual view. Source: [4].

A more functional oriented definition of Network Slicing is also defined [4], [5] and such representation of the Network Slicing concept is given by Figure 2.2. The Network Slicing concept comprise then three functional layers:

- **Service Instance layer:** represents the business or end-user services supported by a Network Slice Instance. These services can be either provided by a network operator or a 3rd party;
- **Network Slice Instance layer:** represents the Network Slice Instances. A Network Slice Instance comprises a set of NFs and resources forming a complete instantiated logical network which provides the network characteristics and functionalities required by the Service Instances it supports. A Network Slice Instance is typically the realization of a Network Slice Blueprint that represents a complete description of the expected Network Slice functionalities and characteristics which are closely related to the type of characteristics it would provide;
- **Resource layer:** represents the network infrastructure physical and virtualized resources and VNFs used to run Sub-network instances composing Network Slice Instances. A Sub-network Instance comprises a set of NFs and the resources they need. A Sub-network instance is also closely related to a Sub-network Blueprint which describes the structure and components of a Sub-network.

Network Slice management and orchestration

The main objective of Network Slicing is to allow network operators and 3rd parties to accurately satisfy a wide range of user's needs providing fully customized services [2]. As a matter of fact, network slices imply different requirements or constraints (e.g., minimum acceptable throughput or E2E latency) in addition to traditional IT resource requirements in terms of computing and storage. All these constraints apply to SLA as requirements of the Network Slice tenant (i.e., user of Network Slice). Hence, proper management and orchestration of Network Slices, VNFs and their associated VLs are essential to meet and maintain the SLAs of concurrent Network Slices.

Network Slice management and orchestration processes include life-cycle management of Network Slices, VNFs and their associated VLs. They also include FCAPS management [3].

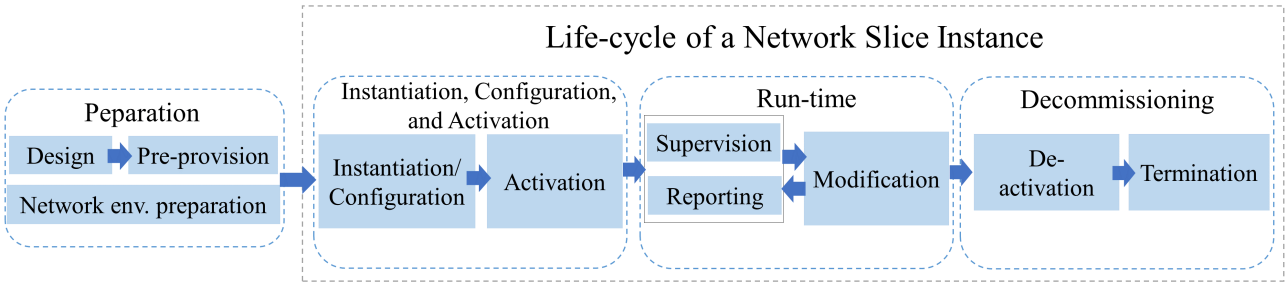


Figure 2.3: Network Slice life-cycle. Source: [5].

According to 3GPP [5], the network slice life-cycle is described by four phases as illustrated by Figure 2.3:

- **Preparation:** includes activities prior to the existance of a Network Slice Instance, that is, the creation and verification of network slice template(s)/blueprint(s), the on boarding of these, preparing the necessary network environment which are used to support the life-cycle of network slice instances, and any other preparations that are needed in the network;
- **Instantiation, Configuration and Activation:** instantiation and configuration includes creation and configuration of all resources to be used by a Network Slice Instance. Activation includes any actions that makes the Network Slice Instance active, e.g., diverting traffic to it, provisioning databases, etc.
- **Run-time:** the run-time phase includes supervision/reporting (e.g., for KPI monitoring), as well as activities related to modification, (e.g., upgrade, reconfiguration), Network Slice Instance scaling, changes of Network Slice Instance capacity, changes of Network Slice Instance topology, association and disassociation of NFs with a Network Slice Instance.
- **Decommissioning:** the decommissioning phase includes deactivation (taking the Network Slice Instance out of active duty) as well as the reclamation of dedicated resources (e.g., termination or re-use of NFs) and configuration of shared/dependent resources. After decommissioning the Network Slice Instance does not exist anymore.

2.2 Approaches and theoretical methods

In this Section, we present the approaches and theoretical methods used during this PhD thesis. In Section 2.2.1, we introduce Graph Theory. In Section 2.2.2, we present ILP and in Section 2.2.3, we discuss heuristic and meta-heuristic methods. Finally, in Section 2.2.4, we introduce DRL, the ML field studied in this PhD thesis.

2.2.1 Graph Theory

In this section, we present some definitions from Graph Theory used in the remaining of this PhD thesis. Except when the contrary is explicitly mentioned, all the definitions are extracted from [21].

Definition 2.2.1 (Simple Graph) *A simple graph G consists of a non-empty finite set $V(G)$ of elements called vertices (or nodes), and a finite set $E(G)$ of distinct unordered pairs of distinct elements of $V(G)$ called edges. We call $V(G)$ the vertex set and $E(G)$ the edge set of G. An edge (v, w) is said to join the vertices v and w.*

Remark

In this PhD thesis, we use the notation $G = (V, E)$ when describing a simple graph with vertex set $V(G)$ and edge set $E(G)$.

Remark

In Figure 2.4, we present a simple graph example $G = (V, E)$ whose vertex set $V(G)$ is $\{u, v, w, z\}$ and whose edge set $E(G)$ is $\{(u, v), (u, w), (v, w), (w, z)\}$.

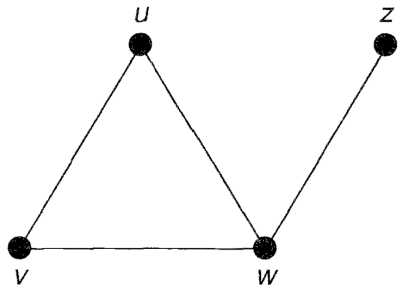


Figure 2.4: Simple Graph example. Source: [21].

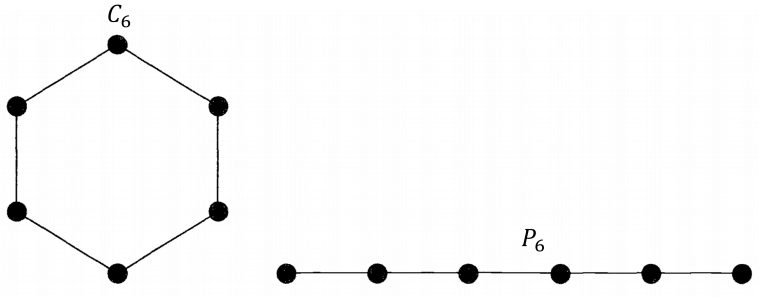


Figure 2.5: An example of cycle graph (C_6) and path graph (P_6). Source: [21].

Definition 2.2.2 (Directed Graph) *A directed graph, or digraph, D consists of a non-empty finite set $V(D)$ of elements called vertices, and a finite family $A(D)$ of* ***ordered*** *pairs of elements of $V(D)$ called arcs.*

Definition 2.2.3 (Weighted graph) *A weighted graph is a connected graph G with a non-negative number assigned to each edge (and possibly vertex) of G. The number assigned to each edge e is the weight of e, denoted by $w(e)$.*

Definition 2.2.4 (Walk) *Given a graph G, a walk in G is a finite sequence of edges of the form $(v_0, v_1), (v_1, v_2)$, $..., (v_{m-1}, v_m)$ also denoted by $v_0 \to v_1 \to v_2 \to ... \to v_m$, in which any two consecutive edges are adjacent or identical.*

Definition 2.2.5 (Path) *A walk $W = (v_0, v_1), (v_1, v_2), ..., (v_{m-1}, v_m)$ in a graph G is a path if all edges and vertices in W are distinct (except, possibly, $v_0 = v_m$).*

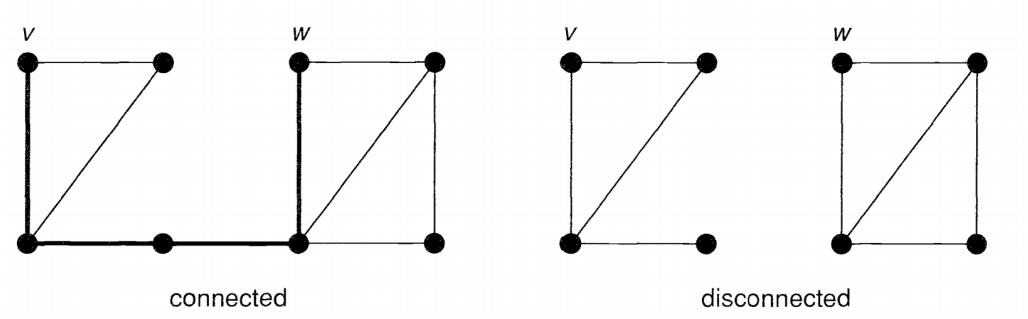


Figure 2.6: Example of connected and disconnected graphs. Source: [21].

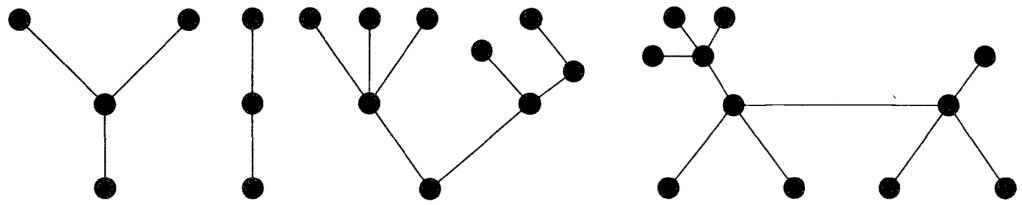


Figure 2.7: Examples of graphs that are trees. Source: [21].

Remark

A path is **closed** if $v_0 = v_m$, and a **closed path** containing at least one edge is a **cycle**.

Definition 2.2.6 (Connected Graph) *A graph G is connected if it cannot be expressed as the union of two graphs and disconnected otherwise.*

Remark

It can be demonstrated that a graph G is connected if and only if there is a path between each pair of vertices.

Definition 2.2.7 (Cycle graph) *A connected graph G which have all nodes with degree 2 is called cycle graph, denoted by C_n.*

Definition 2.2.8 (Path graph) *The graph obtained from a cycle graph with n vertices C_n by removing one edge is the path graph on n vertices, denoted by P_n.*

Definition 2.2.9 (Tree) *A three is a connected graph G with no cycles.*

Remark

It can be shown that saying that a graph T with n nodes is a tree is equivalent to saying that T contains no cycles and has $n - 1$ edges.

Definition 2.2.10 (Subgraph) *A subgraph of a graph G is a graph H, each of whose vertices belongs to $V(G)$ and each of whose edges belong to $E(G)$, that is, $V(H) \subseteq V(G)$ and $E(H) \subseteq V(H)$.*

Shortest Path problem and algorithms

A well studied graph problem which is relevant for this PhD thesis is the shortest path problem. We consider the definition of Elementary Shortest Path problem from [22].

Definition 2.2.11 (Elementary Shortest Path problem) *Given a weighted directed graph $G = (V, A)$ with arbitrary arc weights, the Elementary Shortest Path problem consists of finding a minimum-cost path between two nodes s and t (i.e., a path from s to t whose sum of link weights is minimal) such that each node of G is visited at most once.*

Remark

The elementary shortest path problem is studied in this PhD thesis. We use ILP to formulate the problem in Chapter 4 and routines based on the Dijkstra shortest path algorithm [23] to solve it on Chapters 5 and 6.

Remark

An important variant of the shortest path problem considered in this PhD thesis is the constrained shortest path problem. This problem is the same of the elementary shortest path problem but with an additional resource constraint $r(e)$ to each each edge e.

2.2.2 Integer Linear Programming

Before presenting ILP it is important to introduce Linear Programming. Linear Programming is a mathematical modeling technique used to describe an optimization problem using linear constraints and linear objective function [24].

The word programming does not refer here to computer programming; rather, it is essentially a synonym for planning. Thus, linear programming involves the planning of activities to obtain an optimal result, i.e., a result that reaches the specified goal best (according to the mathematical model) among all feasible alternatives activities given as follows [24],

$$\max_{x} Z = \sum_{j=1}^{n} c_j x_j \tag{2.1}$$

subject to:

$$a_{ij} x_j \leqslant b_i \qquad \forall i = 1, ..., n \tag{2.2}$$

$$x_j \geqslant 0 \qquad \forall j = 1, ..., m \tag{2.3}$$

where m is the number of activities, n is the number of resources, a_{ij} is the number of resource i requested by activity j, b_i is the amount of resource i, c_j is the contribution of activity j to the the objective function Z, and x_j is the decision variable which represents the level of activity j.

Different classes of solutions (i.e., classes of values for the decision variables) are identified for a Linear Programming model:

- **Feasible solution:** is a solution for which all the constraints are satisfied;
- **Infeasible solution:** is a solution for which at least one constraint is violated;
- **Optimal solution:** is a feasible solution that has the most favorable value of the objective function (e.g., Z function on the precedent example).

Linear Programming models can be solved graphically in the simplest cases. In more complex cases other algorithms such as Simplex or interior-point are used. However, an important assumption of Linear Programming is the divisibility, which allows decision variables to assume non-integer values.

This limits the use of Linear Programming since in many practical problems, the decision variables actually make sense only if they have integer values [24]. ILP appears then as a mathematical modeling technique using only linear functions in which some or all the variables are restricted to be integer—and often binary—values [24]. ILP is used in the modeling of multiple problems such as capital budgeting, warehouse location, scheduling of interrelated activities, multiple problems in graphs, etc. For instance, a standard ILP formulation for the shortest path problem (see Definition 2.2.11) is given as follows:

$$\min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{2.4}$$

s.t.:

$$\sum_{(i,j)\in\gamma^+(i)} x_{ij} + \sum_{(j,i)\in\gamma^-(i)} x_{ji} = \begin{cases} 1, & \text{if } i = s \\ -1, & \text{if } i = t \\ 0, & \text{otherwise} \end{cases} \qquad \forall i \in V \tag{2.5}$$

$$\sum_{(i,j)\in\gamma^+(i)} x_{i,j} \leqslant 1 \qquad \forall i \in V \tag{2.6}$$

$$x_{i,j} \in \{0,1\} \qquad \forall (i,j) \in A \tag{2.7}$$

where $\gamma^+(i)$ and $\gamma^-(i)$ are the sets of outgoing and incoming arcs of node i respectively, $c_{ij} \in \mathbb{R}$ is the cost of arc (i,j), and x_{ij} is a binary arc variable that take value 1 if the arc (i,j) belongs to the path and 0 otherwise. Constraints (2.5) are flow conservation constraints and are described in detail in Section 4.3.3 and Constraints (2.6) ensure that the outgoing degree of each node is at most one.

Whereas the the simplex method is the standard for solving Linear Programming models, enumerating techniques (e.g., branch-and-bound algorithm) are often the first approach to solve an ILP model. The branch-and-bound algorithm is based on a "divide and conquer" principle. As stated by [25], it proceeds by repeatedly partitioning the class of all feasible solutions of the model into smaller sub-classes in such a way that ultimately an optimal solution is obtained. The drawback of this kind of technique is that it requires extensive calculations since in the worst case, depending on the problem complexity, all the possible solutions need to be evaluated.

2.2.3 Heuristics and meta-heuristics

As discussed in Section 2.2.2, algorithms for obtaining optimal solutions for ILP models can suffer from combinatorial explosion when solving complex problems. As a consequence, ILP models for this kind of problem cannot be solved in an exact manner within a reasonable amount of time. As a consequence, approximate algorithms such as heuristics or meta-heuristics are often used in these cases. Heuristics are problem dependent; they are designed and applicable to a particular problem whereas meta-heuristics are general approximate algorithms applicable to a large variety of optimization problems [26].

Generally speaking, meta-heuristics solve instances of problems by exploring the solution search space of these instances, which is usually large. These algorithms achieve this by reducing the effective size of the space and by exploring that space efficiently. Meta-heuristics serve three main purposes: solving problems faster, solving large problems, and obtaining robust algorithms [26].

Most of the meta-heuristics mimic natural metaphors to solve complex optimization problems (e.g., evolution of species, annealing process, ant colony, particle swarm, immune system, bee colony, and wasp swarm) [26].

2.2.4 Deep Reinforcement Learning

According to [27], Reinforcement Learning (RL) is simultaneously a class of problems, a class of solution methods that work well on these problems, and the field that studies these problems together with their solution methods. The question in RL problems is how to learn from interaction to achieve a goal. The standard RL setup is depicted in Figure 2.8. It comprises an Agent that interacts with what is called Environment at each step of a discrete time series $t = 0, 1, 2, 3..., T$. At each time step the agent receives some representation of the Environment's state $\sigma_t \in \Sigma$, where Σ is the set of possible states, and selects some Action $a_t \in \mathcal{A}$ to perform, where $\mathcal{A}$ is the set of possible actions.

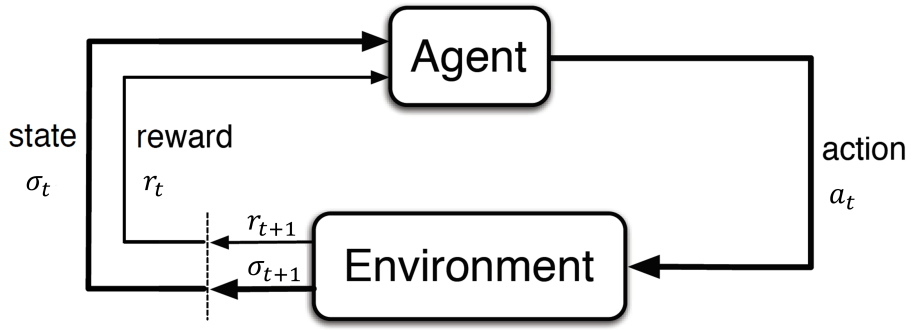


Figure 2.8: Standard RL setup. Source: [27].

The action performed by the Agent modifies its Environment accordingly, leads to a new State σ_{t+1} of the Agent, and generates a Reward $r_{t+1} \in \mathbb{R}$. The Reward is a special numerical value that formalizes the Agent's goal. In general, the agent tries to maximize the sum of discounted Rewards. This objective is called Return and can be formalized as $G_t = \sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1}$, where γ^k is the discount factor γ to the k-th power [27].

To select the Action to perform, the Agent relies on a Policy $\pi : \Sigma \times \mathcal{A} \to \mathbb{R}$ that maps each Action $a \in \mathcal{A}$ and State $\sigma \in \Sigma$ to the probability $\pi(a, \sigma)$ of taking Action a in State σ.

For a specific Policy, we can define the concept of State-value function,

$$v_\pi(t; \sigma) = \mathbb{E}_\pi \left[\sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} | \sigma_t = \sigma \right]$$

as the expected return when starting on state σ and following Policy π thereafter. An interesting property is defined from State-value functions since they allow the definition of a partial order between policies: $\pi \geqslant \pi'$ if and only if $v_\pi(\sigma) \geqslant v_{\pi'}(\sigma)\ \forall \sigma \in \Sigma$.

As a corollary, there is always at least one policy π^* that is better than or equal to all other policies. This Policy is called the optimal Policy and is associated with an optimal State-value function v_{π^*}[27]. However, differently from Optimal Control Theory, RL agents can only access the system transition probabilities (i.e., $p(\sigma'|\sigma, a)$) trough sampling. Hence, π^* and v_{π^*} can only be accessed by approximation [27].

In an essential way, RL problems are closed-loop problems because the learning system's actions influence its later inputs. A RL agent uses methods to discover which actions yield the most reward by trying them out [27]. RL is different from supervised learning since supervised learning involves learning from a training set of labelled examples provided by a knowledgeable external supervisor. This is an important kind of learning, but alone it is not adequate for learning from interaction. In interactive problems it is often impractical to obtain examples of desired behaviour that are both correct and representative of all the situations in which the agent has to act [27]. RL is also different from unsupervised learning, which is typically about trying to extract some structure in some collection of unlabelled data [27].

The book [27] classifies RL solution methods in two macro-categories: i) Tabular solution methods, and ii) Approximate solution methods. Tabular solution methods adopt the strategy of building a model of the environment. The goal of these methods is to build a table or array containing an approximation of the action-value function—the function that represents the expected sum of rewards that can be obtained by taking an action from a specific state, for each possible state-action pair. The advantage of these methods is that they can often find exact solutions, that is, they can often find exactly the optimal value function and the optimal policy to use to select actions. However, they can only be applied in situations where the number of states and actions is sufficiently small to be represented in a table. Examples of Tabular methods are Dynamic Programming, Monte Carlo Methods and Temporal Difference Learning [27].

Approximate solution methods do not use tables to approximate value functions. They use instead function approximation strategies (e.g., artificial neural networks) to approximate the state-value function and the policy. Optimality cannot be ensured but in return these methods can be applied effectively too much larger problems. Approximate solution methods can be divided in two categories: "On-policy" methods and "Off-policy" methods. On-policy methods are designed to learn an optimal policy while using it to select the actions made by the agent.

Off-policy methods, in contrast, train a policy different from the one used by the agent to select the decision [27]. DRL refers to the usage of Deep Learning to improve the function approximation needed to solve RL problems. Fig 2.9 presents how neural networks can be used in DRL.

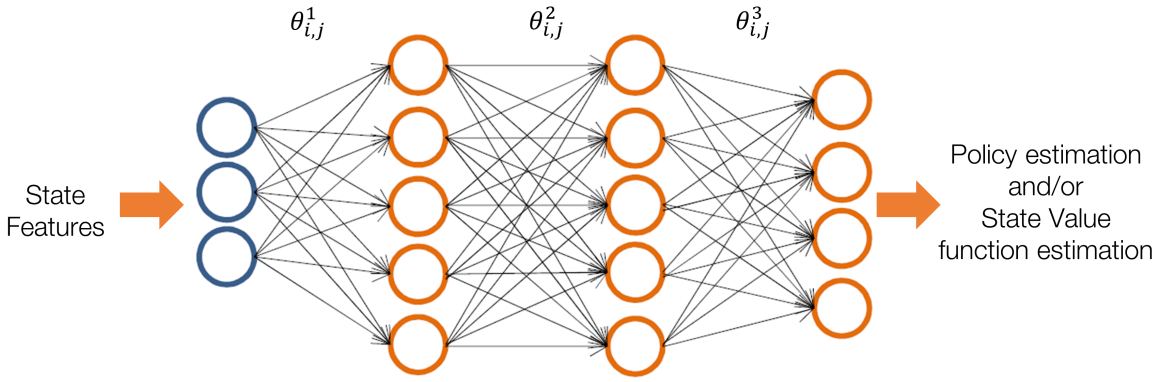


Figure 2.9: Usage of neural networks on DRL.

DRL frameworks often represent the Policy and other learned functions (i.e., State-value function) as Deep Neural Networks (DNNs) and develop specialized algorithms that perform well in this setting. As usual with neural networks, a Loss function $J(\theta)$ is defined to measure the error between the estimated functions and appropriate target values. Typically, a minimum of this loss function is computed using some iterative optimization algorithm (i.e., gradient descent) based on the gradients $\nabla J(\theta)$.

Conclusion

In this chapter, we introduced background knowledge. We first presented, the NFV pardigm and Network Slicing as useful concepts to understand the scope of this work. Then, in Section 2.2, we focused on the approaches and theoretical methods adopted during the research: Graph Theory, ILP, heuristics, meta-heuristics and DRL.

Chapter 3

State-of-the-art

Contents

Introduction

In this chapter, we present the state-of-the-art analysis considering comprehensive surveys [7], [28]–[35], and recent publications in main journals and conferences. This analysis has been performed according to the three domains and aspects as defined by the thesis scope introduced in Section 1.3: optimization models and algorithms for placement in virtual networks, presented in Section 3.1, Edge-enabled placement algorithms, presented in Section 3.2, and, finally, automated and reliable placement and management, presented in Section 3.3. We close the chapter with some concluding remarks.

3.1 Optimization models and algorithms for placement in virtual networks

In this section, we analyse existing works on optimization models and algorithms for placement in virtual networks. In Section 3.1.1, we present modeling aspects of the analyzed works focusing on the characterization of the optimization problems, constraints and optimization targets considered. In Section 3.1.2, we present algorithmic aspects of the proposed works, focusing on heuristic and approximation algorithms and discussing large-scale considerations for placement.

3.1.1 Placement optimization models

Optimization problems characterization

The NSP problem considered in this PhD thesis is intimately related with a broad family of resource allocation problems that have become critical for network operators with the emergence of network virtualization, NFV and Network Slicing, for instance: VNF/VM placement (VNF/VM-P), SFC placement (SFC-P), Virtual Network Embeding (VNE), VNF Placement and Chaining (VNF-PC) and VNF Fowarding Graph Embeding (VNF-FGE).

References	Problem Solved				Modeling Strategy	
	SFC-P/VNE	VNF-FGE/PC	VNF/VM-P	OT	IP	OT
[36]–[44]	✓					
[45]–[62]	✓				✓	
[63]	✓					✓
[64]	✓			✓	✓	
[65], [66]		✓				
[67]–[69]		✓			✓	
[70]		✓				✓
[71]		✓		✓		
[72]		✓		✓	✓	
[73]			✓			
[74], [75]			✓		✓	
[76]–[80]			✓			✓
[81]			✓	✓		✓
[82]–[85]				✓		
[86]–[92]				✓	✓	
[93]–[95]				✓		✓

Service Chain Placement/Virtual Network Embedding (**SFC/VNE**), VNF Fowarding Graph Embeding/Placement and Chaining (**VNF-FGE/PC**), VNF/Virtual Machine Placement (**VNF/VM-P**), Other (**OT**), Integer Programming (**IP**).

Table 3.1: Modeling aspects: problem considered and modeling strategy.

References	Technical constraints			Optimization targets		
	RU	Q	UL	RU	Q	C/R
[36]–[39], [45]–[50], [65], [67], [71], [75], [82], [89], [90]	✓			✓		
[63], [70], [76], [94]	✓			✓		
[73], [78]	✓			✓	✓	
[80], [91]	✓				✓	
[40]–[42], [57], [58], [92], [95]	✓					✓
[43], [44], [51]–[55], [68], [77], [83], [86], [87]	✓	✓		✓		
[69]	✓	✓		✓	✓	
[62], [74], [81], [93]	✓	✓			✓	
[56], [66], [79], [84], [88]	✓	✓				✓
[85]	✓	✓		✓		✓
[59], [60], [64], [72]	✓	✓	✓	✓		
[61]	✓		✓	✓		

Resource Utilization (**RU**), Quality of Service (**Q**), User-location (**UL**), Cost/Revenue (**C/R**).

Table 3.2: Modeling aspects: technical constraints and optimization targets.

Table 3.1 shows that most of the related works analyzed here deal with these problems—specially with SFC-P and VNE. Other optimization problems have also been considered (represented by the OT column of Table 3.1).

All theses problems have in common the objective of optimizing allocation of virtual resources available in a physical substrate infrastructure in order to accommodate user requests subject to a set of constraints on the amount of available resources (e.g., CPU units) and sometimes also to constraints related to service related metrics (e.g., latency, service availability).

The less complex of the problems considered is VNF/VM Placement [31]. Generally speaking, this problem consists on the choice of a placement server or node for deploying an unique virtual node/VNF ignoring the VNF chaining step, that is, the interconnection of the placed VNF with other VNFs and the steering of traffic between them. An exception is [82], which considers container placement optimization with inter-container traffic routing constraints.

The SFC-P problem is generally more complex than VNF/VM-P since it involves placing a SFC, i.e., a series of interconnected VNFs [30] which compose a network service. In this problem, the placement request is often modeled as a path graph topology with node requirements (e.g., CPU, RAM) and link requirements (e.g., bandwidth, latency). This graph may be directed or undirected, and constraints on the order in which VNFs must be placed and spanned by traffic may also apply. Ingress and egress nodes, i.e., the nodes from which traffic originates and those to which it is directed, respectively, may also be considered.

The VNE problem [29], [32] concerns the embedding of a virtual network into a physical network. This problem has been widely studied and authors have considered a huge set of parameters, constraints, objective functions and decision variables [28]. In general, this is a broader problem since virtual and physical networks are often modeled as undirected graphs and can be represented as generic graph topologies, including topologies with cycles. VNF-FGE can be seen as a generalization of the SFC-P and VNE since it tackles more complex placement requests in terms of topology and considers allocation of multiple resources simultaneously [7]. This problem is also very similar to the VNF placement and chaining problem [65] and is $\mathcal{NP}$-hard [96].

Other alternatives associated with these classical problems have been studied. An interesting example is the slice embedding problem, described in [90]. The authors define the slice embedding problem considering three steps. First, resource discovery, that is, proper monitoring of the state of physical resources and discovery of a set of elements that can be used to respond to a given user request based on a set of parameters given as input. Second, virtual network mapping, that is, matching users requests with the available resources. Third, allocation, meaning the assignment of the resources that match the user's requests and scheduling their usage. They propose a high-level model for these problems in distributed service architectures, that is, large-scale distributed system whose architecture is based on a service paradigm, e.g., data center (DC) based systems, cloud computing, grid computing).

Another important alternative is VNF-FG Composition (VNF-FGC) [71], [72]. The VNF-FGC problem is another optimization problem in the family of NFV resource allocation problems. It consists on defining a suitable VNF-FG for offering a specific service by taking as input a set of VNFs and order constraints for them [7]. A Network Slice Design problem is treated by [86], [88] and is characterized by joint consideration of composition and placement decisions in the context of 5G slicing.

The slice admission control problem consists of, deploying network slices over the physical infrastructure, as well as the admission of the user service requests into the slices after they have been deployed [95]. An interesting approach to the problem proposed on [94] is considering that when a slice request cannot be accepted in some moment due to lack of resources, it waits in one of multiple queues for a next opportunity.

Out of the scope of the contributions of this PhD thesis are resource reallocation problems, concerned with re-optimization of an initial placement decision. An example is VNF scaling, the problem of treating a VNF scaling request concerning a virtual network/slice already placed [91]. In [91], both horizontal (scale out/in) and vertical (scale up/down) scaling are considered. Horizontal scaling is also treated by [40], in which the authors consider VNF migration and VL splitting.

In [93], the authors consider the problem of migrating VNFs from one Edge node to another according to user mobility in a way that minimizes migration cost (i.e., distance and number of migrations). Authors of [89] also deal with horizontal scaling. They propose to extend already deployed SFCs by inserting new nodes and links on the SFC without compromising the nodes and links already deployed.

From Table 3.1, we can see that SFC-P and VNE problems are the most studied ones followed by the OT category which includes multiple problems, some not in the scope of this thesis. In third place comme VNF/VM-P and finally VNF-FGE/PC. We rely mainly on SFC-P, VNE and VNF-FGE in order to define our proposed NSP problem.

Optimization modeling strategies

There are several papers studying the modeling of VNF/VM-P, SFC-P, VNE, VNF-FGE, etc and the majority of works analyzed uses some kind of exact mathematical model to formalize the problem in terms of input data, decision variables and problem constraints (when the problem is constrained). These mathematical models are useful since they allow to unambiguously define a model for the problem and to obtain proven optimal solutions in small scale scenarios. We divide the considered modeling strategies in two categories as shown in Table 3.1: Integer Programming (IP) and Others (OT).

We can observe in Table 3.1 that most of the analyzed works fall in the IP category. The IP category includes ILP [45] and Quadratic Programming [51]. Generally speaking, these strategies are mainly concerned with *offline* optimization—in which complete knowledge about the slice requests and network capacities is assumed—considering a certain optimization objective subject to constraints. In the case of ILP both optimization objective and constraints are linear which is not the same in the case of quadratic programming (e.g., for instance in [51] quadratic constraints are assumed).

Even if quadratic programming allows to propose more refined models, the quadratic constraints or objective

References	Problem solving strategy			
	E	H/A	ML	ML/H
[45], [51], [55], [68], [77]–[79], [86]–[88], [90], [94], [95]	✓			
[36]–[38], [40], [43], [47]–[50], [52]–[54], [56], [57], [59]–[65], [67], [70]–[74], [76], [80], [82], [89], [91], [93]		✓		
[39], [41], [42], [44], [46], [66], [75], [81], [83]–[85], [92]			✓	
[58], [69]				✓

Exact (**E**), Heuristics/Approximation algorithms (**H**/**A**), Machine Learning (**ML**), ML/Heuristic hybrid (**ML**/**H**).

Table 3.3: Algorithmic aspects: problem solving strategies.

function renders it impossible to solve the problem using a largely adopted linear solvers (e.g., CPLEX).

The OT category includes Game Theory [79], Markov Decision Process [93], Matching Theory [63] and Queuing theory [77]. In general these works are concerned with *online* optimization and assume probabilistic behaviour of parameters of the system (e.g., requests arrivals and requirements follow some probability law) instead of deterministic behaviour considered by ILP-based solutions (i.e., requests arrival dates and resource requirements are fixed).

As a first step, for modeling the problem, we leverage on ILP as a way to efficiently model the proposed NSP problem as an offline optimization problem. We then move to online optimization and include probabilistic models in our work to propose simpler algorithms to efficiently solve the problem in more realistic scenarios.

Optimization targets and constraints

As exhibited in Table 3.2, existing models cope with different optimization targets and constraints. We divide technical constraints in three categories: resource utilization (RU), Quality-of-Service (Q) and User-location (UL). We also divide optimization targets in three categories: resource utilization (RU), Quality-of-Service (Q), and Cost/Revenue (C/R).

User location constraints are constraints on the geographical location of the end-user of the virtualized network. RU, Q and C/R categories comprise a set of constraints or optimization targets as described in Figures 3.1 and 3.2 respectively.

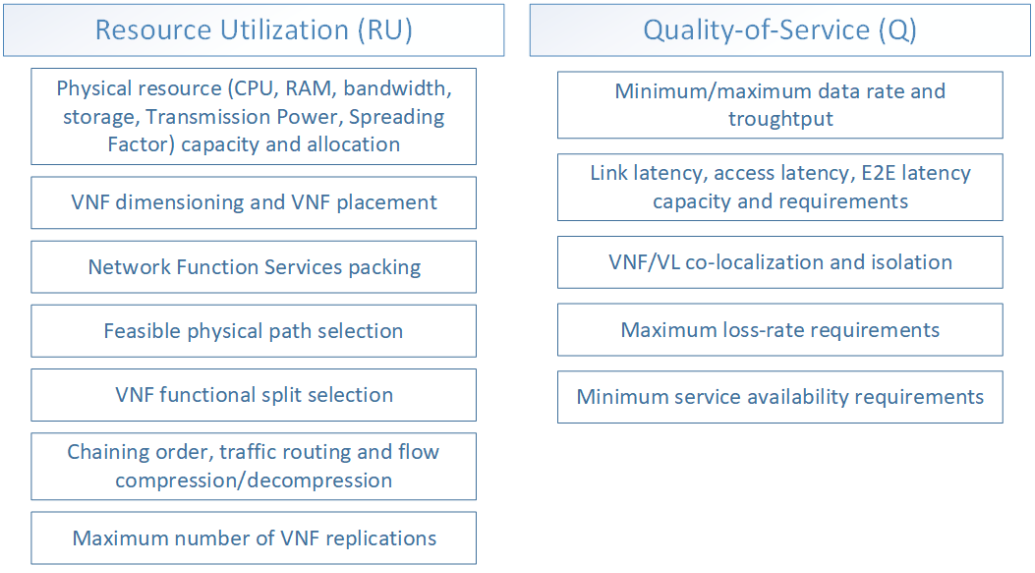


Figure 3.1: Description of constraints categories.

Table 3.2 shows that most works have focused on different aspects related to resource utilization without always integrating QoS constraints. In the remainder of this section, we focus on the most representative works in terms of the optimization objectives and the constraints they consider.

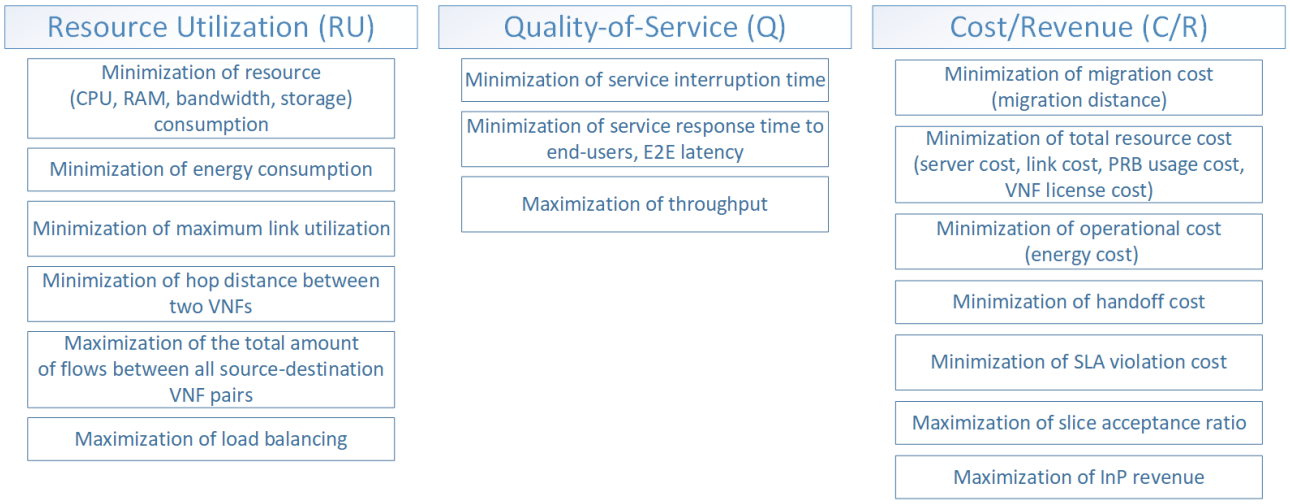


Figure 3.2: Description of objectives targets categories.

For instance, in reference [45] indicated on the first line of Table 3.2, the authors focus on load-balancing. They propose an ILP strategy to the problem of SFC-P and explore the potential of VNFs replications to help to balance the load generated in the network.

Traffic dependent VNF resource requirements are considered in [47], which also study the the SFC-P problem. They adopt a flow maximization ILP modeling with a set of fixed origin and destination nodes for each service chain and consider that the amount of the resources consumed by a VNF is not fixed a priory but is dependent on the amount of traffic it steers in a given moment. However, the authors neglect the multi-dimensional aspect of the resources a node can have and also QoS aspects.

Similarly, reference [57] also adopts a flow maximization ILP modeling of the SFC-P problem in which traffic demand for each slice is divided into different flows and each flow is associated with a source node and a destination node. A particularity of the proposed problem is VNF ordering constraints defined for the specified flows. The authors propose an ILP model which jointly optimizes the profit of service providers and the profit of network operators and the interplay of two such metrics. QoS related constraints are not considered.

Maybe the most complete set of constraints is considered on [86]—shown in line 10 of Table 3.2—in which the authors propose an ILP-model to Network Slice Design problem. The proposed problem consolidate various constraints emerging from 5G network slicing specifications such as functional split selection, service isolation level, and different levels of mapping for creating a complete 5G virtual network. For instance, the authors consider the mapping of network function services to network functions and the mapping of network functions to slices and slice sub-nets. Out of ILP-based strategies, the authors of [63] propose a perfect 2-matching exact approach for VNF placement but only focused on CPU and bandwidth allocation. In the context of the SFC scaling problem the authors of [89] propose an interesting Steiner-based ILP formulation and approximation algorithm, but the authors neglect QoS constraints which are of utmost importance in this kind of reconfiguration problem.

In this thesis, we focus on specific parameters for Edge-enabled network slicing. Such parameters are translated into problem constraints: E2E latency due to its critical importance in Edge-enabled Network Slicing (e.g., for ultra-low latency communications in 5G use-cases [97]) and user-location. These aspects are discussed in more details in Section 3.2.

3.1.2 Placement optimization algorithms

Placement algorithms characterization

We classify in Table 3.3 the different analyzed works according to the type of problem solving strategy. Some of the proposed works adopt exact problem solving strategies mainly using mathematical models for offline optimization. For instance, [86] proposed a very complete problem modeling using ILP. In [45], an ILP strategy is used to solve VNF-FGE allowing VNF replications.

References	Algorithm characteristics				
	ON	DY	MA/D	MO	RO
[47]–[50], [54], [55], [61], [62], [74], [79], [86], [93] [36], [37], [39], [42]–[44], [51], [58]–[60], [63]–[65], [69], [70], [72], [82], [89], [91], [95]	✓				
[41], [90]			✓		
[45], [68], [73], [78]				✓	
[56], [88]					✓
[38]			✓	✓	
[66], [84]	✓		✓		
[83]	✓		✓	✓	
[46], [67], [71]	✓			✓	
[52], [53]	✓			✓	✓
[40], [57], [75], [80], [81], [87], [94]	✓	✓			
[77], [85]	✓	✓		✓	
[76], [92]	✓	✓	✓		

Online (**ON**), Dynamic (**DY**), Multi-agent/Decentralized (**MA/D**), Multi-objective (**MO**), Robustness (**RO**).

Table 3.4: Algorithmic aspects: algorithm characteristics.

Out of ILP-based strategies, the authors of [63] propose a perfect 2-matching exact approach. In [93] the authors model the VNF migration problem as a dynamic sequential decision making problem using the framework of Markov Decision Process. This framework is useful to find optimal service migration policies but it to complex to be used in practice. The authors propose then an approximation scheme to calculate the optimized policies in a faster way. Reference [80] propose a queuing-based model to jointly optimize VNF placement, resource assignment, and traffic routing.

To address the scalability issues due to the ever-growing complexity of exact approaches like ILP, most of the analyzed works use heuristic or approximation algorithms approaches as problem solving strategies. Reference [89] is an example of the application of approximation algorithms in the context of SFC scaling. In [60], the authors propose two novel heuristic algorithms based on the transformation of the VNE problem with location constraints in a minimum cost maximum clique problem with the construction of a compatibility graph. The authors provide a formal proof of the problem $\mathcal{NP}$-completeness and the proposed algorithms solve the node and link mapping phases of the VNE problem jointly. The drawback of the proposed approach is its prepossessing stage, which increases the overhead of the proposed solution and reduces its scalability.

Another example of graph based heuristic is the multi-stage graph method combined with a matrix-based optimization approach proposed in [63] to solve the SFC-P problem. Some recent approaches use machine learning techniques or hybrid machine learning/heuristic strategies as problem solving strategies. The advantages and limits of such approaches are discussed in detail in Section 3.3.

We classify the analyzed works in Table 3.4 according to five algorithm characteristics: Online (ON), Dynamic (DY), Multi-agent/Descentralized (MA/D), Multi-objective (MO), Robustness (RO). Robustness of placement algorithm is an important aspect of this PhD thesis and the works considering this aspect will be introduced in Section 3.3.

Table 3.4, shows that most of existing placement algorithms are static—as opposed to dynamic approaches in which a (re)optimization of the already performed placement is possible—and use centralized strategies, mostly online algorithms, where placement requests arrive dynamically and are not known in advance as opposed to offline algorithms. We have identified one ILP-based strategy ([90]), two heuristic approaches ([38], [76]), two federated learning approaches ([83], [84]) and three multi-agent learning approaches ([41], [66], [92]) which are decentralized, where the decision is distributed among by multiple agents.

Some works considered multi-objective optimization, mainly focusing on genetic algorithms. For instance, in [71], the authors tackle two problems. They first propose two-step strategy for VNF-FGC. Then, they propose a multi-objective optimization algorithm for VNF placement tackling the minimization of the total bandwidth consumption and of the maximum link utilization simultaneously. Four multi-objective genetic algorithms are proposed based on the combination of two state-of-the-art approaches—multiple objective genetic algorithm and improved non-dominated sorting genetic algorithm—with two different strategies population initialization—random and greedy. With the greedy strategy, nodes that have more residual resources will have higher priority during the VNF-to-node mapping, and links with shorter length are prioritized as well. This is an interesting comparative study but QoS aspects are absent. Also, as the majority of the proposed algorithms, these are not tested in large-scale scenarios.

References	Network scale					Test environment	
	PSN			NSPR		SD	TI
	S	M	L	S	L		
[43], [49], [90], [93]							
[71]	✓					✓	
[38]–[42], [45], [48], [51]–[53], [56], [57], [59]–[61], [64], [66], [68], [73], [77]–[79], [82], [84], [85], [87], [88], [92], [94], [95]	✓			✓		✓	
[75], [81]	✓			✓			✓
[37], [44], [46], [47], [50], [54], [55], [58], [62], [63], [67], [69], [70], [72], [74], [76], [80], [83], [86], [91]		✓		✓		✓	
[36], [65]			✓	✓		✓	
[89]			✓		✓	✓	

Small (**S**), Medium (**M**), Large (**L**), Simulated data (**SD**), Testbed implementation (**TI**).

Table 3.5: Algorithmic aspects: network scale and test environment.

Another reference considering multi-objective genetic algorithms is [37]. The authors propose to use a non-dominated sorting genetic algorithm II to address SFC-P. Again, large-scale and QoS considerations are absent. Reference [73] focus on virtual machine placement. The authors also propose a multi-objective genetic algorithm for optimizing load balancing and resource usage under memory and CPU capacity constraints.

Large-scale considerations for slice placement

Table 3.5 shows how existing algorithms support large-scale for slice placement. Only small and medium scale networks seem to be mainly considered and none of the works considering medium or large-scale perform testbed implementation, the majority of the works being based on simulations done using the GT-ITM tool [98] or the SNDLib [99].

In [78], the authors propose a multi-objective three layer binary tree algorithm to the virtual machine placement problem. They propose to jointly maximize the machine and bandwidth elasticity, that is, improving how well the data-center can satisfy the growth of the input VMs resource demands under both the limitations of physical machines capacities and links capacities. The proposed approach is only tested in small scale environments and considering only link capacity constraints.

In [62], the authors propose and ILP formulation and an Affinity Based heuristic method to solve the SFC-P problem. The authors focus on inter-cloud traffic minimization and response time in a multi-cloud subject to important constraints such as total deployment costs, link delays and computational delays. The proposed approach includes a big set of constraints, but the model is low level, with a lot of parameters and constraints, what compromise its use in large-scale scenarios.

The authors of [54] propose a greedy heuristic to increase solution efficiency of a previously proposed ILP, but they do not evaluate it in large-scale network scenarios. In [76], the authors propose a Markov Approximation strategy and a matching algorithm to solve the VNF placement problem. The problem formulation with quadratic constraints is complex and hard to solve in large-scale. In [100], distributed optimization strategies are used to solve VNF-PC. The authors treat the placement of E2E services in a PSN composed of multiple administrative domains but without including important QoS criteria like E2E latency.

In [70], the authors consider the VNF Placement and Chaining problem and propose an exact solution based on Perfect 2-Factor theory. To increase scalability, the authors propose a multi-stage graph heuristic algorithm to find near-optimal solutions in few seconds for large instances. The exact algorithm can efficiently solve the problem until 3 VNFs on the service. The authors do not evaluate the proposed heuristic in large-scale networks to access its scalability. Only medium scale networks are considered.

In [79] game theory is used to solve a VNF placement problem. This work have the following limitations: it considers only placement of Virtualized Evolved Packet Core (vEPC), it is not scalable, and it considers a small number of tracking areas and cloud networks. In [74], the authors propose an iterative linear relaxation ILP-based strategy VNF placement problem. But, the problem is solved only in small scale and the proposed approach is limited to virtualized Evolved Packet Core (EPC) placement. The same authors of [45], propose in [50] a Genetic Algorithm and a Random Fit Placement algorithm for the VNF placement problem in order to efficiently solve the previously proposed ILP for SFC-P with the possibility of VNF replications. But again, no QoS elements are considered.

Among the works that try to evaluate their algorithms in large-scale scenarios, we consider, [67], where the authors propose an ILP model to solve the VNF-FGE problem and use the strategy they call "boosted ILP".

The proposed strategy is based on a selection of only a subset of paths of the network to consider as candidates to map the VLs. This helps to reduce the solution space and, hence, to accelerate the computation of an approximate solution for the problem. The considered optimization objective is to minimize power consumption. An interesting aspect of their model is that they allow VNFs to be shared between different VNF-FGs. An important drawback is the assumption that all the paths in the network are known in advance. As the enumeration of all paths in a network is an $\mathcal{NP}$-hard problem, this formulation is difficulty directly applicable in practice. Besides, QoS constraints are not explicitly treated.

In [36], the authors consider the SFC-P problem which is formulated using a decision tree approach. Each node in the tree corresponds to a virtual resource embedding and each tree branch to the mapping of a client request on some physical candidate. The authors propose a Monte Carlo Tree Search based approach which is tested in large-scale scenarios, but neglected QoS related aspects.

Reference [65] proposes an online Eigen decomposition approach to solve VNF-FGE. The authors show the scalability of their approach via simulations taking into account PSNs with up to 5000 nodes. However, QoS constraints (e.g., latency, delay) are not considered in their work and no optimality comparison is performed.

We evaluate our proposed approaches considering a large-scale PSN and small scale NSPRs and considering QoS constraints. Hence, there is clearly still a lack of approaches considering the large-scale aspect for both the PSN and the NSPR.

3.2 Edge-enabled placement algorithms

We have seen in Section 3.1 that existing placement models and algorithms cope with different optimization targets and constraints. Most of them use online heuristic approaches. However, they are focused on the placement problem, often with a resource consumption reduction target, rather than a complete orchestration mechanism with QoS constraints. We focus in this section on the papers considering Edge-specific constraints. We introduce in Section 3.2.1 works considering latency. Then, in Section 3.2.2 we introduce works considering another parameter that is user-location.

3.2.1 Latency-aware placement

Mainly, two kinds of approaches are currently used to deal with latency. The first one considers latency in the optimization target. In [80], for instance, average E2E latency is studied as an optimization criteria. In [62], the authors focus on minimization of response time and inter-cloud traffic in a multi-cloud subject to important constraints such as total deployment costs, link delays and computational delays.

The second kind of approach deals with latency as a strict constraint, that is, a constraint that absolutely needs to be satisfied. For instance, the authors of [51] propose a Mixed Integer Quadratically Constrained Programming model to the SFC-P problem to determine the optimal VNF placement minimizing resource consumption while providing specific latency (i.e., E2E delay) and avoiding violation of SLA.

An interesting point of the proposed approach is that it does not consider a fixed packet processing delay but a delay dependent of the amount of resources allocated. The proposed packet processing delay function is approximated using piecewise linearization. But, the use of quadratic constraints increases the complexity of the proposed model.

In [54], the authors formulate the SFC-P problem as an ILP focusing on QoS driven placement and reduction of expenditures. The authors consider an objective function with multiple costs including VNF licensing cost subject to multiple QoS related constraints such as minimum availability of service, the maximum E2E latency and the minimum data rate. Reference [87] proposes a dynamic Radio Access Network (RAN) slicing based scheme based on ILP taking into account multiple, baseband resources, fronthaul and backhaul capacities, quality of service metrics such as latency, and also interference aspects.

In [68], an ILP is proposed for the VNF chain routing optimization, which is based on a definition of a VNF chain request as a flow between an origin and destination. The ILP is designed to take into account specific aspects of VNF chain routing optimization such as the fact that the bit rate of each flow demand can change along a VNF chain (flow compression/ decompression), the fact that the VNF processing latency and computing load can be a function of the demands traffic, and finally, the fact that VNFs can be shared among demands.

Another strategy considering latency constraints is [77], the authors propose a multi-objective strategy to VNF placement and provisioning optimization strategies over an edge-central carrier cloud infrastructure taking into account QoS requirements (i.e., response time, latency constraints and real-time requirements) and using queuing models and ILP. The authors seek to minimize a weighted sum of the maximum utilization of CPU resources in the edge and cloud and to minimize QoS violation in terms of real time requirements of each VNF. The approach considers dynamic aspects of the problem, but focuses on single VNF placement.

Since meeting the latency requirements are not ensured when it is treated within an optimization target rather than a strict constraint, considering latency as a strict constraint is more accurate. However, the existing works that consider E2E latency as a strict constraint (e.g., [51]) are incomplete for Network Slice orchestration. In these works, E2E latency is calculated between a source and a destination VNF. In Network Slicing, the service users must be considered as an end-point in the E2E latency calculation. Hence, taking user location into consideration is of utmost importance.

3.2.2 Location-aware placement

Edge-specific constraints are usually related to link latency, access latency and E2E latency and directly linked to user location. The authors of [59] were the first ones to propose an approach to solving VNE taking into account user location. They propose an approach based on multi-commodity flow model using ILP and relying on linear relaxation methods to solve the problem efficiently. The proposed algorithm is based on the transformation of the problem via the construction of an augmented substrate network graph that includes virtual nodes. The proposed algorithms solve the node and link mapping phases of the VNE problem jointly taking into account geographic location constraints for each virtual node.

The algorithms of [59] are based on the assumption that a preferred placement location is known for each Virtual Node of the Virtual Networks to be placed. This assumption is reused in [60], where the authors propose the transformation of the location-constrained VNE into a minimum cost maximum clique problem. Their work is original but the assumption that two virtual nodes of the same Virtual Network cannot be embedded on the same machine is structuring for the proposed algorithms. Hence their applicability is limited in the context of Network Slicing.

The assumption of [59] was also recently further exploited in VNF-FGE. In [72] the authors propose an ILP and heuristic algorithms for solving the Service Embedding and Chain Composition problem. In spite of the originality of their work, the authors consider trees as the only possible topologies for VNF-FG. Hence, the proposed algorithms cannot deal with cyclic VNF-FGs, which limits its applicability to Network Slicing.

We identified only two works which consider user location without considering the preferred location assumption introduced by [59]. For instance, the authors of [61] deal with user location constraints in the context of RAN slicing. They propose an ILP formulation and heuristics the SFC-P problem in RAN. The study developed in [64] proposes an ILP and a heuristic to solve a double problem. They consider both user association and SFC-P problems taking user location into account. This work is original, but it presents some limitations: as it solves SFC-P problem, it considers that the requests to place can only be path graphs. Also, it models individually each User Equipment that is connected to an SFC instead of grouping them which brings scalability issues due to drastic increase in the number of binary variables.

The preferred location assumption regarded in most of the above works have the advantage of reducing the problem complexity. However, to the best of our knowledge, the accurate determination of the preferred placement location for each Virtual Node while solving VNE or VNF-FGE is not straightforward and the authors do not provide any algorithm to calculate it. Hence in our proposals, we do not consider this assumption and we propose an E2E latency model that look at the user location as an end-point of the Network Slice. In addition, to be more generic, we do not make any assumption on the embedding location of two nodes of the same Network Slice (unlike [60]).

3.3 Automated and reliable placement and management

The automation of orchestration of resources and services is a way of enabling agile management of network functions and services. This provides new opportunities to efficiently handle more services while reducing costs in terms of resource consumption through mutualization for example.

According to the ETSI, efforts are needed to ensure automation support. Service usage should drive on-demand resource requirements, triggering feedback from the systems that the management functions analyze and make changes enabling the infrastructure to provide the needed resources and services [3]. This kind of automation is hard to obtain using classical optimization techniques such as ILP and heuristics due to the lack of flexibility of such techniques.

Also, even if from an operational perspective, heuristic approaches are more suitable than ILP as they yield faster placement results which is very important for operational networks because traffic conditions are fluctuating and placement response time is an important performance indicator in the customer relationship, they have an important drawback: they give sub-optimal solutions. To address this issue, ML offers a corpus of methods able to overcome the convergence issues of ILPs while being more accurate than heuristics and has received recently used for management automation. We review in this section recent studies on automated and reliable placement and management. We analyse existing works along two lines: i) ML models algorithms for placement optimization and automation (Section 3.3.1), and ii) Reliable methods for placement and management (Section 3.3.2).

3.3.1 ML models and algorithms for placement optimization and automation

Unsupervised and supervised learning models and algorithms

Few papers adopt classical ML approaches such as supervised or unsupervised learning to optimize placement. An example is Bunyakitanon et. al. [81] which targets two problems: VNF performance prediction and VNF placement. Their solution relies on the forecasting of the Total Response Time for a VNF running an E2E service. The forecasting is done using multiple regression models (Lasso, KNN-Regression, SVR), as well as Decision Trees and Random Forests. The solution is tested on a realistic small scale testbed and presents good results. However, no formal model is given making it difficult to understand the proposed solution.

In [92], the authors propose a unsupervised learning approach using primal-dual method. Deep Learning is used for obtaining optimal network node configurations based on distributed observations. The authors introduce a generic framework to solve distributed non-convex constrained optimizations in wireless networks where multiple computing nodes, interconnected via back-haul links, desire to determine an efficient assignment of their resources based on local observations.

The distributed approach is based on partial sharing of data between different nodes using a quantizer neural network that encodes the data to be transferred via a limited capacity backhaul/fronthaul link. However, here the authors study a reconfiguration and more generic resource allocation problem that the one we study in this thesis.

Reinforcement learning models and algorithms

The most relevant ML-based approaches for placement optimization are RL or DRL based solutions. In Table 3.6, we classify these works according to the RL setup elements of each solution (i.e., state, action, reward). In Table 3.7 we classify these works according to modeling and algorithmic criteria (i.e., network scale, training algorithm, use of convolution, use of heuristics). We describe hereafter the analysis of these different aspects.

State representation: The state representation is almost similar in all analyzed solutions, except for [83], in which the authors concentrate in slicing for Internet-of-Things networks and consider energy efficiency and Internet-of-Things devices associated to the slice as state features.

Some works consider only one unique resource related feature. Is the case of [84], which focuses on mobility management for Radio Access Network Slicing and considers only bandwidth features, and [75] which considers VNF placement and only CPU related features. In [39], [41], [42], [46], [58], more than one resource-related feature is used to represent the state: i) the number of resources available on physical nodes (CPU and RAM) and links (bandwidth) and ii) the number of these resources required by the VNFs and VLs of the network slice to be placed.

Latency-related features are also considered in [44], [66], [69], [85]. However, adding such features to the state brings additional complexity to their models. In particular, the model in [85] requires additional chaining information and performance indexes for the VNFs. In [66], the model requires an explicit VL representation. We adopt a resource oriented state representation. A simpler model considering latency remains an open issue.

Action representation: In [39], [42], [44], [46], [58], [84], the problem is modeled by considering finite action spaces. In [84], the action refers to electing both a target base station and network slice when a hand-off occurs. The action in [39], [44], [46] is the index of the physical node in which to place a specific VNF of the slice. This representation of the action requires breaking the process of placing one slice in a sequence of placement actions and has the advantage of reducing the size of the action space to the number of physical nodes. In [58], the action is represented as a binary variable used to modify or accept the placement of a specific VNF of the slice previously computed by a heuristic. The action in [42] is either to mark a VNF to be placed on a physical node or to place a VNF on the previously marked physical node. The placement of the entire slice is then iteratively constructed and the algorithm stops when all the VNFs of the slice are placed or when a maximal number of iterations is reached.

In [41], [66], [69], [75], [83], [85], infinite action spaces are adopted, i.e., the actions are real numbers. In [85] the action is defined as an instruction to a scheduler to adjust the resources allocated to a VNF by a certain percentage whereas in [66], the action is the placement price returned to the client. In [83], the action is defined by a transmission power and spreading factor setting. In [75], the action is a real number describing how much to increase or decrease the next surveillance epoch. In [41] the action consists of selecting Internet-of-Things devices to be served, the power allocated to each device as the data rate and the path selected for each service request.

Reference	State Features					Reward Function			
	EE/I	CPU	ST	BW	LAT	AR	LB	C/R	SLA
[83]	✓							✓	
[75]		✓						✓	
[84]				✓				✓	
[85]		✓	✓	✓	✓				✓
[39], [41], [42], [58]		✓		✓				✓	
[66]		✓		✓	✓			✓	
[44]		✓		✓	✓	✓		✓	
[69]		✓		✓	✓	✓			
[46]		✓		✓		✓	✓	✓	

Energy Efficiency & IoT Devices Associated with the Slice (**EE/I**), Storage (**ST**), Bandwidth (**BW**),Latency (**LAT**), Acceptance Ratio (**AR**), Load Balancing (**LB**), Cost/Revenue (**C/R**), Service Level Agreement violations (**SLA**).

Table 3.6: RL setup elements of DRL-based placement algorithms.

In [69], the action is represented by two sets of weights: i) the placement priority of each VNF in the slice on each physical node and ii) the placement priority of each VL in the slice on each physical link. To reduce complexity, we adopt a finite action space represented as in [46].

Reward function: Most of analyzed solutions adopt placement cost or revenue in their reward function [39], [41], [42], [44], [46], [58], [66], [75], [83], [84]. Cost and revenue are mainly calculated in terms of resource consumption. Some solutions adopt a reward associated to acceptance ratio [44], [46], [69]. A penalty for SLA violations is considered only in [85]. The most complete reward function is the one proposed in [46] combining acceptance ratio and placement cost with load balance. We leverage on [46] reward function criteria but propose a formulation that reduces bias during training.

Use of convolutions: Most of the analyzed papers employ a Convolutional Neural Network to perform automatic feature extraction except the approach proposed in [44], which uses regular DNN. Classical Convolutional Neural Networks are limited by the fact that they only work on Euclidean objects (e.g., images, grids).

DRL algorithms for NSP built upon this technique have reduced real-life applicability because they cannot work on unstructured networks. To overcome this issue, GCN have been used in [101], [102] in order to work on arbitrarily structured graphs. Based on spectral graph theory, they define the convolution operation of graphs using Chebyshev polynomials with trainable parameters that are learned in a neural network model. GCNs are used to automatically extract advanced features from graphs allowing to represent its semantics in a very effective way as it encodes and accumulates features of local neighbours. GCNs are used in [46] to automatically extract features from the physical network when solving a VNE problem. However, GCNs can only be applied for representing homogeneous graphs that are graphs in which all nodes and all links are of the same type.

To deal with this limitation authors of [103] have developed a new technique called Relational Graph Convolutional Network (RGCN). The goal of RGCN is to extract features in heterogeneous graphs, where we have more than one type of node and link.

This propriety is essential to learn appropriate representations of large-scale relational data graphs [103]. As GCNs, RGCNs accumulate transformed feature vectors of neighbouring nodes. However, RGCNs perform relation-specific transformations i.e., depending on type and direction of an edge. RGCN is used by [58] to automatically learn how to improve the quality of an initial placement computed by a heuristic. We also use the power of GCN in our proposal.

Training algorithms: As shown in Table 3.7, we classify the training algorithms proposed in the papers analyzed in Policy-based and Value-based. Policy-based methods build an explicit representation for the policy and keep it in memory during learning. Value-based methods do not store any explicit policy and only store a value function which is used to derive the policy implicitly (selecting the the action with the best action-value).

One Value-based method which was extensively used is Deep Q-learning. In [83], the authors concentrate on slicing for Internet-of-Things networks and propose a Multi-Agent deep Q-learning method for federated and dynamic network slice management and resource allocation for differentiated QoS services. Their approach considers optimization of resource allocation in terms of Transmission Power and Spreading Factor according to the slices QoS requirements. Simulation results show that the approach improves energy consumption, delay and throughput when compared with other traditional approaches.

In [41], the authors employ Multi-Agent RL to solve the SFC placement problem for Internet-of-Things connected devices. Their proposed solution is based on multiple Deep Q-Network agents that map the SFCs to the substrate network. They assume a priory knowledge of the paths between two nodes in the network.

In [84], propose a multi-agent RL-based Smart handoff policy with data sharing. The proposed policy aims to reduce handoff cost while maintaining user QoS requirements in RAN slicing. It consists in a distributed version of the Q-learning algorithm that is more efficient than centralized Q-learning due to smaller action space. Numerical results show that in the proposed distributed Q-learning approach can significantly reduce the handoff cost when compared with traditional handoff policies (without learning). Again, no real implementation of the proposed approach is provided.

The majority of the analyzed solutions use Policy-based methods, mainly focusing on Policy Gradient training algorithms. An algorithm that is often used is Deep Deterministic Policy Gradient. For instance, in [66], Quang et. al., propose a multi-domain non cooperative VNF-FGE DRL based approach. They consider a PSN with multiple domains. One learning agent is assigned to each domain and it is trained to perform VNF-FGE in this domain using embedding cost minimization as its optimization criterion. To train their agents they adopt Deep Deterministic Policy Gradient. Their proposal is innovative, since they are one of the first works to use multi-agent deep RL to solve the VNF-FGE problem considering non-cooperative domains. However, they do not consider the existence of multiple technological domains which limits the applicability of the proposed approach, and they neglect the cost of VNF migrations focusing only on the impact of VNF collocation.

We rely on the Advantage Actor Critic approach introduced in [104] and also used by [46] owing to its robustness and improved performance.

3.3.2 Reliable methods for placement and management

We discuss in this section placement and management methods which focus on algorithm reliability. First, we introduce hybrid ML approaches which consider the use of heuristics to increase the reliability of ML. Second, we study works applying robust models and finally we introduce the works considering realistic network load conditions.

Hybrid ML Approaches for placement optimization

Although ML-based techniques, such as DRL, have been shown to be robust when applied to solving various problems, these approaches also exhibit some drawbacks as they: i) are difficult to run due to a variety of hyper parameters to tune; ii) have convergence times difficult to control since they act as self-controlled black boxes; and iii) take too much time to start finding good solutions because their performance depends on the exploration of a huge number of states and actions.

Reference	Network Scale		Training Algorithm			Use of Convolutions		Use of heuristics
	S	M	VB	PB	MA/D	CNN	GCN	
[75]	✓		✓					
[42]	✓		✓			✓		
[41], [83], [84]	✓		✓		✓			
[85]	✓			✓		✓		
[44], [66]		✓		✓		✓		
[46]		✓		✓			✓	
[69]		✓		✓				✓
[58]		✓		✓			✓	✓

Small (**S**), Medium (**M**), Value-based (**VB**), Policy-based (**PB**), Multi-agent/descentralized (**MA/D**), Convolutional Neural Network (**CNN**), Graph Convolutional Network (**GCN**).

Table 3.7: Modeling and algorithmic aspects of DRL-based approaches.

This makes pure DRL approaches uncertain in real online optimization scenarios. Researchers then focused on developing hybrid methods that combine ML-based optimization with safer but less scalable techniques such as heuristics. The concept of Heuristically Accelerated RL is introduced in [105] as a way to solve RL problems with the explicit use of a heuristic function to influence the choice of actions by a learning agent.

As shown in [13], heuristically accelerated versions of well-known RL algorithms such as Q-learning, SARSA(λ) and TD(λ) have lower convergence times than those of their classical versions when applied to a variety of RL problems. Although Heuristically Accelerated RL has shown relevant results in different fields of applications such as video streaming [106], multi-agent systems [107] and robotics [108], to the best of our knowledge, it has only been used in [109] in the networking domain, namely, autonomous spectrum management.

We build on Heuristically Accelerated RL to propose HA-DRL and we apply this algorithm in this PhD thesis to slice placement. To the best of our knowledge, we are the firsts to work on HA-DRL. We rely on the formulation of heuristic function proposed in [105]. We adapt this method to DRL and combine it with an efficient placement heuristic proposed in [12] and presented in Chapter 5 to strengthen and accelerate the DRL learning process. This improves performance and safety when compared with state-of-the-art DRL placement approaches [46].

Other hybrid approaches for placement optimization combining DRL and heuristics were proposed recently. In [69], a HFA-DDPG algorithm combining Deep Deterministic Policy Gradient with a Heuristic Fitting Algorithm to solve the VNF-FGE problem is proposed. Evaluation results show that HFA-DDPG converges much faster than Deep Deterministic Policy Gradient. However, the approach adopts the strategy of using infinite action space formulation. Our approach, with a bounded action space, limits the complexity of the model specifically aiming its use for large-scale scenarios.

In [58], the REINFORCE algorithm is used to learn and to improve a VNE solution previously computed by a heuristic. In spite of the significant improvement of the initial heuristic solution obtained by this approach, this approach adopts a different principle than we use: using the DRL to improve the initial heuristic calculation and similarly to [69] and [58] relying on the heuristic algorithm to compute the placement decisions. In our case, we only use the heuristic to improve the DRL exploration process. As a result, our DRL agent can scale and be used after training even without the support of the heuristic.

Robust Optimization and ML methods

The term robustness has different meanings depending on the field of application. In Robust Optimization, robustness is related to the decision/solution itself. It is the capability of the algorithm solution of coping with the worst case without losing feasibility [110]. As depicted on Table 3.4, there are only a few recent works taking the robustness of placement procedures into account, most of them on Robust Optimization [52], [53], [56], [88].

In [56], the authors relies on the Γ-robustness [111], [112] concept to propose an ILP for the VNE problem considering uncertainties in resource requirements of the VNFs while fulfilling individual average round-trip delay bounds for each chain of VNFs. The models consider a picewise linearization model for queeing delays in each VNF and ordered chains with fixed ingress/egress points. The authors also propose a Variable Neighbourhood Search heuristic to enhance the scalability of the proposed solution.

The proposed model is interesting as it leverages Robust Optimization concept. But, a scalability evaluation of the proposed heuristic is missing. As a follow-up of [56], the authors of [88] apply the concept of light-robustness [113], [114] to solve a network slice design problem. In contrast to the classical VNE problem statements, which map a service (i.e. a given virtual network graph) directly onto a network infrastructure, the proposed network slice design problem includes the determination of the network slice topology. The authors model the problem as an ILP and compare it with other robust optimization solutions. However, no approximate strategy to efficiently solve the proposed problem is given.

Reference [52] is another example of work applying robust optimization techniques to solve a VNF placement problem under resource demand uncertainty. The authors formulate the problem as latency constrained flow problem using an ILP model. They consider multi-objective optimization with the minimization of energy usage and traffic injected to the network and propose a greedy heuristic for online optimization. However, the authors do not evaluate the performance of the heuristic in large-scale scenarios.

The same issue is encountered on [53]. The proposed multi-objective ILP model and heuristic explicitly provides for robustness to unknown or imprecisely formulated resource demand variations. It actuates powering down unused routers, switch ports and servers, and calculating the energy optimal VNF placement and network embedding also considering latency constraints on the service chains. They propose both exact and heuristic methods. But, an evaluation of the proposed heuristic in large-scale scenarios is missing.

Generally speaking, these works answer a question different from the one we are investigating as they evaluate the robustness of the decision whereas we want to evaluate the robustness of the learning process. In addition, despite their originality, the above approaches present some drawbacks, such as the lack of scalability of ILP, the sub-optimality of heuristic solutions, the fact that they consider offline optimization in which all slices to be placed are known in advance, and the fact that they are single objective optimization approaches, mainly focusing on energy consumption minimization. In this thesis, we propose to rely on a DRL-based approach in order to overcome ILP and heuristic drawbacks and consider multiple-optimization objectives.

In ML, specially in Deep Learning, robustness is related to the learned model. It is the property of the model (i.e., DNN) that determines its integrity under varying operating conditions [115].

The authors of [116] are the first to discuss robustness in the DRL context. They propose to use Genetic Algorithm to improve the robustness of a self-driving car application. Robustness is considered as the capacity of sustaining a high accuracy on image classification even when perceived images change and it is measured by neuron coverage, i.e., the ratio of the activated neurons in the DNN.

To the best of our knowledge, paper [117] is the only one to have proposed a DRL-based approach for slice placement and evaluated the learning robustness. However, the authors focus on evaluating the robustness of the DRL approach against random topology changes (e.g., node failures or deploying new nodes in the network topology). In this thesis, we focus on evaluating robustness against network load unpredictable variations. To the best of our knowledge, the present work is the first to perform such an evaluation in the context of DRL for slice placement.

The use of heuristics aims at increasing the reliability of DRL algorithms. However, most of the work is based on the assumption that the network load is static, i.e., slice arrivals occurs at a constant rate. To the best of our knowledge, the work we proposed in [16] is the first attempt to evaluate an online DRL-based approach in a non-stationary network load scenario whereas [118] only considers offline learning.

In addition, in both [16] and [118] it is assumed that network load has periodic fluctuations. In this PhD thesis we also study the behaviour of the algorithms proposed in [16] in case of an unpredictable network load disruption.

Placement optimization under realistic network load conditions

A recent body of research considers realistic network load scenarios when applying AI/ML-based approaches to support optimization of network slice life cycle management. For instance, the authors of [119] propose a deep learning-based data analytics tool to predict dynamic network slice traffic demands to help avoid SLA violations and network over-provisioning. The paper [120] also adopts neural networks to predict network slice traffic demands but, in this case, to perform proactive resource provisioning and congestion control.

To the best of our knowledge only paper [118] considers placement optimization using DRL in dynamic network load scenarios. The authors propose a Double Deep Q-learning Network algorithm for re-optimizing an initial VNF placement. They consider that the network load changes periodically with a time cycle T.

They then separate time cycle T in time intervals Δt, and train a Double Deep Q Network model to specifically take charge of VNF placement re-optimization in each time interval Δt. Despite its originality, the approach proposed in [118] presents two drawbacks: 1) it depends on offline learning, which is not applicable to online optimization scenarios; 2) it does not use DRL to optimize placement directly as the DRL algorithm selects the region of the network to re-optimize and delegates the optimization to threshold policy procedure. The heuristic calculation of placement decisions can lead to sub-optimal solutions. Contrary to [118], in this PhD thesis we study offline and online learning and propose a method that directly learns a placement optimization policy.

The paper [75] can be considered as a reference for VNF placement and orchestration automation. While adaptive automation has been treated in the context of wireless networks [121]–[124], this is the first work proposing an adaptive automated framework for the optimization of VNF orchestration and monitoring. It proposes a distributed management architecture and a machine learning based solution that performs VNF life-cycle operations (e.g., instantiation, scaling-up/down) while dynamically defining the length of the surveillance time and the frequency of operation for their monitoring system.

The main functional blocks of their solution are a VNF profiling component that applies unsupervised learning to cluster VNFs according to their resource consumption profile, a VNF profile monitoring that gathers real-time data allowing to verify the VNF profile prediction, a VNF optimal placement solution that calculates the VNF placement decision based on VNF profiles and a Surveillance Epoch Adjustment component that increases or reduces the number of monitoring operations according to the efficiency of VNF profile prediction.

Their solution is implemented in a small scale but realistic environment considering the deployment of an open source virtualized EPC VNF instances in a DC cluster. They treat traffic generated by User Equipment connected through two eNodeBs.

Despite the originality of this work, it leaves different open questions. It treats mainly VNF placement and does not investigate the challenge of managing multiple VNFs interconnected in the form of a network service or a Network Slice. Even if it considers the concept of Quality-of-Decision to allow coupling between orchestration and monitoring optimization, QoS metrics (e.g., latency, service availability) are not monitored or taken into account while calculating VNF profiling and placement.

Also, even if the proposed monitoring architecture is distributed, the proposed data analytics and decision making processes are done in a centralized way. Hence the concepts of federated management and self-management as mentioned in [125] are not present in this solution.

Furthermore, concerning the tested environment, it considered only virtualized EPC VNF deployment. Also, the PSN considered consists in only one DC. The evaluation of the proposed approach with the deployment of different types of VNFs interconnected in a PSN within multiple DCs is still an important step to be done.

Conclusion

In this chapter we discuss specific challenges and important questions open into the state-of-the-art. The state-of-the-art analysis has been performed according to the three domains and aspects as defined by the thesis scope introduced in Section 1.3: optimization models and algorithms for placement in virtual networks, presented in Section 3.1, Edge-enabled placement algorithms, presented in Section 3.2, and, finally, automated and reliable placement and management, presented in Section 3.3.

Chapter 4

Enabling on-Edge and on-Network Slice Placement: an ILP-based Solution

Contents

Introduction

In this chapter, we present the first contribution of this PhD thesis. In Section 4.1, we present definitions and assumptions for the proposed NSP problem. In Section 4.2, we state the NSP problem as well as the two approaches: offline and online of NSP. In Section 4.3, we describe the proposed unified mathematical modeling using ILP for the offline and online approaches of the NSP problem. The proposed mathematical models for the arrival process of requests and the network load are described in Sections 4.3.5 and 4.3.6, respectively. We discuss how we apply the unified ILP to the offline and online approach of the NSP problem in Sections 4.4.1 and 4.4.2 respectively. In Section 4.5, we present the evaluation of the proposed mathematical model for the offline approach of the problem. We conclude this chapter with some final remarks.

4.1 Definitions and assumptions

In this section, we define the key components of the proposed NSP problem. We also introduce the main assumptions adopted during the problem modeling process. The elements of the proposed problem are summarized in Table 4.1.

4.1.1 Physical Substrate Network Modeling

The PSN is composed of the infrastructure resources. These ones are IT resources (CPU, RAM, disk, etc.) needed for supporting the VNFs of slices along with the transport network, in particular VLs for interconnecting VNFs of slices. As illustrated in Figure 4.1, the PSN is divided into three components: the Virtualized Infrastructure corresponding to IT resources, the Access Network, and the Transport Network.

Problem domains	Problem elements	Problem sub-elements
PSN	Virtualized Infrastructure	DCs
		Server and switch
		DC link
	Transport Network	Router
		Transport link
	Access Network	UAP
		Access link
Network Slice	Network Slice Provider	Network Slice User
	NSPR	VNFs
		VL
		E2E chain

Table 4.1: Summary of the main elements of the NSP problem.

Virtualized Infrastructure

This component is the set of DCs interconnected by network elements (switches and routers) and either distributed in Points-of-Presence or centralized (e.g., in a big cloud platform). They offer IT resources to run VNFs. Following the reference architecture presented in [126] and describing the architecture of a network operator, we define three types of DCs—as shown in Figure 4.1—with different capacities: EDCs as local DCs with small resources capacities, Core DCs (CDCs) as regional DCs with medium resource capacities, and Central Cloud Platforms (CCPs) as national DCs with big resource capacities.

Access Network

This set represents User Access Points (UAPs) (Wi-Fi APs, antennas of cellular networks, etc.) and Access Links. Users access slices via one UAP, which may change during the life time of a communication by a user (e.g., because of mobility).

Transport Network

This is the set of routers and transport links needed to interconnect the different DCs and the UAPs.

Undirected graph model

The complete PSN is modeled as a weighted undirected graph $G_s = (N, L)$ where N is the set of physical nodes in the PSN, and $L \subset \{(a, b) \in N \times N \wedge a \neq b\}$ refers to a set of substrate links (i.e., transport, access and intra-DC links). Each node has a type in the set {UAP, router, switch, server}. The available CPU and RAM capacities on each node are defined respectively as $cap_n^{cpu} \in \mathbb{R}$, $cap_n^{ram} \in \mathbb{R}$ for all $n \in N$. The available bandwidth on the links are defined as $cap_{(a,b)}^{bw} \in \mathbb{R}, \forall (a, b) \in L$. The latency induced in each link is defined as $cap_{(a,b)}^{lat} \in \mathbb{R}, \forall (a, b) \in L$.

Other assumptions

Some assumptions are made for our PSN model. First, we assume that the cost of a server is the same for any server, and the cost of a link is the same for any link. This may not be the case if we consider that the different DCs available are not owned by the Network Slice Provider. Also, we assume that all the physical links are bidirectional and offer the same capacities when used for uplink and downlink. As the communication latency between two VNFs placed in the same server is not of the same order of magnitude as when they are placed in different servers, we assume that it can be neglected i.e., latency in DC links are set equal to 0. For the sake of simplicity, we consider that the latency induced by physical links are constant. Finally, we assume that the number of VNFs that can be deployed inside the same server is only bounded by the amount of CPU and RAM available in this machine.

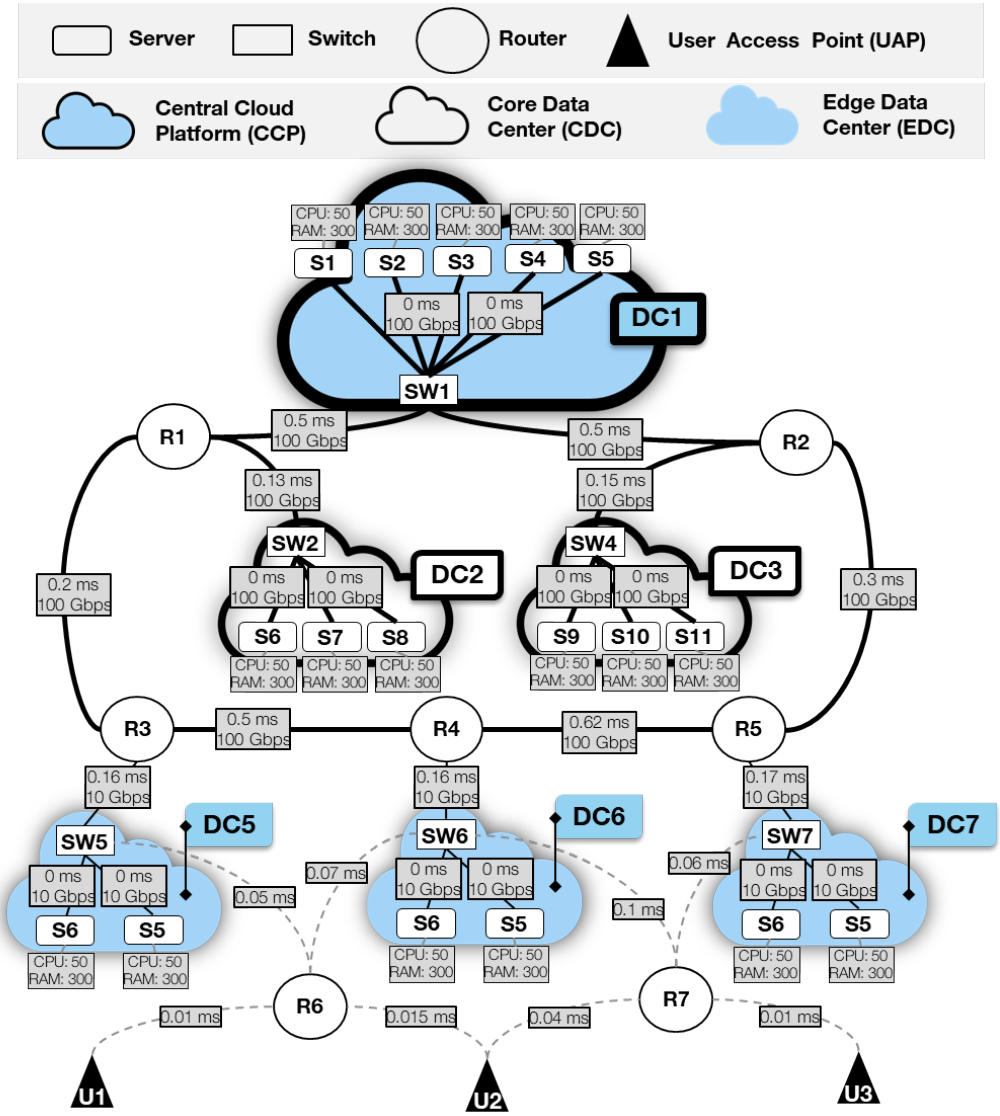


Figure 4.1: PSN.

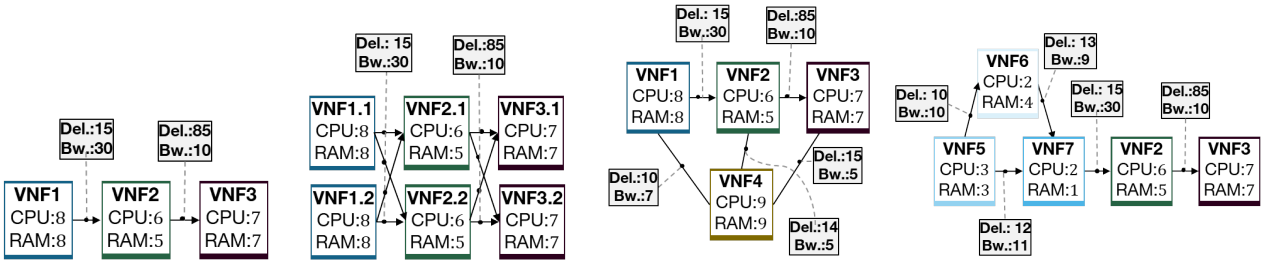
The amount of CPU available CPU and RAM on nodes of type switch, router and UAP is 0. Transmission links and DC links are labeled with a bandwidth capacity and a latency. Access links are labeled with a latency.

4.1.2 Network Slice Provider

The Network Slice Provider is the entity that offers the Network Slice to specific groups of NSUs. In our case it is considered also the owner of the PSN. We consider the cases where the NSUs for a given Network Slice are located nearby the respective UAP and that their location does not change in time (e.g., events on stadiums or airports, industry 4.0 use cases). This does not prescribe an individual user to move from one UAP to another.

4.1.3 Network Slice Placement Requests

We consider each slice as a finite number of VNFs to be placed and chained on the PSN. VNFs are batched and introduced in the network as NSPRs. The NSPRs represents a view of the resources requirements for a Network Slice to be placed by the slice orchestrator.



(a) SFC.

(b) Network Slice with VNF and VL redundancy.

(c) Network Slice with a centralized control plane VNF.

(d) Network Slice with a VNF implemented using a non-monolithic architecture.

Figure 4.2: Network Slice vs SFC.

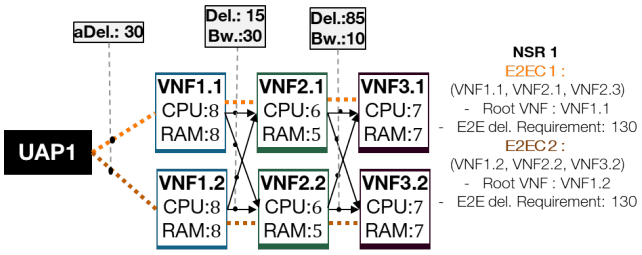


Figure 4.3: Example of NSPR and corresponding UAP.

Undirected graph model

The NSPR is similarly represented as a weighted undirected graph $G_v = (V, E)$ where V is the set of VNFs in the NSPR, and $E \subset \{(\bar{a}, \bar{b}) \in V \times V \wedge \bar{a} \neq \bar{b}\}$ is a set of VLs. The CPU and RAM requirements of each VNF of a NSPR are defined as $req_v^{cpu} \in \mathbb{R}$ and $req_v^{ram} \in \mathbb{R}$ for all $v \in V$, respectively. The bandwidth required by each VL in a NSPR is given by $req_{(\bar{a},\bar{b})}^{bw} \in \mathbb{R}$ for all $(\bar{a}, \bar{b}) \in E$ and the latency requirement of each VL in a NSPR is given by $req_{(\bar{a},\bar{b})}^{lat} \in \mathbb{R}$ for all $(\bar{a}, \bar{b}) \in E$.

NSPR topology and E2E chains

Most of the works on placement adopt the concept of SFC to represent what would be a Network Slice. A Network Slice in these cases are then described by sequence of VNFs (e.g., [64]). Following the SFC concept, a 5G Network Slice could be described as the following VNF sequence: NR $\rightarrow$ 5GC $\rightarrow$ Application Server.

Note however, that this is a high level view of 5G Network Slice. A lower level, i.e., more implementation oriented, view of a 5G Network Slice would have a more complex topology. We illustrate this in Figure 4.2 by showing at least three features that a 5G Network Slice may have that cannot be modeled using the SFC concept. The first one is high availability that is ensured via redundancy of VNFs and VLs (Figure 4.2(b)). Another characteristic is separation between user plane and control plane VNFs. A simple example of Network Slice comprising a centralized control plane function (VNF4) that is connected to the user plane functions is captured in Figure 4.2(c). The third one is when at least one VNF of the Network Slice is implemented using a non-monolithic architecture (e.g., micro-services architecture). An example is given on Figure 4.2(d), where VNF1 of Figure 4.2(a)) is decomposed in VNFs 5, 6 and 7.

These examples illustrate the importance of considering a Network Slice as a generalization of SFC as we do in the proposed NSPR modeling. Another particularity of the proposed NSPR model is the concept of E2E Chain. As presented in Figure 4.3, the E2E chains of a NSPR are defined by the sequences of VNFs traffic can traverse. We name the first VNF of each E2E chain by root VNF.

Inclusion of Network Slice Users' location

Each NSPR is associated with a group of NSUs. To consider the user location in the NSP decision, we insert in the E2E latency calculation the communication latency induced between the UAP of each NSPR and the Virtualized Infrastructure servers. This latency depends on UAP and servers locations. We refer to this latency as the access latency. Each group of NSUs imposes a maximum acceptable access latency between the UAP users are connected to and the root VNFs of the NSPR they request in order to ensure the feasibility of the communication. The E2E latency requirement for each E2E chain of one NSPR stands for the maximum latency allowed between the UAP associated to the NSPR and the last VNF of the E2E chain.

4.2 Network Slice Placement problem statement

In the proposed NSP problem, we consider that the NSPRs arrive sequentially to be placed and the placement optimization solution running inside the slice orchestrator need to choose which servers of the PSN to use to deploy the VNFs of these NSPRs and also which paths on the network to use to steer traffic between these VNFs. Multiple optimization objectives can be defined for this problem as discussed in chapter 3. In this proposal, we define two optimization objectives: resource usage minimization, slice acceptance ratio maximization.

The NSP problem is stated more formally as follows.

- *Given:* a NSPR graph $G_v = (V, E)$ arrived to be placed in arrival date $t_a \in [0, T]$ and exiting the network in exit date $t_e \in [t_a, T]$; a PSN graph $G_s = (N, L)$,
- *Find:* a mapping $G_v \rightarrow \bar{G}_s = (\bar{N}, \bar{L})$, $\bar{N} \subset N$, $\bar{L} \subset L$,
- *Subject to:* the VNF CPU requirements $req_v^{cpu}, \forall v \in V$, the VNF RAM requirements $req_v^{ram}, \forall v \in V$, the VLs latency requirements $req_{(\bar{a},\bar{b})}^{lat}, \forall (\bar{a}, \bar{b}) \in E$, the access latency requirements α_{max}, the E2E latency requirements δ, the server CPU available capacity $cap_s^{cpu}, \forall s \in S$, the server RAM available capacity $cap_s^{ram}, \forall s \in S$, and the physical link bandwidth available capacity $cap_{(a,b)}^{bw}, \forall (a, b) \in L$.
- *Objective:* maximize the NSPR acceptance ratio or minimize the total resource consumption.

In this PhD thesis we study two approaches of the NSP problem: the offline approach and the online approach. The particularities of the offline and online approaches of the proposed NSP problem are described in sections 4.2.1 and 4.2.2 respectively.

4.2.1 Offline Network Slice Placement problem

The offline NSP approach assumes complete knowledge of the future, i.e., it assumes that all the information about arrival dates and exit dates for all NSPRs arriving to the system during time horizon $t = 0, ..., T$ is available at time step $t = 0$.

By taking this assumption, we can calculate the optimal placement for all NSPRs in advance and in an "one-shot" way. If we aim to optimize the NSPR acceptance ratio, for example, we can decide at time step $t = 0$ which NSPRs r in R should be accepted and which ones should be rejected in order to obtain the optimal acceptance ratio.

To model this approach of the NSP problem, we need to consider the availability of three special datasets: 1) the set R comprising all the NSPRs arriving to be place during time horizon $t = 0, ..., T$; 2) the set T_a comprising the arrival dates t_a^r for all NSPRs r in R; and 3) the set T_e comprising the exit times t_e^r for all NSPRs r in R.

4.2.2 Online Network Slice Placement problem

The online NSP approach assumes partial knowledge of the future, i.e., it assumes that the arrival date t_a^r and the exit date t_e^r for an NSPR r in R are only available at the time when NSPR r arrives to be placed. This is equivalent to saying that, unlike the offline approach, the online approach assumes that the sets R, T_a and T_e are unknown.

Without complete information about the future, the NSP optimization must be done "on-the-fly", time step by time step, and NSPR by NSPR, using only the data of the NSPR that have just arrived on the system and of the resources available on the PSN at the current time step.

4.3 Mathematical modeling of the Network Slice Placement problem

In this section, we present an unified ILP formulation for the offline and online approaches of the NSP problem described in Sections 4.2.1 and 4.2.2 respectively. To formulate the optimization problem, we introduce the input data (Section 4.3.1), the decision variables (Section 4.3.2), identify the constraints (Section 4.3.3), which has to be satisfied by the placement algorithm, and describe the objective function formulation (Section 4.3.4). We describe in details how we apply this unified ILP formulation to the offline and online approaches of the NSP problem in Sections 4.4.1 and 4.4.2 respectively.

4.3.1 Input data

The inputs considered by the proposed ILP are given in Tables 4.2 and 4.3 for PSN and NPSR, respectively. Definition 4.3.1 complements Table 4.3 introducing the resource holding parameter used to formulate the network capacity constraints described in Section 4.3.3.

Definition 4.3.1 (Resource holding parameter) *Let T be a discrete time horizon. Let T_a and T_e be two sets comprising, respectively, the arrival dates t_a^r and exit dates t_e^r of all NSPRs r in R. We define the resource holding parameter $m_t^r \in \{0,1\}$ for all $t \in T$ and $r \in R$ as follows:*

$$m_t^r = \begin{cases} 1, & \text{if } t \in [t_a^r, t_e^r] \\ 0, & \text{otherwise} \end{cases} \tag{4.1}$$

Remark

Definition 4.3.1 assumes that if NSPR r in R is accepted to be placed in the PSN, it will "hold" the requested network resources from time step t_a^r until time step t_e^r.

Remark

The m parameter can actually be seen as a $R \times T$ binary matrix. For a given entry (r,t), m_t^r will assume value 1 if NSPR r requests to "hold" PSN resources at time t, i.e., $t_a^r \leqslant t \leqslant t_e^r$.

Parameter	***Description***
N	Network nodes (switch, servers, routers)
$S \subset N$	Set of servers
$L = \{(a,b) \in N \times N \wedge a \neq b\}$	Set of physical links
$cap_{(a,b)}^{bw} \in \mathbb{R}, \forall (a,b) \in L$	Bandwidth capacity of physical link (a,b)
$cap_s^{cpu} \in \mathbb{R}, \forall s \in S$	CPU capacity of server s
$cap_s^{ram} \in \mathbb{R}, \forall s \in S$	RAM capacity of server s
$\delta_{(a,b)} \in \mathbb{R}, \forall (a,b) \in L$	Latency induced by physical link (a,b)

Table 4.2: PSN parameters.

4.3.2 Decision variables

We use the two following binary decision variables:

Parameter	*Description*
R	Set of NSPRs to be placed
T_a	Set of NSPR arrival dates
T_e	Set of NSPR exit dates
$t_a^r \in T_a$	Arrival date of NSPR r
$t_e^r \in T_e$	Exit date of NSPR r
$m_t^r \in \{0,1\}$	Resource holding parameter for NSPR r *
V^r	Set of VNFs of the NSPR r
$E^r = \{(\bar{a},\bar{b}) \in V^r \times V^r \wedge \bar{a} \neq \bar{b}\}$	Set of VLs of the NSPR r
C^r	Set of E2E chains on the NPSR r **
$v_{root}^{r,c} \in V$	Root VNF of the E2E chain c of the NSPR r
$req_v^{cpu} \in \mathbb{R}$	CPU requirement of VNF v
$req_v^{ram} \in \mathbb{R}$	RAM requirement of VNF v
$d_{(\bar{a},\bar{b})}^{bw} \in \mathbb{R}$	Bandwidth requirement of VL $(\bar{a},\bar{b})$
$req_{(\bar{a},\bar{b})}^{lat} \in \mathbb{R}$	Latency requirement of VL $(\bar{a},\bar{b})$
$\delta_c^r \in \mathbb{R}$	Maximum E2E latency accepted for E2E chain c of NSR r
$\alpha_{max}^{r,c} \in \mathbb{R}$	Access latency requirement of E2E chain c of NSPR r
$\alpha_s^r \in \mathbb{R}$	Access latency between the UAP of NSPR r and server s

* See Definition 4.3.1.
** Each E2E chain is described by an ordered list of VLs.

Table 4.3: NSPR parameters.

- $x_s^v \in \{0,1\}, \forall r \in R, \forall v \in V^r, \forall s \in S$ is equal to 1 if the VNF v of the NSPR r is placed onto server s and 0 otherwise,
- $y_{(a,b)}^{(\bar{a},\bar{b})} \in \{0,1\}, \forall r \in R, \forall (\bar{a},\bar{b}) \in E^r, \forall (a,b) \in L$ is equal to 1 if the VL $(\bar{a},\bar{b})$ of the NSPR r is mapped onto physical link (a,b) and 0 otherwise,
- $z^r \in \{0,1\}, \forall r \in R$ is equal to 1 when the NSPR $r \in R$ is accepted and 0 otherwise. This is an auxiliary variable that is only needed when the acceptance ratio optimization objective is used.

4.3.3 Problem constraints

In this section, we describe the formulation of the problem constraints using linear equations and inequalities. We consider four blocks of constraints: VNF placement constraints (Section 4.3.3), Network resource capacity constraints (Section 4.3.3), Eligible physical path calculation constraints (Section 4.3.3), and Network Slice latency requirements constraints (Section 4.3.3). It is worth noting that the VNF placement constraints can be relaxed and considered as linear inequalities when using the maximization of accepted slice requests objective as described in Section 4.3.4.

VNF placement

The following constraint ensures that 1) all VNFs of each NSPR must be placed and 2) each VNF must be placed in only one server:

$$\forall r \in R, \quad v \in V^r, \quad \sum_{s \in S} x_s^v = 1 \tag{4.2}$$

Network resource capacities constraints

Equations (4.3) and (4.4) below ensure that the resource capacities of each server (for CPU and RAM, respectively) are not exceeded; the subsequent Equation (4.5) guarantees that the bandwidth capacity of each physical link is not exceeded.

Note that these constraints are defined for all arrival dates $t \in T_a$ since each time a NSPR arrives to be placed, the model need to evaluate if the network capacities are respected to define whether the NSPR can be accepted or not.

$$\forall t \in T_a, \forall s \in S, \quad \sum_{r \in R} \sum_{v \in V^r} req_v^{cpu} m_t^r x_s^v \leqslant cap_s^{cpu} \tag{4.3}$$

$$\forall t \in T_a, \forall s \in S, \quad \sum_{r \in R} \sum_{v \in V^r} req_n^{ram} m_t^r x_s^v \leqslant cap_s^{ram} \tag{4.4}$$

$$\forall t \in T_a, \forall (a,b) \in L, \quad \sum_{r \in R} \sum_{(\bar{a},\bar{b}) \in E^r} req_{(\bar{a},\bar{b})}^{bw} m_t^r y_{(a,b)}^{(\bar{a},\bar{b})} \leqslant cap_{(a,b)}^{bw}. \tag{4.5}$$

Eligible physical path calculation

We use flow conservation constraints [22] formulated by Equations (4.6), (4.7), and (4.8) to the definition of the eligible physical paths in which to map every VL $(\bar{a},\bar{b}) \in E^r$ for all NSPRs $r \in R$ to place as: for all $a \in S$, $r \in R$, and $(\bar{a},\bar{b}) \in E^r$,

$$\sum_{\substack{b \in N: \\ (a,b) \in L}} y_{(a,b)}^{(\bar{a},\bar{b})} - \sum_{\substack{b \in N: \\ (b,a) \in L}} y_{(b,a)}^{(\bar{a},\bar{b})} = x_a^{\bar{b}} - x_a^{\bar{a}}, \tag{4.6}$$

and

$$\forall a \in N \setminus S, \forall r \in R, \forall (\bar{a},\bar{b}) \in E^r, \quad \sum_{\substack{b \in N: \\ (a,b) \in L}} y_{(a,b)}^{(\bar{a},\bar{b})} - \sum_{\substack{b \in N \\ (b,a) \in L}} y_{(b,a)}^{(\bar{a},\bar{b})} = 0, \tag{4.7}$$

$$\forall (a,b) \in L, \forall r \in R, \forall (\bar{a},\bar{b}) \in E^r, \; y_{(a,b)}^{(\bar{a},\bar{b})} + y_{(b,a)}^{(\bar{a},\bar{b})} \leqslant 1. \tag{4.8}$$

The left hand sides of Equations (4.6) and (4.7) compute the difference between the activated links outgoing and incoming from/to each node $a \in N$. One computation is done for each VL $(\bar{a},\bar{b}) \in E^r$. If $a \in S$, the right hand side of Equation (4.6) ensures that the computed difference must be equal to $x_a^{\bar{b}} - x_a^{\bar{a}}$. That is: 0 if server a is used to place both VNFs $\bar{a}$ and $\bar{b}$ or if its not used to place neither of them; -1 if only the source VNF $\bar{a}$ is placed on a; 1 if only the destination VNF $\bar{b}$ is placed on a. If $a \in N \setminus S$, Equation (4.7) ensures that this difference will always be 0. Equation (4.8) imposes that each link must be used only in one direction when mapping a specific VL.

Network Slice latency requirements constraints

Equation (4.9) below guarantees that the latency requirements of each VL of all NSPRs will be respected. Equation (4.10) ensures that the access latency requirements are respected and Equation (4.11) reflects the fulfillment of the required E2E latency:

$$\forall r \in R, \forall (\bar{a},\bar{b}) \in E^r, \sum_{(a,b) \in L} \delta_{(a,b)} y_{(a,b)}^{(\bar{a},\bar{b})} \leqslant req_{(\bar{a},\bar{b})}^{lat}, \tag{4.9}$$

$$\forall r \in R, \forall c \in C^r, \sum_{s \in S} \alpha_s^r x_s^{v_{root}^{r,c}} \leqslant \alpha_{max}^{r,c}, \tag{4.10}$$

$$\forall r \in R, \forall c \in C^r, \sum_{s \in S} \alpha_s^r x_s^{v_{root}^{r,c}} + \sum_{(a,b) \in L} \sum_{(\bar{a},\bar{b}) \in c} \delta_{(a,b)} y_{(a,b)}^{(\bar{a},\bar{b})} \leqslant \delta_c^r. \tag{4.11}$$

4.3.4 Objective function

We consider two possible objective functions. The first one is to minimize the consumption of resources (Section 4.3.4). The second one is the maximization of acceptance ratio (Section 4.3.4).

We study in this chapter the NSP as a single objective optimization problem, i.e., considering only one of the formulated optimization objectives. The study of the multi-objective problem, i.e., the problem considering multiple objectives at once, is treated in Chapter 6.

Minimization of the total resource utilization

The minimization of the total resource utilization objective function is given by the minimization the weighted sum of resource consumption represented by Equation (4.12):

$$\min_{x,y} \sum_{r\in R}\sum_{v\in V^r}\sum_{s\in S}\frac{req_n^{ram}x_s^v}{cap_s^{ram}} + \sum_{r\in R}\sum_{v\in V^r}\sum_{s\in S}\frac{req_n^{cpu}x_s^v}{cap_s^{cpu}} + \sum_{r\in R}\sum_{(\bar{a},\bar{b})\in E^r}\sum_{(a,b)\in L}\frac{req_{(\bar{a},\bar{b})}^{bw}y_{(a,b)}^{(\bar{a},\bar{b})}}{cap_{(a,b)}^{bw}} \tag{4.12}$$

We set the weight of each considered resource as the resource scarcity (i.e., resource demands divided by resource capacity). This is done in order to differentiate the cost of a resource from another. In this way, the the scarcest resources are considered more expensive.

Note, however, that in the standard version of the proposed model, the placement of all VNFs of all NSPRs is mandatory otherwise the solution would violate Equation (4.2) and be considered infeasible. The optimization objective in this case can then be simplified to the minimization of bandwidth resources utilization given by Equation 4.13:

$$\min_{x,y} \sum_{(\bar{a},\bar{b})\in E}\sum_{(a,b)\in L} y_{(a,b)}^{(\bar{a},\bar{b})} req_{(a,b)}^{bw} \tag{4.13}$$

Maximization of slice acceptance ratio

The maximization of accepted NSPRs objective function which is given by Equation (4.14):

$$\max_{x,y,z} \sum_{r\in R}\frac{z_r}{|R|} \tag{4.14}$$

where the additional constraints given by Equation (4.15) and (4.16) below need to be inserted in the model:

$$\forall r \in R, \forall v \in V^r, \qquad z_r \leqslant \sum_{s\in S} x_s^v, \tag{4.15}$$

$$\forall r \in R, \qquad z_r \geqslant \sum_{s\in S}\sum_{v\in V^r} x_s^v - |V^r - 1|. \tag{4.16}$$

Note that using this strategy would require to remove VNF acceptance obligation to the model, i.e., removing Equation (4.2).

4.3.5 Network Slice Placement Requests arrival process modeling

In the work developed in this chapter, we assume that the NSPRs arrive in the system following a stationary Poisson process, that is, a Poisson process in which the rate parameter λ is constant. This is not necessary the case in the work developed in Chapter 6. We call this process the static NSPR arrival process and define it in Definition 4.3.2.

Definition 4.3.2 (Static NSPR arrival process) *Let J be the set of resources in the network (i.e., CPU, RAM, bandwidth). Let $\mathcal{K}$ be the set of NSPR classes. We name λ^k the rate parameter of the Poisson distribution describing the number of arrivals of NSPRs belonging to class k, i.e., $f(a;\lambda^k) = Pr(X^k = a) = \frac{(\lambda^k)^a e^{-\lambda^k}}{a!}$ in which a is the number of arrival occurrences and X^k is the random variable described by the Poisson distribution and f is the probability mass function.*

Remark

In this whole manuscript, we assume that the lifespan of a NSPR belonging to class k follows a negative exponential distribution with rate parameter μ^k.

Using Definition 4.3.2, we can simulate the arrival dates t_a^r and exit dates t_e^r for all NSPRs r in R arriving to the system. We only need to set the parameters λ^k and μ^k and sample from the respective Poisson and Negative Exponential distributions to determine the number of NSPRs arriving and leaving the network at each simulated time unit. Typically these parameters are set according to the network load condition we want to simulate. The network load modeling is introduced in Section 4.3.6.

4.3.6 Network load modeling

From the Definition 4.3.2 of Static NSPR arrival process we define here the network load model adopted in the work performed in this chapter as follows:

Definition 4.3.3 (Stationary Network Load model) *Let J be the set of resources in the network (i.e., CPU, RAM, bandwidth). Let $\mathcal{K}$ be the set of NSPR classes. We compute the load generated by arrivals of NSPRs of class k in $\mathcal{K}$ for resource j in J as in [127]:*

$$\rho_j^k = \frac{1}{C_j} \frac{\lambda^k}{\mu^k} A_j^k, \tag{4.17}$$

where C_j is the total capacity of resource j, A_j^k is the number of resource units requested by an NSPR of class k, λ^k is the arrival rate for an NSPR of class k, and $1/\mu^k$ is the average lifetime of an NSPR of class k.

Remark

The global network load ρ_j for each resource j in J can be calculated as the sum of the network loads generated by all clases, that is,

$$\rho_j = \sum_{k \in \mathcal{K}} \rho_j^k \tag{4.18}$$

It is worth noting $0 \leqslant \rho_j(t) \leqslant 1$.

4.4 Application of the mathematical model

In this section, we describe how we apply the proposed mathematical model to the offline NSP approach (see Section 4.4.1) and to the online NSP approach (see Section 4.4.2).

4.4.1 Application of the mathematical model to the offline approach

It is easy to see that the formulation proposed in the above sections is directly applicable to the offline approach of the NSP problem described in Section 4.2.1. However, we make two additional simplifying assumptions when applying the proposed ILP model to solve the offline NSP problem in Section 4.5: 1) we consider a time horizon with a single time unit, i.e., $T = \{0\}$; and 2) we consider that all NSPRs arrive at time $t = 0$, i.e., $t_a^r = 0$ for all r in R.

These two simplifying assumptions are motivated by two facts: 1) the ILP for the offline problem is not tractable if we consider realistic scenarios, i.e., considering a PSN with thousands of nodes and links and a time horizon typically covering millions of time units, because the number of Constraints (4.3), (4.4), and (4.5) to solve would become too large and solving the problem using a linear solver such as CPLEX [128] would require an explosive amount of storage and computational resources; 2) the main objective of the evaluation performed in Section 4.5 is to evaluate the relevance of the proposed E2E latency model (see Section 4.5.3), and this can be accomplished using the simplified scenario based on the two simplifying assumptions described here.

By using the two simplifying assumptions mentioned we reduce the number of Constraints (4.3), (4.4), and (4.5) of our problem. The set T_a can be reduced to $\{0\}$ without loss of generality and thus facilitate the evaluation of the ILP model for our desired purposes.

4.4.2 Application of the mathematical model to the online approach

As shown in Section 4.2.2, in the online approach of the NSP problem the sets R, T_a and T_e are not completely known since the arrival date t_a^r and exit date t_e^r for a request r in R is only known in the moment it arrives to be placed (i.e., at date t_a^r).

We apply then the proposed mathematical model to the online approach of the NSP problem as follows. Let $R'_t = \{r_1^t, r_2^t, ..., r_{|R'_t|}^t\}$ be the set of all the NSPRs arriving to the system in date t for all t in time horizon T. We consider the case in which NSPRs are placed sequentially one by one. In this case, at each time step t the model is ran $|R'_t|$ times; in each run the model tries to place one NSPR. Hence, in each run i in $\{1, ..., |R^t|\}$ of the model for specific time step t, the set R is defined as $R = \{r_i^t\}$. The set R is updated using this same rule for all time steps t in T. Following the same reasoning the sets T_a and T_e are defined containing only one element each, i.e., $T_a = \{t_a^{r_i^t}\}$ and $T_e = \{t_e^{r_i^t}\}$ in which r_i^t is the NSPR to be placed on the current run of the model.

4.5 Experiments, evaluation results, and discussions

We present in this section the implementation and the numerical experiments carried out to evaluate the proposed ILP model for the offline approach of the proposed NSP problem. In Section 4.5.1, we describe the implementation details and experimentation settings. In Section 4.5.2, we describe the algorithms tested. In Section 4.5.3, we describe the evaluation metrics used. Finally, in Section 4.5.4, we describe the evaluation results and discussion.

4.5.1 Implementation details and experimentation settings

Proposal architecture description

The architecture of the proposed NSP solution is given in the Figure 4.4. First, we assume an application service provider requests to a Network Operator a Network Slice in a B2B mode to serve his own set of end-users. The application service provider will express some high-level (e.g., application level) QoS and QoE requirements that the requested network service should guarantee. These requirements are integrated to be part of the SLA used for the selection of VNFs and VLs capacities which defines the Network Slice template. This template will be registered in the NSU database. Then, the NSPRs and the PSN Transit Stub model (containing the PSN topology and resource availability information) are used by the Placement Module so that the Placement decision is calculated using the proposed ILP algorithm.

This decision is to be further applied by the different Network Orchestrator layers down to the PSN. The PSN information is provided in real-time by the Infrastructure Monitoring System and registered into the PSN database.

Proposal implementation description and tools

To allow the experimentation the proposed ILP model for offline NSP, we have implemented the Placement Module, the NSU database and the PSN database described in Section 4.5.1. Since the NSU and PSN information are taken as a static input dataset in our case, we do not need to implement the SLA neither the Network Monitoring Subsystem. Note, however, that for studying the Network Slice Composition, the SLA and a service composition logic need to be implemented. In a case which we have multiple placement decisions done successively in time, the Network monitoring subsystem need to be implemented to monitor the availability of network resources.

Figure 4.5 presents a UML diagram view of the proposal implementation. We have implemented the proposed ILP formulation in Julia [129] and used the default branch-and-bound algorithm from `ILOG` CPLEX 12.9 solver to solve it [128].

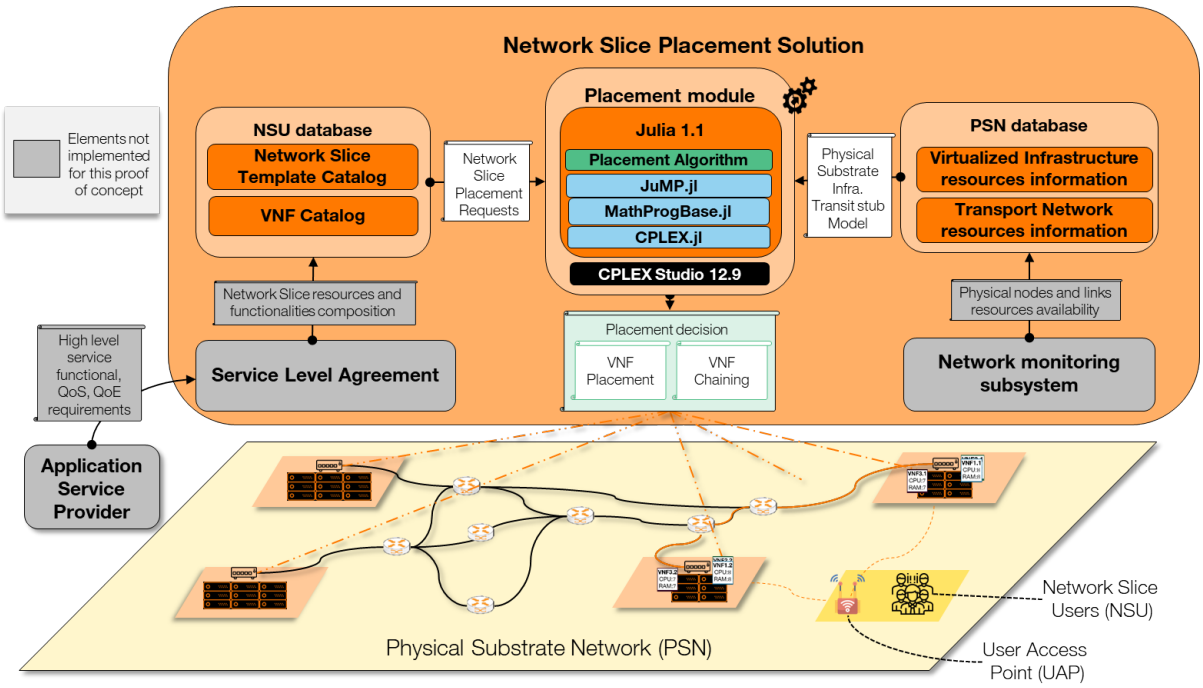


Figure 4.4: ILP-based NSP solution architecture.

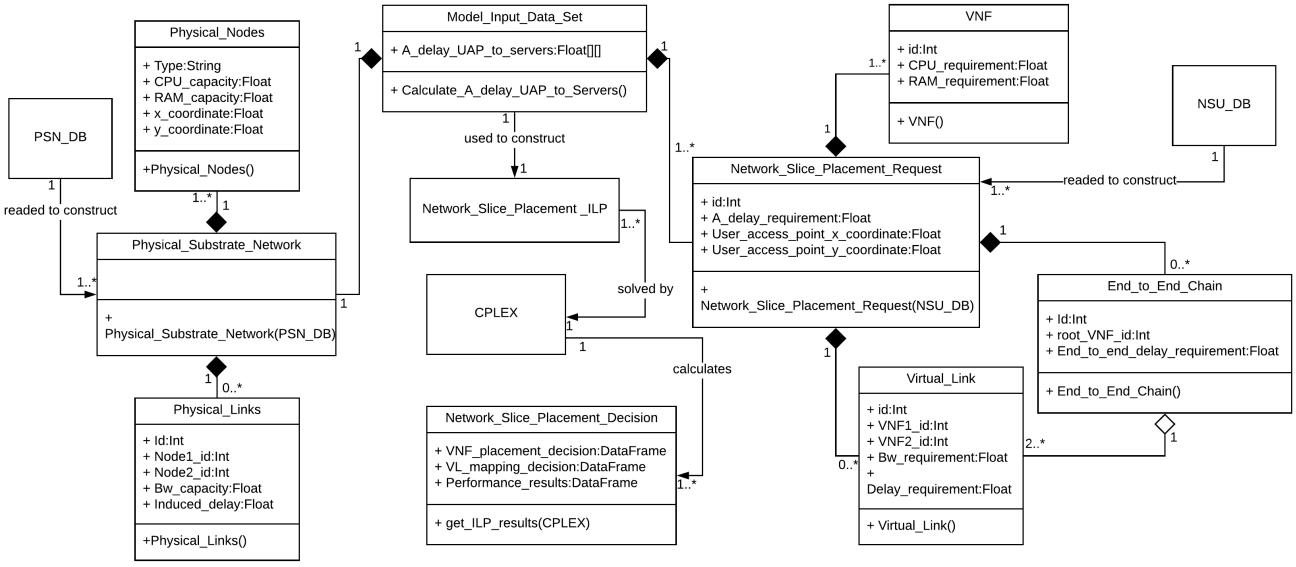


Figure 4.5: View of the proposal architecture implementation using UML class diagram.

We used a 2x6 cores @2.95Ghz CPU machine with 96GB of memory in our experiments. To allow extensive simulations we stop CPLEX execution when the gap is lower than 1% or after two hours. We designed a random parameter generator based on the GT-ITM tool [98] to generate different simulation scenarios. Each simulation scenario is defined by a PSN and a set of NSPRs. A set of 10 random scenarios was generated for each combination of the input values to model parameters generator detailed in Tables 4.4-4.6.

Input elements to model parameters generator	Input values to model parameters generator	Parameters generated to model execution
# of servers in each DC	5	List of identifiers of servers in each DC
# of switches in each DC	1	List of identifiers of switches in each DC
Grid for servers and switches coordinates	100×100	Servers locations (cartesian coordinates)
		Switches locations (cartesian coordinates)
Min. and max. CPU capacity for servers	[50,100]	Servers CPU capacity
Min. and max. RAM capacity for servers	[50,100]	Servers RAM capacity
Link existence probability	1	List of identifiers of links between servers and switches
Min. and max. bandwith capacity	[50,500]	Links bandwidth capacity
		Latency induced by each link (euclidean distance)

Table 4.4: Virtualized Infrastructure elements to model parameters generator and parameters generated to model execution.

Input elements to model parameters generator	Input values to model parameters generator	Parameters generated to model execution
# of routers	5, 10, 20 and 40	List of identifiers of routers
Grid for routers coordinates	100×100	Routers locations (cartesian coordinates)
Link existence probability	1	List of identifiers of links between routers
Min. and max. bandwidth capacity	[50,500]	Links bandwidth capacity
		Latency induced by each link (euclidean distance)
# of DCs to which each router is connected	1	List of identifiers of links between routers and DCs
Min. and max. bandwidth capacity	[50,500]	Links bandwidth capacity
		Latency induced by each link (euclidean distance)
Grid for UAP coordinates	100×100	UAP location (cartesian coordinates)
		Latency between UAP and each server (euclidean distance)

Table 4.5: Transport Network and Access Network input elements to model parameters generator and parameters generated to model execution.

Input elements to model parameters generator	Input values to model parameters generator	Parameters generated to model execution
# of VNFs	3, 5, 10, 15 and 20	List of identifiers of VNFs
Grid for VNFs pseudo coordinates	100×100	VNFs pseudo coordinates
Min. and max. CPU requirement	*	VNFs CPU requirement
Min. and max. RAM requirement	*	VNFs RAM requirement
VLs existence probability	0.5	List of identifiers of VLs between VNFs
Min. and max. bandwidth requirement	[5,10]	VLs bandwidth requirement
		Latency required by each VL (euclidean distance)
		List of identifiers of E2E chain
Min. # of VNFs by E2E chain	3	List of identifiers of VNFs in E2E chain
Max. # of end E2E chain by NSPR	10	Root VNF of E2E chain
		Latency required between UAP and root VNF
		E2E latency required by each E2E chain

Table 4.6: NSPR input elements to model parameters generator and parameters generated to model execution.

Physical Substrate Network settings

Tables 4.4 and 4.5 summarize the PSN generator inputs and outputs. A PSN is generated as a Transit-stub graph [130]. Transport Network is represented by a transit domain with a certain number of routers. Each DC is represented by a stub domain composed by 5 servers connected to a switch. Here we do not make a differentiation between the capacities of EDC, CDC and CCP DCs. Each switch is connected to a Transport Network router. Resource capacities of servers and physical links are generated randomly in the intervals specified in Tables 4.4 and 4.5. A grid is used to generate the locations of physical nodes as Cartesian coordinates. The latency induced by the physical links is given by the Euclidean distance between the nodes.

Network Slice Placement Requests settings

Table 4.5 summarize the NSPR generator inputs and outputs. An NSPR is generated as a random graph. In each simulation scenario, we consider 5, 10, 20, 40, 80, or 100 NSPRs to be placed each one having specific requirements for VNFs and VLs. The resource requirements of VNFs and VLs are generated randomly within an interval. This interval is fixed for bandwidth requirement generation (see Table 4.6) but varies according to the scenario for CPU and RAM requirements generation. The lower bounds (Lb) and upper bounds (Ub) of these intervals are defined by Equation (4.19), where $nVNFs$ represent the number of VNFs per NSPR in the scenario, S is the set of servers and cap_s^j refers to the capacity of resource $j \in \{cpu, ram\}$ for server $s \in S$.

$$\begin{aligned} Ub_j &= \min_{s \in S} cap_s^j \\ Lb_j &= \begin{cases} \frac{\max_{s \in S} cap_s^j}{nVNFs} + 1, & \text{if } \frac{\max_{s \in S} cap_s^j}{nVNFs} + 1 \leqslant Ub_j \\ 1, & \text{otherwise} \end{cases} \end{aligned} \tag{4.19}$$

To compute VL latency requirements, we generate Cartesian coordinates for the VNFs of each NSPR and calculate the Euclidean distance between them.

The E2E latency requirement for each E2E chain is the sum of the latency requirements of the VLs composing the chain. The Access Latency requirement for a given NSPR is the percentile 60 of an ordered vector containing the latency between UAP associated to the the NSPR and the servers.

4.5.2 Tested algorithms

In this evaluation, we consider the ILP formulation described in Section 4.3 applied to the offline approach of the NSP problem as described in Section 4.4.1. We consider a single objective optimization problem using the minimization of the total resource utilization objective function described in Section 4.3.4.

4.5.3 Evaluation metrics

We examine two types of performance metric.

Scalability

To evaluate the scalability of our approach we adopt two metrics: the model execution time in seconds and the estimated gap to optimal in % after two hours.

Relevance of the E2E latency model

We compare our location-based model with a location-agnostic model widely used in the state-of-the-art, that considers E2E latency but does not take user location into account (e.g., [51]).

This comparison is done using two metrics: E2E latency requirement violation and average E2E latency requirement violation. These metrics are defined by Equations (4.20) and (4.22).

Let Δ_c^r and $\bar{\Delta}_c^r$ $\forall r \in R$, $\forall c \in C^r$ be the E2E latencies calculated by the location-based and location-agnostic models, respectively. E2E latency requirement violation for each E2E chain is given by Equation (4.20). E2E latency requirement violation metric for each simulation scenario m is given by Equation (4.21). The average E2E latency requirement violation metric is given by Equation (4.22).

$$\epsilon_c^r = \begin{cases} \max(0, \frac{\Delta_c^r - \delta_c^r}{\delta_c^r}), & \text{if location-based model} \\ \max(0, \frac{(\Delta_c^r + \alpha_{max}^r) - \delta_c^r}{\delta_c^r}, & \text{otherwise} \end{cases} \tag{4.20}$$

$$\epsilon_m = \sum_{r \in R} \frac{1}{|C^r|} \sum_{c \in C^r} \epsilon_c^r \tag{4.21}$$

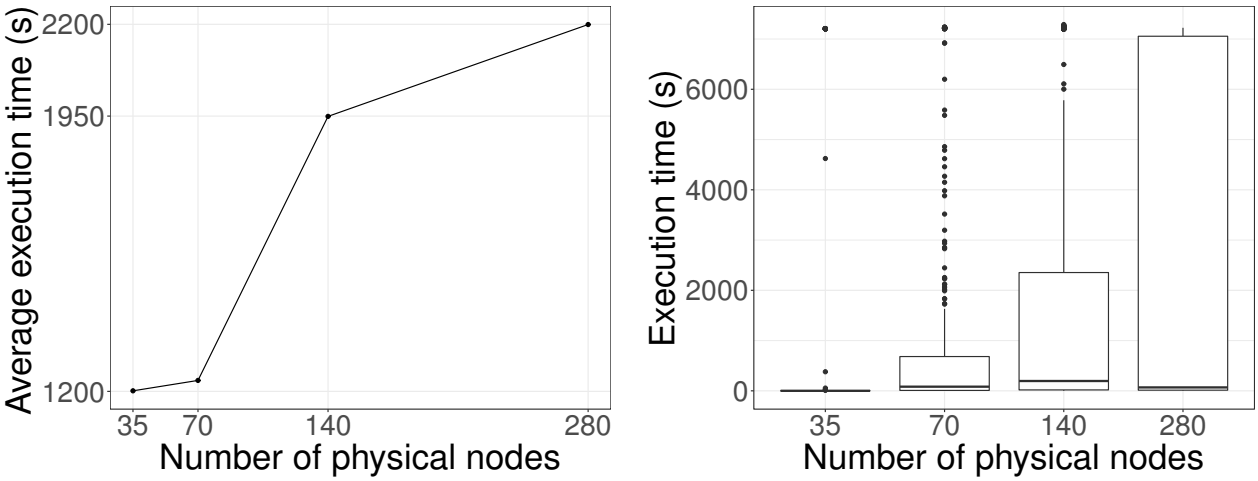
$$\bar{\epsilon} = \frac{1}{M} \sum_{m=1,\ldots,M} \epsilon_m \tag{4.22}$$

4.5.4 Evaluation results and discussion

Scalability evaluation results

Figures 4.6(a) and 4.7(a) respectively show the evolution of the average execution time of our model and the average gap to optimal estimated by CPLEX at the end of the execution time in function of the number of PSN's nodes. These two metrics significantly increase for the scenarios with a PSN with more than 70 nodes due to complexity explosion. The variation of execution times and optimality gaps for all solved simulation scenarios are illustrated in Figures 4.6(b) and 4.6(b), respectively.

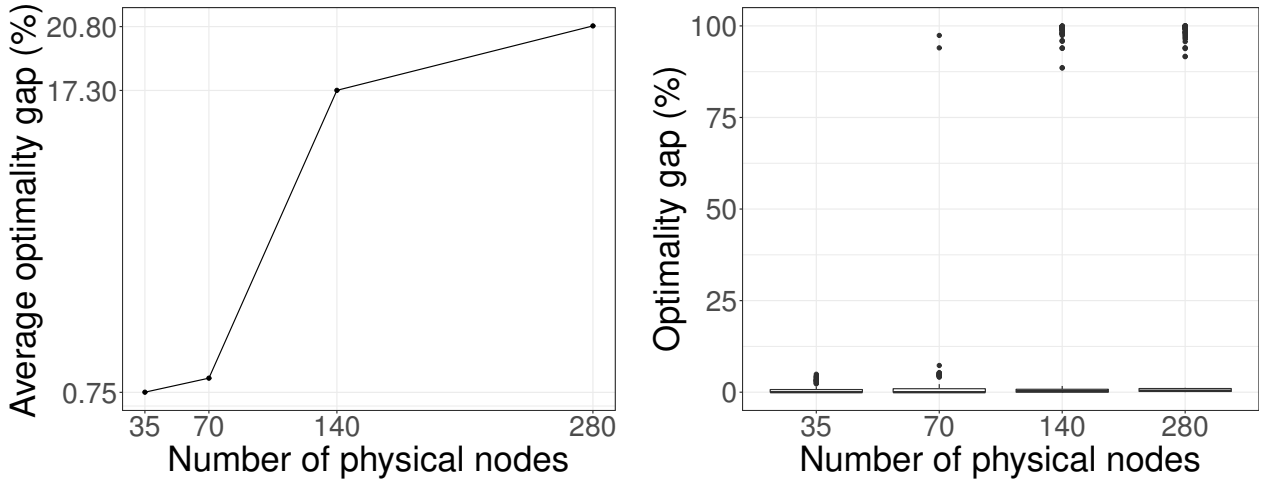
We note high variation in the execution times associated with the different levels of complexity of simulation scenarios. Optimality gaps are often at 0% showing that the algorithm converged in 2 hours in most simulations.



(a) Average execution time.

(b) Execution time variation.

Figure 4.6: Scalability evaluation results: execution time.



(a) Average optimality gap.

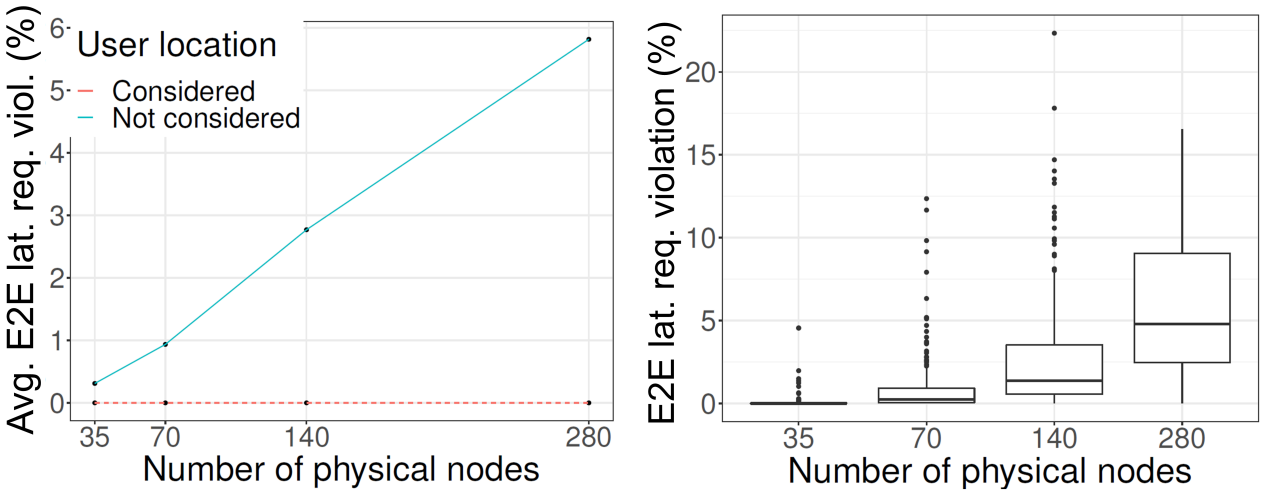
(b) Optimality gap variation.

Figure 4.7: Scalability evaluation results: optimality gap.

E2E latency model relevance analysis

The pertinence of the proposed E2E latency model is analyzed in Figures 4.8 and 4.9. We compare E2E latency requirement violations obtained when we solve the proposed ILP (UAP location is considered) and when we solve the alternative ILP (UAP location is not considered).

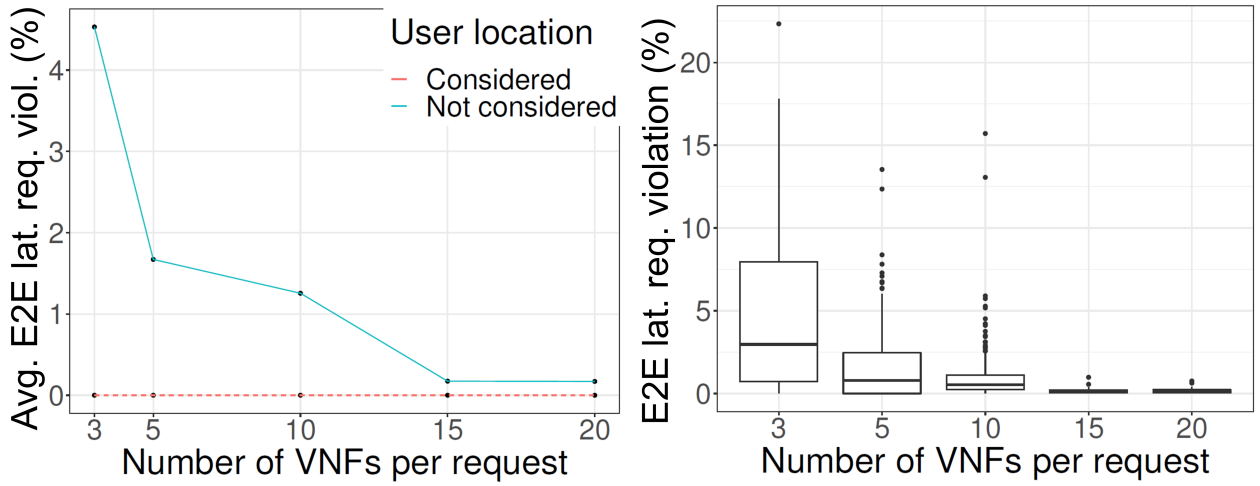
Figure 4.8(a) presents the evolution of the average E2E latency requirement violation according to the number of nodes on the PSN. In contrast to our formulation that always respects E2E latency requirements, a growing average E2E latency requirement violation is observed when we do not take into account user location. We observe a high variation of the E2E latency requirement violations captured by Figure 4.8(b) as we notice that it may exceed 15% in some cases.



(a) Average E2E latency requirement violation.

(b) E2E latency requirement violation variation.

Figure 4.8: E2E latency model relevance analysis variating number of physical nodes.



(a) Average E2E latency requirement violation.

(b) E2E latency requirement violation variation.

Figure 4.9: E2E latency model relevance analysis variating number of VNFs.

Figure 4.9(a) presents the evolution of the average of the E2E latency requirement violations according to the number of VNFs of the NSPRs. We also observe that the average violation decreases while the number of VNFs in the NSPRs increases. This happens because the more VNFs we have in the NSPRs, less the NSPRs are spread in the PSN, this reduces the E2E latency requirements violations. In fact, the CPU and RAM requirements of each VNF decrease when the number of VNFs per request increases (see Equation (4.19)). Hence, the tested models concentrate more VNFs inside the same machines to prevent from using link resources. Observing the variation of the E2E latency requirement violations in Figure 4.9(b), we see again a high deviation of the violation results still higher than the median up to 20%.

Resource consumption evaluation

Figure 4.10 shows the average resource consumption of the proposed solutions. We remark that the RAM and CPU consumption are always equal to 100% of the required since the mapping of all VNFs is mandatory. However, we notice an average bandwidth economy of at least 20% of the required due to the placement of multiple VNFs of the same NSPR inside a same server.

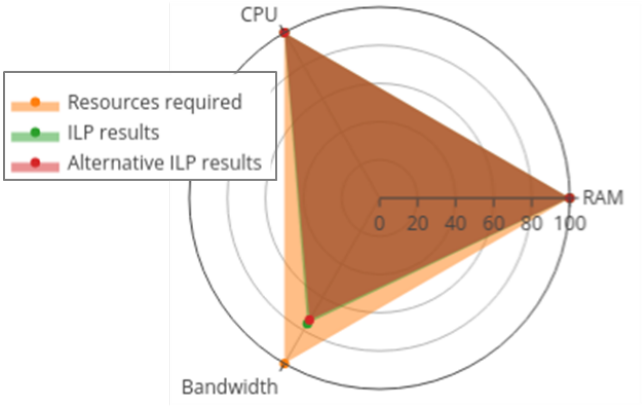


Figure 4.10: Resource consumption evaluation: CPU, RAM, BW.

Conclusion

In this chapter, we presented the first contribution of this PhD thesis, namely, *Enabling on-Edge and on-Network Slice Placement: an ILP-based Solution.*

We proposed an ILP model for the NSP problem focusing on Edge-specific constraints (i.e., user location, E2E latency) and evaluate the offline approach of the considered NSP problem. The proposed ILP solution gathers four main contributions. First, a new E2E latency model integrating the user location as an end-point of the Network Slice without assuming that the preferred location for each Virtual Node is known (generalizing the hypothesis introduced by [59]). Second, a model that can consider a higher number of NSUs and explore the possibility of grouping them (unlike [64]) for scalability issues. Moreover, modeling Network Slices as a generalization of SFCs since they can have a cyclic topology (generalizing at least [72] and [64] hypotheses). Finally, removing all restrictions on the placement location of two VNFs of the same Network Slice (generalizing [60] hypothesis).

We have implemented the proposed ILP formulation and used the default branch-and-bound algorithm from `ILOG` CPLEX 12.9 solver to do extensive simulations in order to evaluate the proposed ILP when applied to solve offline NSP. The experiments performed were useful to show the relevance of the proposed E2E latency model since the simulations showed that taking into account the user location as an end-point of the Network Slices is essential in order to ensure the fulfillment of the E2E latency requirements. By solving the different simulation scenarios, we were able to see that we can reach 5% of average E2E latency requirement violation and up to 20% when we do not include the user location to calculate the NSP decision.

However, the scalability analysis performed in Section 4.5.4 shows a first limitation of this offline ILP as even when applying the simplifying assumptions described in Section 4.4.1 to reduce the problem size, the ILP suffer from combinatorial explosion as the number of NSPRs to be placed increase. Hence, even if the the strategy of placing the complete set of NSPRs in "one shot way" allow to obtain the optimal placement, the ILP suffers from complexity explosion, i.e., convergence takes hours, when solving large scale scenarios.

Besides, the assumption about the data availability, i.e., complete knowledge about R, T_a and T_e sets is highly unpractical. Even if operators are investing more on forecasting systems in order to predict future network demands, assuming a complete a priory knowledge about the configuration of future requests, their arrival dates, and exit dates still not possible.

Motivated by these two limitations of the offline NSP, in the rest of this PhD thesis we concentrate in the online approach of the NSP problem. We propose in the following chapter a heuristic algorithm to solve the online NSP problem in an efficient way.

Chapter 5

Optimizing Large Scale Network Slice Placement: a Heuristic using the Power of Two Choices

Contents

Introduction

In this chapter, we present the second contribution of this PhD thesis. We build on the problem formulation and the conclusions obtained in Chapter 4 to propose a heuristic for the online approach of the NSP problem described in Section 4.2.2. In Section 5.1.1, we introduce the additional assumptions. In Section 5.1.2, we present P2C principle used in the heuristic design. In Section 5.1.3, we describe how the P2C principle is adapted to the proposed NSP problem. In Section 5.2, we provide a complete description of the proposed optimization heuristic. Then, we provide a detailed description of the main procedures of the proposed heuristic: calculation of eligible servers for placement, in Section 5.2.2, and the selection policy for candidate placement servers, in Section 5.2.3. In Section 5.3, we present how we implemented the heuristic and the performance evaluation results. We close the chapter with some concluding remarks.

5.1 Heuristic design assumptions and principles

In this section we introduce the assumptions and principles adopted for the designed of the proposed heuristic for online NSP. In Section 5.1.1, we introduce the additional assumptions adopted for the heuristic design. In Section 5.1.2, we introduce the P2C principle. Finally, in Section, 5.1.3, we describe how we adapt the P2C principle to the proposed online approach for the NSP problem.

5.1.1 Additional assumptions adopted for the heuristic design

In this part of the work, to facilitate the evaluation of the algorithm, we adopt an additional assumption that all NSPRs are undirected path graphs.

Typically, in the literature, this NSPR topology is referred to as a SFC and we discussed the limits of this approach in Section 4.1.3. The algorithms described in this section consider this topology for NSPRs but note that the P2C principle used in the heuristic design can be extended to other NSPR topologies as well.

5.1.2 The "Power of Two Choices" principle

The P2C algorithm was introduced in [10] to solve a supermarket model. In this model, we have a set of customers that arrive in the system following a Poisson process. The system comprises a set of parallel servers and service times on these servers follows a exponential distribution. Customers choose d servers independently and uniformly at random from n and join the queue with the smallest number of customers. The analysis performed by the authors of [10] shows that having $d = 2$ choices leads to exponential improvements in the expected time users spend in the system over $d = 1$, whereas having $d = 3$ choices is only a constant factor better than $d = 2$.

5.1.3 Adaptation of the "Power of Two Choices" for Network Slice Placement

The adaptation of the P2C algorithm to solve the online approach of the NSP problem is described in the following. First, the customers are the VNFs. Instead of having arrivals of one customer, we have arrivals of batches of customers represented by the NSPRs.

The servers, in our case, are the DC servers. Our interest is to enable the acceptance of the highest number of NSPR as possible. The criteria for evaluating the 2 servers selected randomly is resource consumption calculation considering the VNF placement in the server but also the VNF chaining.

5.2 Proposed Network Slice Placement optimization heuristic

In this section, we present the proposed heuristic for solving the NSP problem. In Section 5.2.1, we present the description of the proposed algorithm. In Section 5.2.2, we present the method used for calculation of eligible servers for placement. Finally, on Section 5.2.3, we present the proposed selection policies for candidate placement servers.

5.2.1 Algorithm description

Algorithm 1 presents the pseudo code for the proposed heuristic to solve the online approach of the NSP problem. The algorithm is fed with the $NSPR$ to place, the PSN and also a $policy\_id$ parameter used to differentiate two possible candidate server selection policies. It returns the *status* of the $NSPR$ (Accepted or Rejected) and, if $status = Accepted$, it also returns the amount of bandwidth consumed C by the $NSPR$ and the values for x and y decision variables (see Section 4.3.2).

The algorithm performs a sequence of steps for each VNF v of the $NSPR$. Step 1 (line 5) calculates the set S' of feasible servers for the placement of VNF v. This is done by the procedure `getFeasibleServers` detailed in Section 5.2.2.

If there are feasible servers for the placement of VNF v, i.e. $S' \neq \varnothing$, the algorithm proceeds to Step 2 (line 7) that returns two candidate servers s_1 and s_2 for the placement of VNF v using the `getTwoCandidateServers` procedure detailed in Section 5.2.3. Finally, the algorithm proceeds to the Step 3 (lines 8-53) evaluate candidate servers s_1 and s_2. If one of these two servers was previously used to place VNF $v-1$, i.e., it is the same than $last\_s$, this server is chosen for the placement of VNF v as it is the optimal solution with 0 bandwidth consumption. Otherwise one path P_i needs to be calculated between $last\_s$ and s_i, for $i = 1, 2$. This is done using the `dijkstra` procedure (line 17). This latter procedure implements the Dijkstra shortest path algorithm and first tries to calculate the feasible path providing minimum bandwidth consumption between $last\_s$ and the evaluated server s_i, $i = 1, 2$. If the path satisfies the latency constraint between VNFs $v-1$ and v, it is returned by the `dijkstra` procedure. Otherwise the procedure tries to find a feasible path minimizing the latency between VNFs $v-1$ and v. If no feasible path is found the procedure returns $\varnothing$. The server allowing minimal embedding cost, i.e., the one which induces the use of less bandwidth resources, is chosen for the placement of VNF v.

Variables x and y, the available server and physical links capacities and the $NSPR$ placement cost C are updated accordingly. If for some VNF v no feasible servers or paths are found, there is a blocking and the algorithm returns *Rejected* status (lines 52 and 56).

5.2.2 Calculation of eligible servers for placement

Algorithm 2 presents the pseudo code for the procedure `getFeasibleServers`. In addition to the $NSPR$ and PSN this procedure receives as input the id v of the VNF to be placed, the id of the DC ($last\_dc$) and of the server ($last\_s$) in which the VNF $v-1$ was placed and outputs the set S' of feasible servers.

The procedure `getFeasibleServers` implements the different problem constraints described in Section 4.3 to filter eligible servers S' for the placement of VNF v. The implementation of these constraints is different according to the value of v.

The value $v = 1$ corresponds to the root VNF of the $NSPR$. The procedure first obtains the eligible DCs DC', i.e., the ones satisfying the access latency requirement of the $NSPR$. Two conditions are used to define whether a server s located in a DC $dc \in DC$ is eligible for the placement of VNF 1:

1. Server s has enough CPU and RAM resources to host VNFs 1 and 2. In this case, the available bandwidth capacity of the physical link connected to this server does not need to be checked since the server can host both VNFs 1 and 2 without using any bandwidth.

2. Server s has enough CPU and RAM resources available to host VNF 1 only. In this case, if server s were selected to place VNF 1, VNF 2 would need to be placed on a server $s' \neq s$. Hence, to be considered eligible, the DC link connected to server s must have enough available bandwidth capacity to host VL (1,2).

Assume now that $v = |V|$, where $|V|$ is the length of the NSPR in number of VNFs to be placed. This means that the last VNF of the $NSPR$ is to be placed. The procedure iterates through all servers in the network to find the feasible ones. The server $last\_s$ is eligible if it has enough CPU and RAM capacities to host VNF $|V|$. Other servers s located in $last\_dc$ will be eligible if they have enough CPU and RAM capacities to host VNF $|V|$ and if there is an intra-DC path between $last\_s$ and s with enough available bandwidth capacity to steer traffic between VNFs $|V|-1$ and $|V|$. A server s in a DC $dc \neq last\_dc$ is considered eligible if there is feasible path between $last\_s$ and s to map VL $(|V|-1, |V|)$, i.e., a path respecting the latency requirements and bandwidth requirements of the VL $(|V|-1, |V|)$.

If $1 < v < |V|$, the procedure also iterates through all servers in the network to find the eligible ones but the conditions to determine if a server is eligible are different from the case when $v = |V|$. Two conditions are used to define if server $last\_s$ is eligible: 1) $last\_s$ has enough CPU and RAM capacity to host VNF v and VNF $v+1$. In this case the available bandwidth capacity of the physical link connected to $last\_s$ does not need to be checked since the server can host both VNFs without using any bandwidth; 2) $last\_s$ has only enough CPU and RAM capacity to host VNF v and there is an intra-DC path between $last\_s$ and s with enough available bandwidth capacity to steer traffic between VNFs v and $v+1$. In this case, if server $last\_s$ were selected to place VNF v, VNF $v+1$ would need to be placed in a server $s' \neq last\_s$ hence the DC link connected to server $last\_s$ must have enough available bandwidth capacity to the VL between VNFs v and $v+1$.

Two conditions are used to define if a server $s \neq last\_s$ located in $last\_dc$ is eligible: s has enough CPU and RAM capacities to host VNF v and VNF $v+1$ and there is an intra-DC path between $last\_s$ and s with available bandwidth capacity to serve VL $(v-1,v)$; s has enough CPU and RAM capacities to host only VNF v. In the last case there must be an intra-DC path between $last\_s$ and s with enough available bandwidth capacity to serve VL $(v-1,v)$ and the DC link connected s must have enough available bandwidth capacity to serve VL $(v,v+1)$. For a server s in a DC $dc \neq last\_dc$ to be eligible, it must exist a feasible path between $last\_s$ and s to map VL $(v-1, v)$, i.e., there is a path that respects the latency requirements and bandwidth requirements of the VL $(v-1, v)$.

5.2.3 Selection Policies for candidate placement servers

Algorithm 3 present the pseudo-code for the the `getTwoCandidateServers` procedure. This procedure receives as arguments the set S' and the $policy\_id$ parameter and returns two candidate servers s_1 and s_2 selected using the server selection policy coded by $policy\_id$ parameter.

Algorithme 1 : Heuristic for NSP Optimization using P2C.

Data : $NSPR$, PSN,$policy\_id$
Result : $status$, C, x, y

$last\_s \leftarrow 0$, $C \leftarrow 0$;
$x_s^v = 0,\ \forall v \in V,\ \forall s \in S$;
$y_{(a,b)}^{(\bar{a},\bar{b})} = 0,\ \forall(\bar{a},\bar{b}) \in E,\ \forall(a,b) \in L$;
for $v \in V$ **do**
  $S' \leftarrow$ getFeasibleServers($NSPR$, PSN,v) `// Step 1`
  if $S' \neq \varnothing$ **then**
    $s_1,\ s_2$ = getTwoCandidateServers(S', v,$policy\_id$) `// Step 2`
    if $v = 1$ ***or*** $last\_s = s_1$ **then**
      $x_{v,s_1} = 1$;
      $last\_s = s_1$;
      $cap_{s_1}^j -= d_v^j,\ \forall j \in \{cpu, ram\}$;
    else if $last\_s = s_2$ **then**
      $x_{v,s_2} = 1,\ \forall s \in S' \setminus s_2$;
      $last\_s = s_2$;
      $cap_{s_2}^j -= d_v^j,\ \forall j \in \{cpu, ram\}$;
    else
      P_i =dijkstra($last\_s, s_i$), $i = 1, 2$;
      if $P_1 \neq \varnothing$ ***and*** $P_2 \neq \varnothing$ **then**
        $cost_i = |P_i| req_{v-1,v}^{bw},\ i = 1, 2$;
        if $cost_1 \leqslant cost_2$ **then**
          $x_{v,s_1} = 1$;
          $last\_s = s_1$;
          $cap_{s_1}^j -= d_v^j,\ \forall j \in \{cpu, ram\}$;
          $y_{(a,b)}^{(v-1,v)} = 1,\ \forall(a,b) \in P_1$;
          $cap_{(a,b)}^{bw} -= req_{(v-1,v)}^{bw},\ \forall(a,b) \in P_1$;
          $C+ = cost_1$;
        else
          $x_{v,s_2} = 1$;
          $last\_s = s_2$;
          $cap_{s_2}^j -= d_v^j,\ \forall j \in \{cpu, ram\}$;
          $y_{(a,b)}^{(v-1,v)} = 1,\ \forall(a,b) \in P_2$;
          $cap_{(a,b)}^{bw} -= req_{(v-1,v)}^{bw},\ \forall(a,b) \in P_2$;
          $C+ = cost_2$;
      else if $P_1 \neq \varnothing$ **then**
        $x_{v,s_1} = 1$;
        $last\_s = s_1$;
        $cap_{s_1}^j -= d_v^j,\ \forall j \in \{cpu, ram\}$;
        $cost_1 = |P_1| req_{v-1,v}^{bw}$;
        $y_{(a,b)}^{(v-1,v)} = 1,\ \forall(a,b) \in P_1$;
        $cap_{(a,b)}^{bw} -= req_{(v-1,v)}^{bw},\ \forall(a,b) \in P_1$;
        $C+ = cost_1$;
      else if $P_2 \neq \varnothing$ **then**
        $x_{v,s_2} = 1$;
        $last\_s = s_2$;
        $cap_{s_2}^j -= d_v^j,\ \forall j \in \{cpu, ram\}$;
        $cost_2 = |P_2| req_{v-1,v}^{bw}$;
        $y_{(a,b)}^{(v-1,v)} = 1,\ \forall(a,b) \in P_2$;
        $cap_{(a,b)}^{bw} -= req_{(v-1,v)}^{bw},\ \forall(a,b) \in P_2$;
        $C+ = cost_2$;
      else
        • Backtrack to initially available PSN capacities;
        $x \leftarrow \varnothing$; $y \leftarrow \varnothing$; $status = Rejected$;
        return ;
  else
    • Backtrack to initially available PSN capacities ;
    $x \leftarrow \varnothing$; $y \leftarrow \varnothing$; $status = Rejected$; **return** ;
$status = Accepted$;

Algorithme 2 : Calculate feasible servers.

Data : $NSPR$, PSN,n,$last_s$,$last_{dc}$
Result : S'

if $n = 1$ **then**
  $DC' \leftarrow \{dc \in DC : \alpha_{dc} \leqslant \alpha_{max},\ \forall s \in S'\}$;
  $S' \leftarrow \varnothing$;
  for $dc \in DC'$ **do**
    for $s \in S_{dc}$ **do**
      if $cap_s^j \geqslant d_n^j + d_{n+1}^j,\ j \in \{cpu, ram\}$ **then**
        $S' = S' \cup s$;
      else if $cap_s^j \geqslant d_n^j,\ j \in \{cpu, ram\},$ ***and*** $cap^{bw}_{(s,SW_{dc})} \geqslant req^{bw}_{(n,n+1)}$ **then**
        $S' = S' \cup s$;
else if $n = |N|$ **then**
  for $dc \in DC$ **do**
    if $dc = last_{dc}$ **then**
      for $s \in S_{dc}$ **do**
        if $s = last_s$ **then**
          if $cap_s^j \geqslant d_n^j,\ j \in \{cpu, ram\}$ **then**
            $S' = S' \cup s$;
        else
          if $cap_s^j \geqslant d_n^j,\ j \in \{cpu, ram\},$ ***and*** $cap^{bw}_{(last_s,SW_{dc})} \geqslant req^{bw}_{(n-1,n)}$ ***and*** $cap^{bw}_{(SW_{dc},s)} \geqslant req^{bw}_{(n-1,n)}$ **then**
            $S' = S' \cup s$;
    else
      $L \leftarrow \{(a,b) \in L : cap^{bw}_{(a,b)} \geqslant req^{bw}_{(n-1,n)}\}$;
      $P \leftarrow$ dijkstra(PSN,$last\_dc$,dc) ;
      if $P \neq \varnothing$ ***and*** $\sum_{(a,b)\in P} \delta_{(a,b)} \leqslant req^{lat}_{(n-1,n)}$ **then**
        for $s \in S_{dc}$ **do**
          if $cap_s^j \geqslant d_n^j,\ j \in \{cpu, ram\},$ ***and*** $cap^{bw}_{(SW_{dc},s)} \geqslant req^{bw}_{(n-1,n)}$ **then**
            $S' = S' \cup s$;
else
  for $dc \in DC$ **do**
    if $dc = last_{dc}$ **then**
      for $s \in S_{dc}$ **do**
        if $s = last_s$ **then**
          if $cap_s^j \geqslant d_n^j + d_{n+1}^j,\ j \in \{cpu, ram\}$ **then**
            $S' = S' \cup s$;
          else if $cap_s^j \geqslant d_n^j,\ j \in \{cpu, ram\},$ ***and*** $cap^{bw}_{(SW_{dc},s)} \geqslant req^{bw}_{(n,n+1)}$ **then**
            $S' = S' \cup s$;
        else
          if $cap_s^j \geqslant d_n^j + d_{n+1}^j,\ j \in \{cpu, ram\}$ ***and*** $cap^{bw}_{(SW_{dc},s)} \geqslant req^{bw}_{(n-1,n)}$ **then**
            $S' = S' \cup s$;
          else if $cap_s^j \geqslant d_n^j,\ j \in \{cpu, ram\}$ ***and*** $cap^{bw}_{(last\_s,SW_{dc})} \geqslant req^{bw}_{(n-1,n)}$ ***and***
            $cap^{bw}_{(SW_{dc},s)} \geqslant req^{bw}_{(n-1,n)} + req^{bw}_{(n,n+1)}$ **then**
    else
      $L \leftarrow \{(a,b) \in L : cap^{bw}_{(a,b)} \geqslant req^{bw}_{(n-1,n)}\}$;
      $P \leftarrow$ dijkstra(PSN,$last\_dc$,dc) ;
      if $P \neq \varnothing$ ***and*** $\sum_{(a,b)\in P} \delta_{(a,b)} \leqslant req^{lat}_{(n-1,n)}$ **then**
        for $s \in S_{dc}$ **do**
          if $cap_s^j \geqslant d_n^j + d_{n+1}^j,\ j \in \{cpu, ram\},$ ***and*** $cap^{bw}_{(SW_{dc},s)} \geqslant req^{bw}_{(n-1,n)}$ **then**
            $S' = S' \cup s$;
          else if $cap_s^j \geqslant d_n^j,\ j \in \{cpu, ram\}$ ***and*** $cap^{bw}_{(SW_{dc},s)} \geqslant req^{bw}_{(n-1,n)} + req^{bw}_{(n,n+1)}$ **then**
            $S' = S' \cup s$;

Policy 1

This server selection policy chooses s_1 and s_2 completely randomly from S'. If $|S'| = 1$, we set $s_2 = s_1$.

Policy 2

This is a more intelligent policy. It also selects s_1 and s_2 randomly but preferentially from CCPs or CDCs. The procedure will try to select s_1 and s_2 from a subset of S' containing only servers located in CCPs. If it is not possible it will try servers located in the CDCs and in the last case it will try to select EDCs servers. The aim of this policy is to save EDCs capacities when possible since these are critical resources.

Algorithme 3 : Get two candidate servers.

```
   Data : S',policy
   Result : s_1, s_2
 1 if policy = 1 then
 2 |   i ← RAND[1 : |S'|]; s_1 ← S'_i; S ← S' \ {s_1};
 3 |   if S' ≠ ∅ then
 4 |   |   i ← RAND[1 : |S'|]; s_2 ← S_i;
 5 |   else
 6 |   |   s_2 ← s_1;
 7 else if policy = 2 then
 8 |   if tmp = {s ∈ S' : DT_s = CCP} ≠ ∅ then
 9 |   |   i ← RAND[1 : |tmp|]; s_1 tmp_i; S ← tmp \ {s_1};
10 |   |   if tmp ≠ ∅ then
11 |   |   |   i ← RAND[1 : |tmp|]; s_2 tmp_i;
12 |   |   else if tmp ← {{s ∈ S' : DT_s = CDC} ≠ ∅ then
13 |   |   |   i ← RAND[1 : |tmp|]; s_2 ← tmp_i;
14 |   |   else if tmp ← {{s ∈ S' : DT_s = EDC} ≠ ∅ then
15 |   |   |   i ← RAND[1 : |tmp|]; s_2 ← tmp_i;
16 |   |   else
17 |   |   |   s_2 ← s_1 ;
18 |   else if tmp ← {s ∈ S' : DT_s = CDC} ≠ ∅ then
19 |   |   i ← RAND[1 : |tmp|]; s_1 ← tmp_i; S ← tmp \ {s_1};
20 |   |   if tmp ≠ ∅ then
21 |   |   |   i ← RAND[1 : |tmp|]; s_2 ← tmp_i;
22 |   |   else if tmp ← {{s ∈ S' : DT_s = EDC} ≠ ∅ then
23 |   |   |   i ← RAND[1 : |tmp|]; s_2 ← tmp_i;
24 |   |   else
25 |   |   |   s_2 ← s_1 ;
26 |   else if tmp ← {s ∈ S' : DT_s = EDC} ≠ ∅ then
27 |   |   i ← RAND[1 : |tmp|]; s_1 = tmp_i; S ← tmp \ {s_1};
28 |   |   if tmp ≠ ∅ then
29 |   |   |   i ← RAND[1 : |tmp|]; s_2 ← tmp_i;
30 |   |   else
31 |   |   |   s_2 ← s_1 ;
```

5.3 Experiments, evaluation results, and discussions

We present in this section the implementation and the numerical experiments carried out to evaluate the proposed algorithms for the online approach of the NSP problem. In Section 5.3.1, we present the implementation details and experimentation settings. In Section 5.3.2, we describe the tested algorithms. In Section 5.3.3, we present the simulation scenarios used. In Section 5.3.4, we discuss the network load calculation. In Section 5.3.5, we present the evaluation metrics. Finally, in Section 5.3.6, we discuss the evaluation results.

5.3.1 Implementation details and experimentation settings

Proposal architecture description

The architecture of the proposed NSP solution is given in the Figure 5.1. The NSPR generator is used to generate NSPR arrivals. The PSN database stores the data about the available resources of the PSN. NSPR requirements and PSN available resources data are used as input by the Placement module. This latter implements the ILP for online NSP and P2C algorithms.

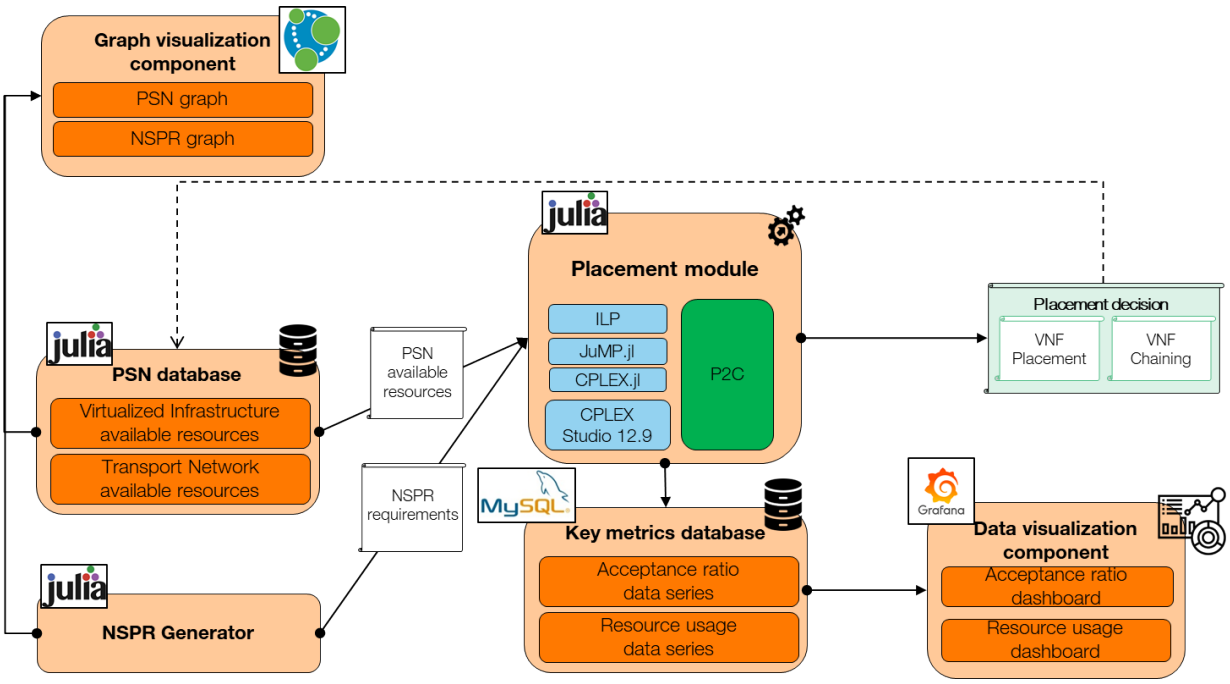


Figure 5.1: Heuristic-based NSP solution architecture.

Both algorithms calculate: i) a VNF placement decision, that is, where each VNF of the NSPR is to be placed and ii) a VNF chaining decision, that is, which paths in the network to use to interconnect the different VNFs. The Placement module can be configured to use one of the Placement algorithms or both if comparison of Placement solutions is necessary.

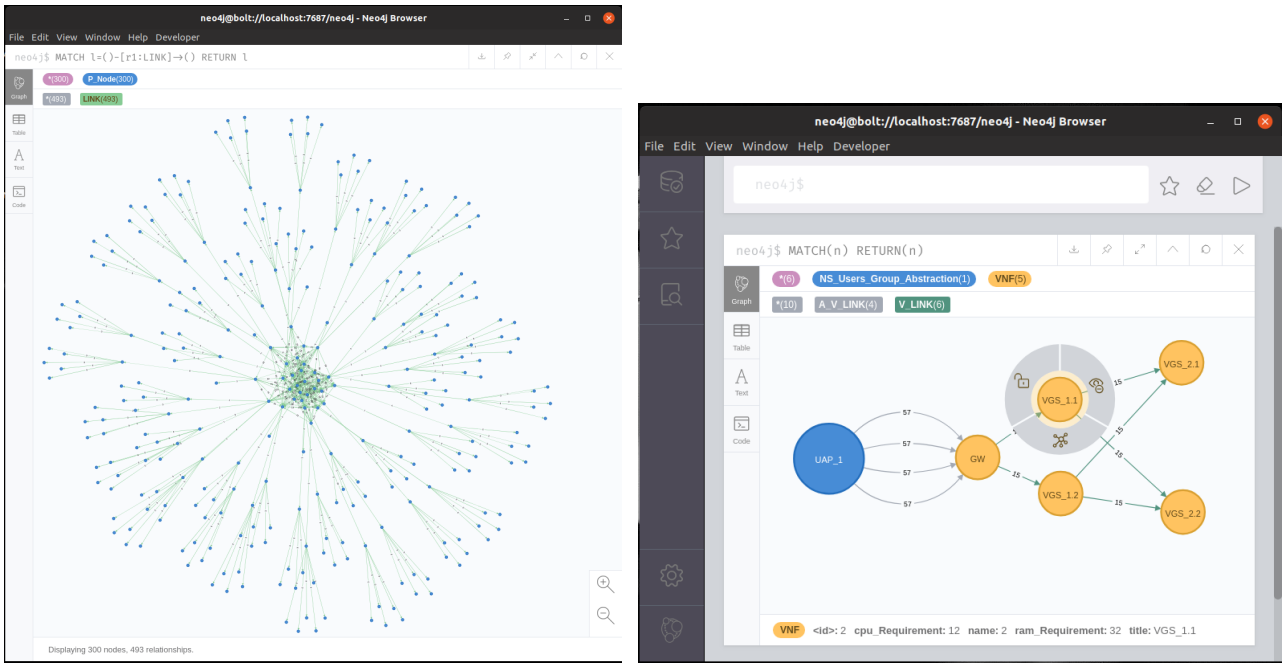
Once the calculation of the Placement decision is done for one NSPR, an update of the available resources on the PSN is made and some key performance metrics are registered in form of data series the Key metrics database: the acceptance ratio of network slices and the resource usage. These time series are used by the Data visualization component to build two dashboards: an acceptance ratio dashboard and a resource usage dashboard. Both dashboards are used to show the performance of the algorithms in real time. Finally, the graph visualization component is used to allow the visualisations of the PSN and NSPR graphs.

Proposal implementation description and tools

We have implemented the proposed NSP solution and we describe below the different tools used to implement the different components of the proposed solution:

- **Julia:** The Julia version 1.1 is used to implement different elements of our solution [129]; The ILP-based Placement algorithm is developed using the JuMP.jl package that allows the development of a generic ILP formulation that can be solved with any linear solver. The CPLEX.jl package is an interface to the CPLEX solver used to solve the ILP. The classes and functions used to implement the P2C heuristic were designed and implemented from scratch using only default Julia packages.
- **CPLEX:** The default branch-and-bound algorithm from `ILOG` CPLEX [128] in its 12.9 version is used to solve the ILPs;
- **Neo4j:** A Neo4j graph database represents and and displays the PSN graph and the NSPR graph and its requirements;
- **MySQL:** We use the MySQL database manager system to implement the Key metrics database with one table for the Acceptance ratio data series and another one for the Resource usage data series;
- **Grafana:** We use the Grafana tool to implement the Data visualization component in which we represent two dashboards using Key metrics MySQL database as data-source.

The experiments performed were executed in a 2x6 cores @2.95Ghz 96GB machine. Examples of PSN and NSPR are displayed by using the graph visualization component are shown in Figure 5.2(a) and 5.2(b), respectively.



(a) PSN view using the Graph-based Visualization Component

(b) NSPR view using the Graph visualization component

Figure 5.2: Example of view using Graph-based Visualization component.

Physical Substrate Network settings

We considered a PSNs that could reflect that of an operator such as Orange, see [127]. In this network, 3 types of DCs match our description made in Section 4.1.

Each CDC is connected to 3 EDCs which are 100km away. CDCs are interconnected and connected to a CCP that is 300 km away. Tables 5.1 and 5.2 summarize the DCs and transport links properties. The CPU and RAM capacities of each server are 50 and 300 units, respectively. Latency is computed by considering the speed of light in fiber.

Data center type	Number of data centers	Number of servers per DC	Intra data center links bandwith capcity
CCP	1	16	100 Gbps
CDC	5	10	100 Gbps
EDC	15	4	10 Gbps

Table 5.1: DCs description.

Network Slice Placement Requests settings

Tables 5.3 and 5.4 show the network resources and latency requirements for the three NSPR classes taken into account: Best Effort (BEF), Ultra Reliable Low Latency Communications (URLLC) and Enhanced Mobile Broadband (EMBB).

	CCP	CDC	EDC
CCP	NA	100 Gbps	100 Gbps
CDC	100 Gbps	100 Gbps	100 Gbps
EDC	10 Gbps	10 Gbps	10 Gbps

Table 5.2: Transport links capacities.

NSPR class	CPU requested by each VNF	RAM requested by each VNF	Bandwidth requested by each VL
Best Effort	10	60	1
URLLC	15	90	1
EMBB	25	150	2

Table 5.3: Resource requirements by NSPR class.

NSPR class	Access latency requirement	Latency requirement for VL1	Latency requirement for VL2	Latency requirement for VL3	Latency requirement for VL4
URLLC	0.03ms	0.33ms	0.33ms	0.33ms	0.33ms
EMBB	0.07ms	0.33ms	1ms	1ms	1ms
Best Effort	0.07ms	0.67ms	1ms	1.33ms	1.33ms

Table 5.4: Latency requirements by NSPR class.

5.3.2 Tested algorithms

We compare four algorithms: two versions of the ILP introduced in Section 4.3 and applied to the online approach of the NSP problem as described in Section 4.4.2 (ILP 1 and 2 for objective functions described in Sections 4.3.4 and 4.3.4, respectively) and two versions of the proposed heuristic approach introduced in Section 5.2 (P2C 1 and P2C 2 for server selection policies described in Sections 5.2.3 and 5.2.3, respectively).

5.3.3 Simulation scenarios

We consider three simulation scenarios named BEF, URLLC, EMBB in which all NSPRs to be placed are of the same class and one simulation scenario named MIX in which we have a percentage of NSPRs of each class: 67% of BEF, 22 % of EMBB and 11% of URLLC. We set simulation duration to 2000 time units.

5.3.4 Network Load calculation

We use the Stationary Network Load model defined in Section 4.3.6 to calculate the arrival rates of NSPRs (λ^k) in the different network load conditions used: underload ($\rho < 1$), critical ($\rho = 1$), and overload ($\rho > 1$). Network loads are calculated using CPU resource since it is the scarcest resource in the PSNs. Generally speaking, we set $1/\mu^k = 100$ time units for all $k \in K$, the set of slice classes.

5.3.5 Evaluation metrics

We consider 3 performance metrics:

1. **Average execution time:** the average execution time in seconds needed to place 1 NSPR;
2. **Average final blocking ratio**: the average of the final blocking ratios. The final blocking rations are calculated as $\frac{\texttt{\# accepted NSPRs}}{\texttt{\# of NSPR's arrivals}}$ at the end of each execution. We average the results for 100 executions;
3. **Resource utilization:** the amount CPU, RAM and bandwidth consumed.

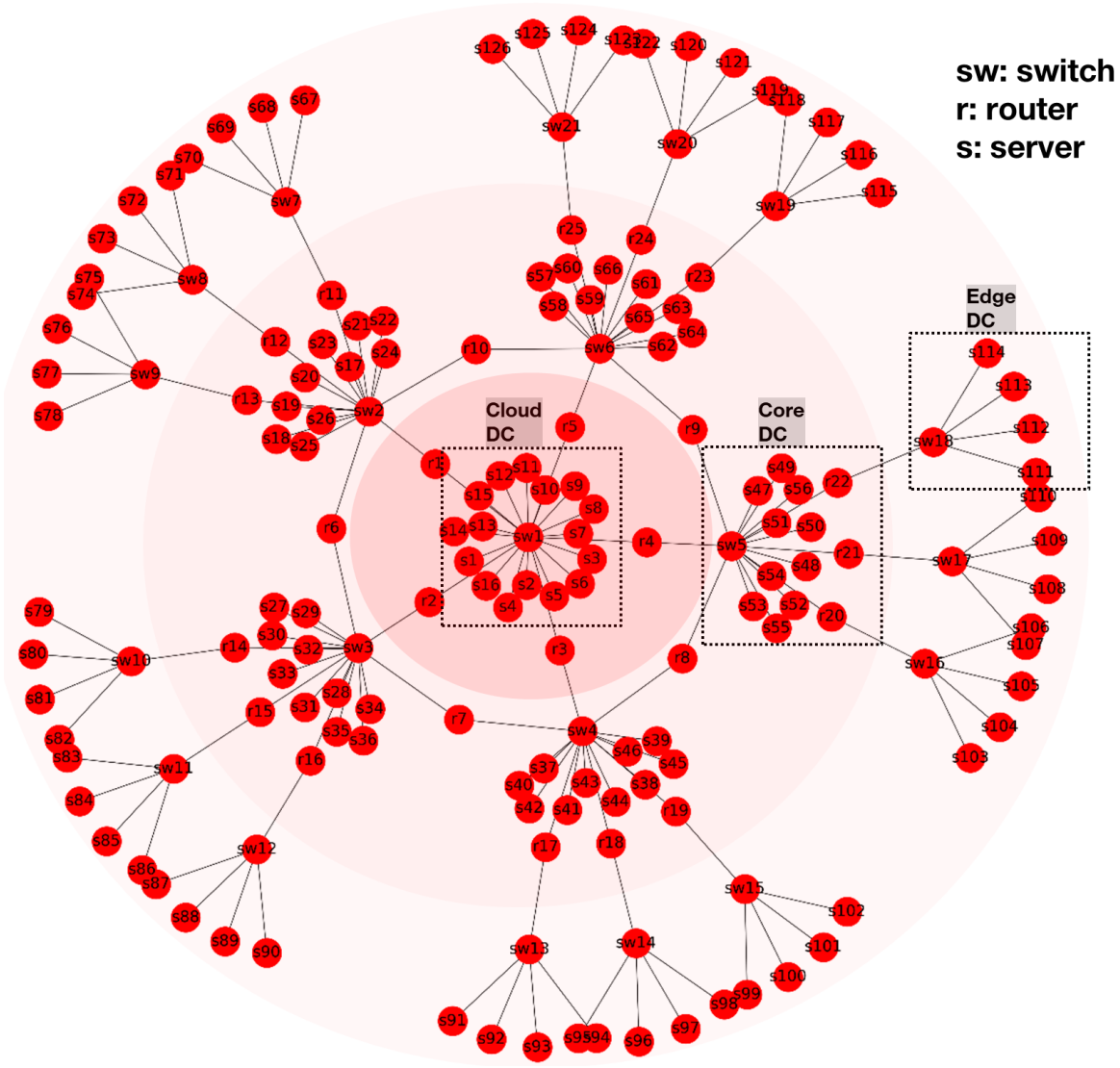


Figure 5.3: Used PSN topology.

5.3.6 Evaluation results and discussion

Average execution time evaluation

The average execution times in function of the number of servers in the PSNs is given in Figure 5.4. Starting from a PSNs with 126 servers as described in Section 5.3.1 and captured in Figure 5.3, we generated new PSNs settings by doubling the number of servers in each DC. The evaluation results confirmed our expectations showing that the average execution time grows much faster for the ILPs than for the heuristics. In the scenario with 16128 nodes the execution times are 9.8 and 12.5 seconds for the ILPs 1 and 2 respectively and 2.17 and 1.96 seconds for P2C 1 and 2 respectively.

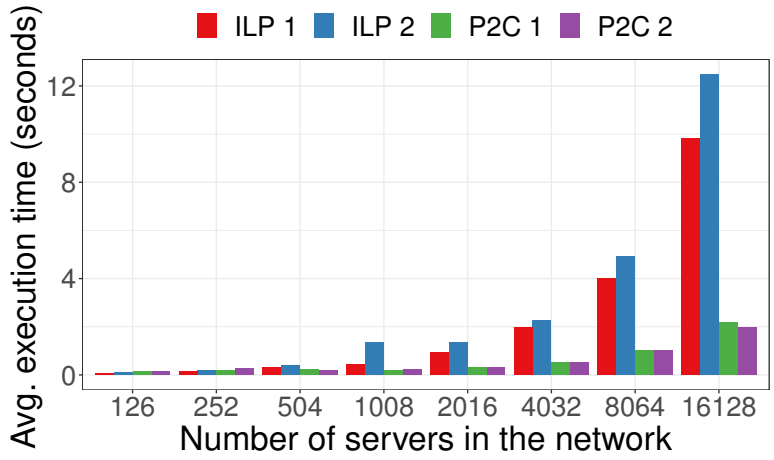
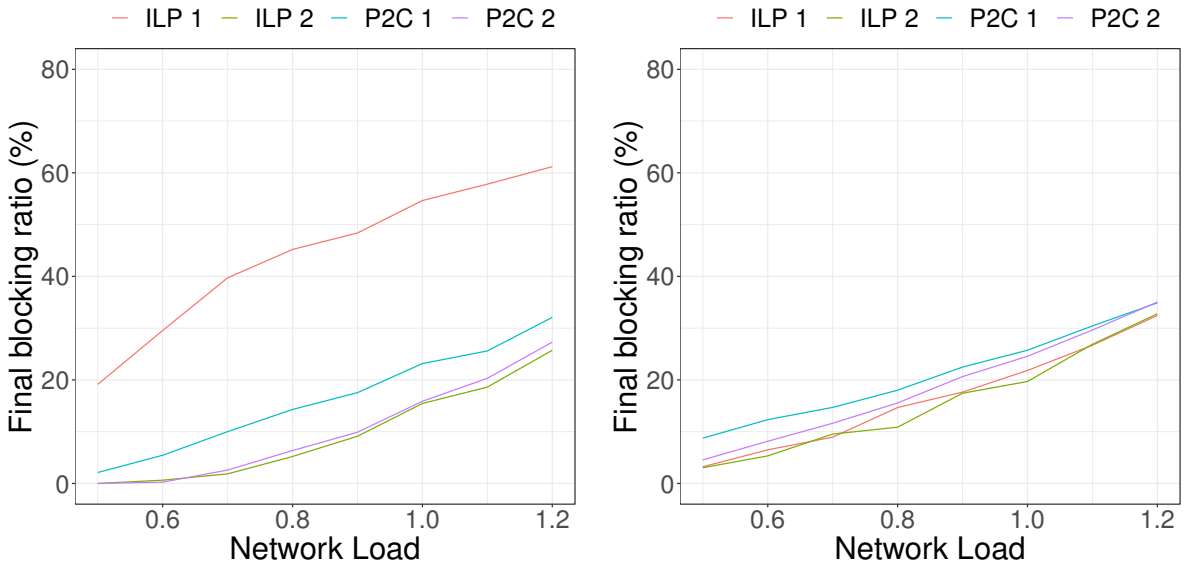


Figure 5.4: Average execution time evaluation.

Average final blocking ration evaluation

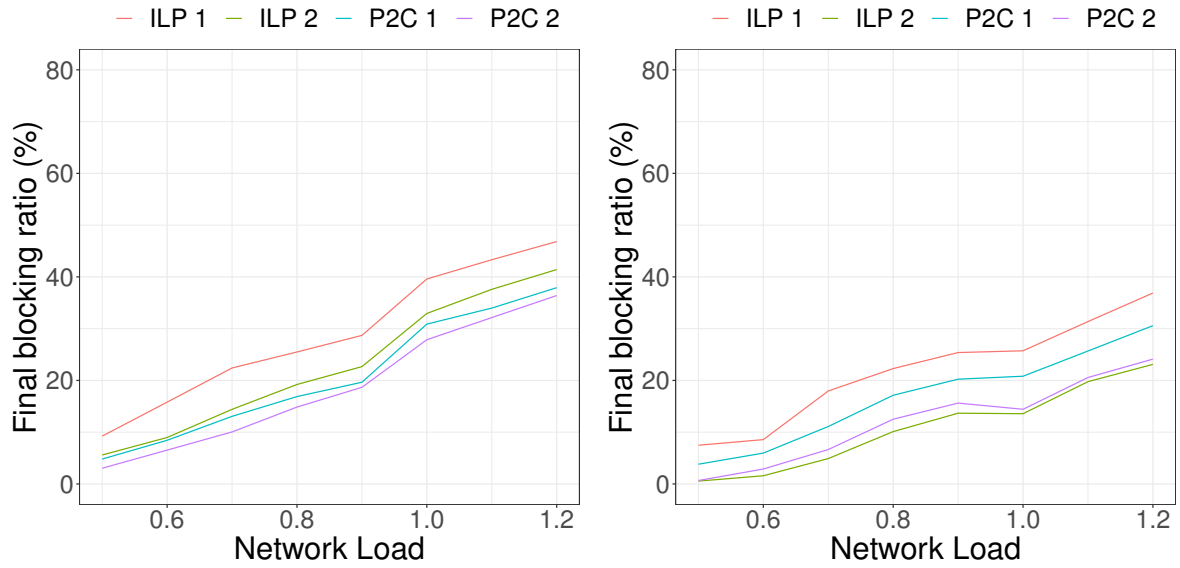
Figures 5.5 and 5.6 shows the blocking ratios results.



(a) BEF simulation scenario.

(b) EMBB simulation scenario.

Figure 5.5: Average final blocking ratios: BEF and EMBB.



(a) URLLC simulation scenario.

(b) MIX simulation scenario.

Figure 5.6: Average final blocking ratios: URLLC and MIX.

We expected ILP 2 would provide the best acceptance ratio results which was not confirmed: ILP 2 obtains the best acceptance ratio in scenarios BEF, EMBB, and MIX, but in URLLC scenario, P2C 1 and 2 provide the best acceptance ratio.

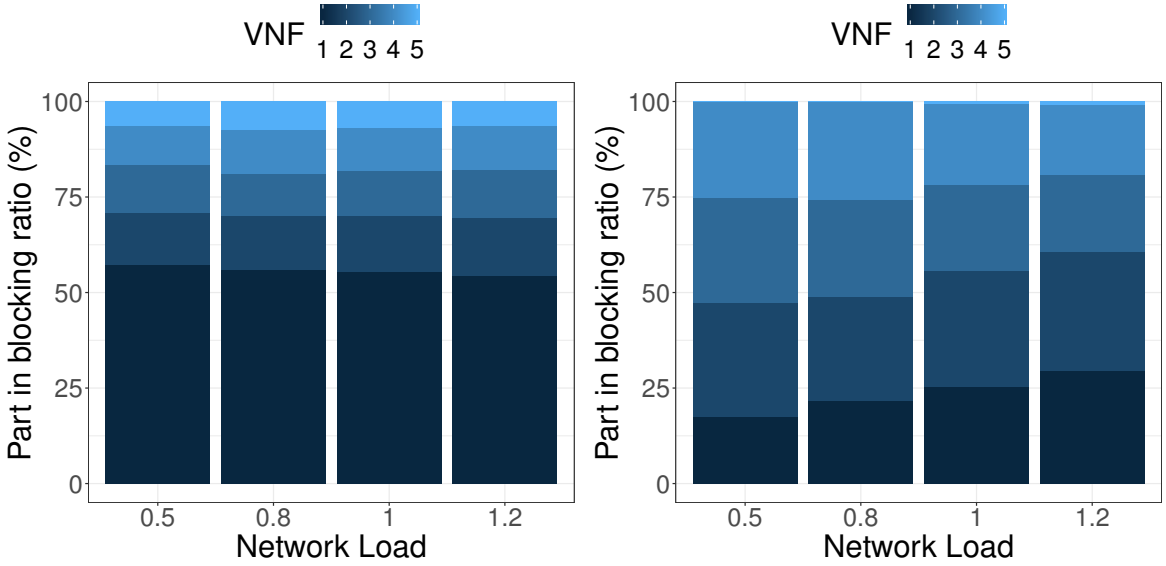
We explain this by a critical load balancing as heuristics usually perform better load balancing (see Figures 5.8a-5.9b). Also, ILP 1 has good performance in the EMBB scenario as shown in Figure 5.5(b) as the bandwidth here is a critical resource so ILP 1 resolve with optimal bandwidth consumption by concentrating the VNFs on the same machines. However ILP 1 have higher final blocking ratio than P2C in BEF, MIX and URLLC simulation scenarios (see Figures 5.5(a), 5.6(a) and 5.6(b)) since the strategy of concentrating VNFs in the same machines does not fit well.

In the BEF simulation scenario, since we have only BEF NSPRs which have low CPU and RAM requirements, the ILP 1 strategy ends up concentrating all the VNFs of each NSPR in the same machine. Since the first VNF of the NSPR is always placed in a EDC due to the access latency requirements, the optimal solution considering bandwidth minimization is to concentrate all the VNFs in the same machines on a EDC. This strategy leads to a overload of EDC servers and without EDC servers available the PSNs cannot accept new NSPRs. In URLLC and MIX simulation scenarios, the same problem happens but with lower intensity since the CPU and RAM required by the VNFs in these cases are higher than in the BEF simulation scenario which reduces the concentration of VNFs in the same machine.

The random selection policies of candidate placement servers implemented in P2C 1 and 2 helps improve load balancing and avoids EDC overload. P2C 2 has the best performance when compared with P2C 1 as it seeks to offload EDC critical resources by placing VNFs in CCP or CDC servers preferentially. Figures 5.7(a) and 5.7(b) present how much each VNF participates in the blocking ratio obtained with P2C 1 and P2C 2 respectively in the URLLC scenario. We see that P2C 2 highly reduces the amount of blocking in the root VNF of the NSPRs and provides a well balanced blocking profile.

Resource utilization evaluation

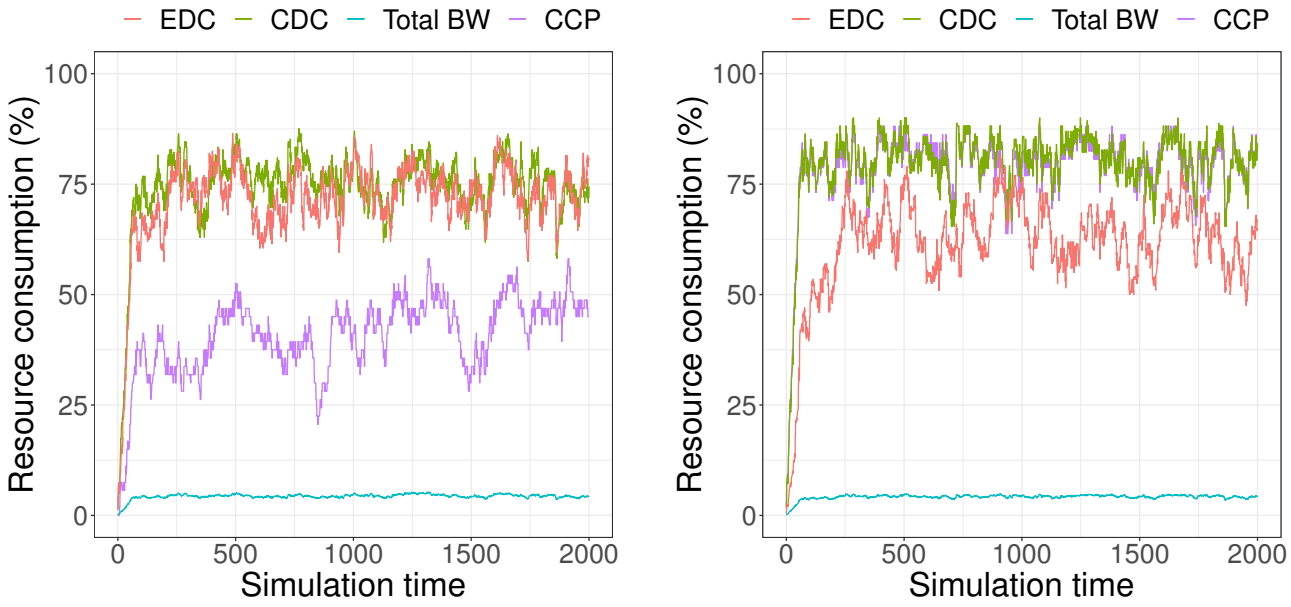
Figures 5.8 and 5.9 show the amount of resources consumed during one simulation for P2C 1, P2C 2 and ILP algorithms, respectively. In this simulation, we consider the URLLC simulation scenario with a critical network load ($\rho = 1$). As expected, the graphics show that the P2C heuristics distributes better the load among DCs and consumes more bandwidth resources. P2C 1 distributes almost equally the load between EDCs, CDCs and the CCP, while P2C 2 concentrates the load on CCP and CDCs to offload EDCs.



(a) P2C 1.

(b) P2C 2.

Figure 5.7: Blocking ratio.

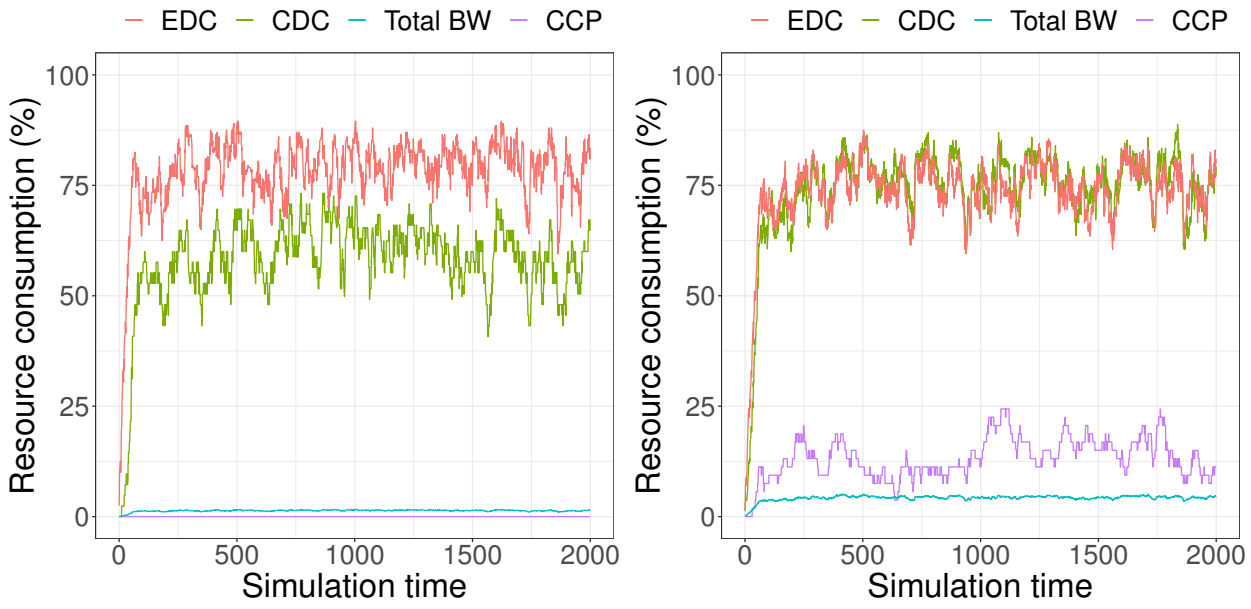


(a) P2C 1 Resource consumption.

(b) P2C 2 Resource consumption.

Figure 5.8: Resource consumption: P2C.

The ILP 1 concentrates the load on EDCs and CDCs to minimize bandwidth consumption, while the ILP 2 also uses uses CCP resources. The total bandwidth consumption obtained with the ILP 1 is minimum since it calculates solutions with optimal bandwidth consumption. ILP 2 objective function leads to solution higher bandwidth consumption than ILP 1.



(a) ILP 1 Resource consumption.

(b) ILP 2 Resource consumption.

Figure 5.9: Resource consumption: ILP.

Conclusion

In this chapter, we presented the second contribution of this PhD thesis, namely, *Optimizing Large Scale Network Slice Placement: a Heuristic using the Power of Two Choices*.

We have proposed an efficient heuristic to solve the online approach of the proposed NSP problem. This heuristic gather three main contributions: i) adaption to NSPRs on large scale networks, ii) integration of Edge-specific and URLLC-based QoS constraints (E2E latency), iii) reuse of the strength of P2C algorithm to implement selection policies. The evaluation results show that the heuristic yields good solutions within a small execution time (1.96s for a PSNs of 16128 nodes). The selection policies improve load balancing and reduce load of Edge DCs which improves the acceptance ratio in most simulation scenarios comparing to an online ILP-based placement algorithm.

From an operational perspective, heuristic approaches are more suitable than ILP as they yield faster placement results. This is very important for operational networks because traffic conditions are fluctuating and placement response time is an important performance indicator in the customer relationship. The main drawback of heuristic approaches is that they give sub-optimal solutions. Another difficulty is that multi-objective heuristics are often based on evolutionary algorithms (e.g., genetic algorithms) that can take time to converge. To remedy these problems, we study in the next chapter the use of ML methods.

Chapter 6

Automating Multi-objective Network Slice Placement with Machine Learning: a Heuristically Assisted Deep Reinforcement Learning solution

Contents

Introduction

In this chapter, we present the third and final contribution of this PhD thesis. We start, in Section 6.1, by presenting the definitions and assumptions adopted. In Section 6.2, we state the online multi-objective NSP problem considered in this chapter, and in Section 6.3, we formalize this problem mathematically. We discuss the modeling of NSPR arrival process in Section 6.3.2, and the network load modeling in Section 6.3.3. We introduce the proposed DRL framework for the online multi-objective NSP problem in Section 6.4. The adaptation of such framework and introduction of a heuristic function to control its convergence is discussed in Section 6.5. Finally, in Section 6.6, we present the experiments performed to evaluate the different DRL-based algorithms proposed in this chapter considering three different network load scenarios: stationary network load scenario (see Section 6.6.2), cycle-stationary network load scenario (see Section 6.6.3) and non-stationary network load scenario (see Section 6.6.4). We close the chapter with some concluding remarks.

6.1 Definitions and assumptions

We build on the definitions and assumptions about the PSN and NSPRs made in the Section 4.1. But, differently from Chapters 4 and 5, in this chapter we do not consider latency constraints. The inclusion of latency in the proposed DRL algorithm remains an open topic of study. As done in Chapter 5, we also adopt in this Chapter the additional assumption that all NSPRs are undirected path graphs.

6.2 Online multi-objective Network Slice Placement problem statement

The online multi-objective NSP problem studied in this chapter is stated as follows.

- *Given:* a NSPR graph $G_v = (V, E)$ arrived to be placed in arrival date $t_a \in \{1, ..., T\}$ and exiting the network in exit date $t_e \in \{t_a, ..., T\}$; a PSN graph $G_s = (N, L)$,
- *Find:* a mapping $G_v \rightarrow \bar{G}_s = (\bar{N}, \bar{L})$, $\bar{N} \subset N$, $\bar{L} \subset L$,
- *Subject to:* the VNF CPU requirements $req_v^{cpu}, \forall v \in V$, the VNF RAM requirements $req_v^{ram}, \forall v \in V$, the server CPU available capacity $cap_s^{cpu}, \forall s \in S$, the server RAM available capacity $cap_s^{ram}, \forall s \in S$, the physical link bandwidth available capacity $cap_{(a,b)}^{bw}, \forall (a, b) \in L$, the physical link bandwidth available capacity $cap_{(a,b)}^{bw}, \forall (a, b) \in L$,
- *Objective:* optimize jointly the NSPR acceptance ratio (maximize), the total resource consumption (minimize), and the node load balancing (maximize).

6.3 Mathematical modeling for the online multi-objective Network Slice Placement problem

In this section, we present the proposed mathematical modeling for the multi-objective NSP problem studied in this chapter. We use in this Chapter the same formulation for the variables, problem constraints, and for the two first objective functions introduced in Section 4.3, but we use a new objective function which is described in Section 6.3.1. We introduce the NSP arrival process and network load models used in this chapter on Sections 6.3.2 and 6.3.3 respectively.

6.3.1 Multi-objective objective function formulation

Equation (6.1) gives the maximization of node load balancing objective function where M_s^{cpu} and M_s^{ram} represent respectively the maximum amount of CPU and RAM resources on server s (see Section 4.3.2 and Table 4.2 for notation description).

$$\max_{x,y} \sum_{r \in R} \sum_{s \in S} \sum_{v \in V^r} x_s^v \left(\frac{cap_s^{cpu}}{M_s^{cpu}} + \frac{cap_s^{ram}}{M_s^{ram}} \right). \tag{6.1}$$

We consider a multi-objective optimization problem. The objective function of this problem can be formulated as linear combination of the three optimization objectives described in Section 6.2 as following:

$$\max_{x,y,z} c_1 \sum_{r \in R} \frac{z^r}{|R|} - c_2 \sum_{r \in R} \sum_{(\bar{a},\bar{b}) \in E^r} \sum_{(a,b) \in L} y_{(a,b)}^{(\bar{a},\bar{b})} req_{(\bar{a},\bar{b})}^{bw} + c_3 \sum_{r \in R} \sum_{n \in N} \sum_{v \in V^r} x_n^v \left(\frac{cap_n^{cpu}}{M_n^{cpu}} + \frac{cap_n^{ram}}{M_n^{ram}} \right) \tag{6.2}$$

for some weight coefficients $c_i > 0$, $i = 1, 2, 3$.

Note that the online approach of the NSP problem treated in this PhD thesis consider $|R| = 1$, i.e., the placement of one NSPR at a time (see Section 4.4.2).

Optimally solving this problem using ILP-based techniques would require a proper definition of the values for the three objective function coefficients and also affording for the long convergence times of ILP on hard

problems. We propose instead a faster and more scalable solution based on DRL controlled by a heuristic method.

6.3.2 Network Slice Placement Request arrival process modeling

In the work developed in this Chapter we assume that NSPRs can follow either a static arrival process (see Definition 4.3.2) or a dynamic arrival process defined in this section.

Definition 6.3.1 (Dynamic NSPR arrival process) *Let J be the set of resources in the network (i.e., CPU, RAM, bandwidth). Let $\mathcal{K}$ be the set of NSPR classes. Let $t = 1, .., T$ be a time horizon of T time steps. We name $\lambda^k(t)$ the rate parameter of the Poisson distribution describing the number of arrivals of NSPRs belonging to class k in time step t, i.e., $f(a; \lambda^k(t)) = Pr(X^k = a) = \frac{(\lambda^k(t))^a e^{-\lambda^k(t)}}{a!}$ in which a is the number of arrival occurrences and X^k is the random variable described by the Poisson distribution and f is the probability mass function.*

Remark

It is worth noting that the Dynamic NSPR arrival process is described by a non-stationary Poisson process.

6.3.3 Network Load modeling

In addition to the stationary network load model introduced in Section 4.3.6, in this Section we introduce two network modeling strategies we use in this chapter, namely the Cycle-stationary network load model (Definition 6.3.2) and the Non-stationary network load model (Definition 6.3.3).

Definition 6.3.2 (Cycle-stationary network load model) *Let $\mathcal{K} \subset \mathbb{N}$ be the set of NSPR classes. Let J be the set of resources in the network (i.e., CPU, RAM, bandwidth). We consider a periodic arrival rate $\lambda^k(t)$ for class k in $\mathcal{K}$ given by*

$$\lambda^k(t) = f^k \sin^2\left(\frac{\pi t}{\tau^k}\right) \tag{6.3}$$

where τ^k is the period of $\lambda^k(t)$ in time units and f^k is a parameter used to control the amplitude of $\lambda^k(t)$. We then adapt Equation (4.17) to compute the network load for NSPR class k and resource j as $\rho_j^k(t) = \frac{1}{C_j}\frac{\lambda^k(t)}{\mu^k}A_j^k$.

Remark

It is worth noting that to preserve $\rho_j^k(t)$ in the $[0, 1]$ interval, f^k must be between 0 and $\frac{C_j \mu^k}{A_j^k}$.

Definition 6.3.3 (Non-stationary network load model) *Let $\mathcal{K} \subset \mathbb{N}$ be the set of NSPR classes. Let J be the set of resources in the network (i.e., CPU, RAM, bandwidth). Let T be a time horizon divided into $T_1, ..., T_n$ disjoint time horizons with $|T_i|$ time units each for i in $\{1, .., n\}$. We consider a static arrival rate λ_i^k associated with each time horizon T_i in such a way that $\lambda^k(t) = \lambda_i^k$ for all $t \in T_i$ and for all i in $\{1, .., n\}$. We then adapt Equation (4.17) to compute the network load for NSPR class k and resource j in each time horizon as $\rho_{i,j}^k = \frac{1}{C_j}\frac{\lambda_i^k}{\mu^k}A_j^k$.*

6.4 Deep Reinforcement Learning for the online multi-objective Network Slice Placement problem

In this section, we describe the DRL agent we propose to solve the online multi-objective NSP problem introduced in Section 6.2. We give a general view of the DRL framework on Section 6.4.1 and describe the DRL framework's elements, the Policy, in Section 6.4.2, the State, in Section 6.4.3 and the Reward, in Section 6.4.4.

6.4.1 Deep Reinforcement Learning framework

Figure 6.1 presents the proposed DRL framework. The state contains the features of the PSN and NSPR to be placed. A valid action is, for a given NSPR graph $G_v = (V, E)$, a sub graph of the PSN graph $\bar{G}_s \subset \bar{G}_s = (N, L)$ to place the NSPR that does not violate the problem constraints described in Section 6.2. The reward evaluates how good is the computed action with respect to the optimization objectives described in Section 6.3. DNNs are trained to: i) calculate optimal actions for each state (i.e., placements with maximal rewards), ii) calculate the State-value function used in the learning process.

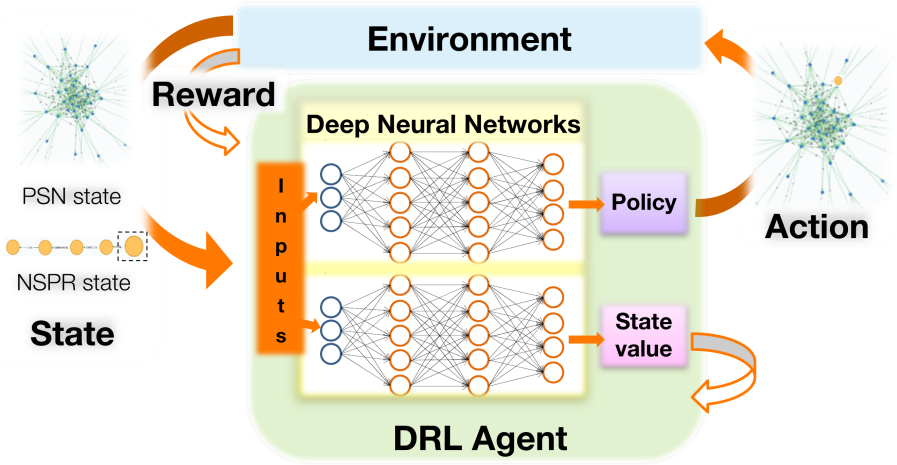


Figure 6.1: DRL framework for NSP Optimization

6.4.2 Policy Enforcement

Let $\mathcal{A}$ be the set of all possible actions that the DRL agent can take and Σ the set of all states that it can visit. We adopt a sequential placement strategy in which, at each time step, we choose a node $n \in N$ where to place a specific VNF $v \in \{1, ..., |V|\}$. The VNFs are placed in ascending order, which means that the placement starts with the VNF $v = 1$ and ends for the VNF $v = |V|$.

We break then the process of placing one NSPR graph $G_v = (V, E)$ in a sequence of $|V|$ actions, one for each $v \in V$, instead of considering the one shot placement of G_v. The latter strategy would require the definition of the action as a subgraph of the PSN graph $G_s = (N, L)$ what would imply $|\mathcal{A}| = |SG|$, where SG is the set of all sub graphs of G_s, that grows exponentially with size of G_s. Note that with the sequential placement strategy $\mathcal{A} = N$, thus $|\mathcal{A}| \ll |SG|$. At each time step t, the DRL agent focuses on the placement of exactly one VNF $v \in V$ of the NSPR.

Given a state σ_t, the DRL agent uses the policy to select an action $a_t \in \mathcal{A}$ corresponding to the PSN node for placing VNF v. The policy probabilities are calculated using the Softmax distribution defined by

$$\pi_\theta(a_t = a|\sigma_t) = \frac{e^{Z_\theta(\sigma_t, a),}}{\sum_{b \in N} e^{Z_\theta(\sigma_t, b)}}, \tag{6.4}$$

where the function $Z_\theta : \Sigma \times \mathcal{A} \to \mathbb{R}$ maps each state and action to a real value. In our formulation this function is calculated by a DNN described in Section 6.5.1. The notation π_θ is used to indicate that policy depends on Z_θ. The control parameter θ represents the weights in the DNN.

6.4.3 State Representation

The state contains a compact representation of the main elements of our model.

Physical Substrate Network state

The PSN State is the real-time representation of the PSN status, which is given by four characteristics. Three of these characteristics are related to resources (see Table 4.2 for the notation) and are defined by the following sets: $cap^{cpu} = \{cap_n^{cpu} : n \in N\}$, $cap^{ram} = \{cap_n^{ram} : n \in N\}$ and $cap^{bw} = \{cap_n^{bw} = \sum_{(n,b)\in L} cap_{(n,b)}^{bw} : n \in N\}$. We consider in addition one characteristic related to placement in order to record the actions taken by the DRL agent during the placement of the current NSPR described by the vector $\chi = \{\chi_n \in \{0, .., |V|\} : n \in N\}$, where χ_n corresponds to the number of VNFs of the current NSPR placed on node n.

Network Slice Placement Request state

The NSPR State represents a view of the current placement. It is composed by four characteristics. Three resource requirement characteristics (see Table 4.3 for the notation) associated with the current VNF v to be placed: req_v^{cpu}, req_v^{ram} and $req_v^{bw} = \sum_{(v,\bar{b})\in E} req_{(v,\bar{b})}^{bw}$. We also consider the number $m_v = |V| - v + 1$ used to track the number of VNFs still to be placed at each time step.

Network Load state

To deal with cycle-stationary network load, we introduce the Load State that represents the network load forecast used to learn the network load variations. It is defined by a set ρ_j of 100 features for each resource j calculated using the network load formula described in Definition 6.3.2 applied to : $\rho_j = \{\rho_j(t) : t_a \leqslant t \leqslant t_a + 100, t \in \mathbb{N}\}$, where t_a is the simulated time instant in which the current NSPR arrives.

6.4.4 Reward Function

The proposed Reward function contains one reward value for each optimization objective introduced in Section 6.2 as detailed in the following sections.

Acceptance Reward

Each Action may lead to a successful or unsuccessful placement depending on whether it respects the problem constraints for the candidate VNF v and its associated VLs or not. We then define the Acceptance Reward value due to action a_t as

$$\delta_{t+1}^{a} = \begin{cases} 100, & \text{if } a_t \text{ is successful,} \\ -100, & \text{otherwise.} \end{cases} \tag{6.5}$$

Resource Consumption Reward

As above, we define the Resource Consumption Reward value for the placement of VNF v via action a_t as

$$\delta_{t+1}^{c} = \begin{cases} \frac{req_{(v-1,v)}^{bw}}{req_{(v-1,v)}^{bw}|P|} = \frac{1}{|P|}, & \text{if } |P| > 0, \\ 1, & \text{otherwise.} \end{cases} \tag{6.6}$$

where P is the path used to place VL $(v-1, v)$. Note that a maximum $\delta_{t+1}^{c} = 1$ is given when $|P| = 0$, that is, when VNFs $v-1$ and v are placed on the same server.

Load Balancing Reward

Finally, we define the Load Balancing Reward value for the placement of VNF v via a_t

$$\delta_{t+1}^{b} = \frac{cap_{a_t}^{cpu}}{M_{a_t}^{cpu}} + \frac{cap_{a_t}^{ram}}{M_{a_t}^{ram}}. \tag{6.7}$$

Global Reward

On the basis of the three reward values introduced above, we define the global as

$$r_{t+1} = \begin{cases} 0, & \text{if } t < T \text{ and } a_t \text{ is successful} \\ \sum_{i=0}^{T} \delta^a_{i+1}\delta^b_{i+1}\delta^c_{i+1}, & \text{if } t = T \text{ and } a_t \text{ is successful} \\ \delta^a_{t+1}, & \text{otherwise} \end{cases} \tag{6.8}$$

where T is the number of iterations of a training episode and the quantities δ^a_{i+1}, δ^b_{i+1}, and δ^c_{i+1} are defined by Equations (6.5), (6.7) and (6.6), respectively. The notation r_{t+1} is used to emphasize that the reward obtained via action a_t is provided by the environment after the action is performed, that is, at time step $t+1$. In the present case, we consider $T \leqslant |V|$. Our reward function formulation is inspired by that proposed in [46] but different. In [46], partial rewards are admitted by considering that $r_{t+1} = \delta^a_{i+1}\delta^b_{i+1}\delta^c_{i+1}$. In practice this approach can lead to ineffective learning, since the agent can learn how to obtain good partial rewards without never reaching its true goal, that is, to place the entire NSPR.

To address this issue, we propose to accumulate the intermediary rewards and return them to the agent only if all the VNFs of the NSPR are placed –second case of Equation (6.8). A 0 reward is given to the agent on the intermediary time steps, where a VNF is successfully placed (first case of Equation (6.8)). If the agent takes an unsuccessful placement action, a negative reward is given (third case of Equation(6.8)).

6.5 Adaptation of Deep Reinforcement Learning and introduction of a Heuristic Function

In this section, we present how we use the DRL framework introduced in Section 6.4 to learn optimal NSP decisions. In Section 6.5.1, we present the proposed DRL algorithm. In Section 6.5.2 we discuss the introduction of a heuristic function to accelerate the learning. Finally, in Section 6.5.3, we provide some implementation remarks.

6.5.1 Proposed Deep Reinforcement Learning Algorithm

To learn the optimal policy, we use a single thread version of the Asynchronous Advantage Actor Critic (A3C) algorithm introduced in [104]. This algorithm has evidenced good results when applied to the VNE problem in [46]. A3C uses two DNNs that are trained in parallel: i) the Actor Network with the parameter θ, which is used to generate the policy π_θ at each time step, and ii) the Critic Network with the parameter θ_v which generates an estimate $\nu^{\pi_\theta}_{\theta_v}(\sigma_t)$ for the State-value function defined by

$$\nu_\pi(t|\sigma) = \mathbb{E}_\pi \left[\sum_{k=0}^{T-t-1} \gamma^k r_{t+k+1} | \sigma_t = \sigma \right],$$

equal to the expected return when starting on state σ and following policy π thereafter with the discount parameter γ.

Deep Neural Networks Structure

As depicted in Figure 6.2 both Actor and Critic Networks have almost identical structure. As in [46], we use the GCN formulation proposed by [102] to automatically extract advanced characteristics of the PSN. The characteristics produced by the GCN represent semantics of the PSN topology by encoding and accumulating characteristics of neighbour nodes in the PSN graph. The size of the neighbourhood is defined by the order-index parameter K. As in [46], we consider in the following $K = 3$ and perform automatic extraction of 60 characteristics per PSN node. Both the NSPR state and Network Load characteristics are separately transmitted to fully connected layers with 4 and 100 units, respectively.

The characteristics extracted by both layers and the GCN layer are combined into a single column vector of size $60|N| + 104$ and passed through a full connection layer with $|N|$ units. In the Critic Network, the outputs are forwarded to a single neuron, which is used to calculate the state-value function estimation $\nu^{\pi_\theta}_{\theta_v}(\sigma_t)$.

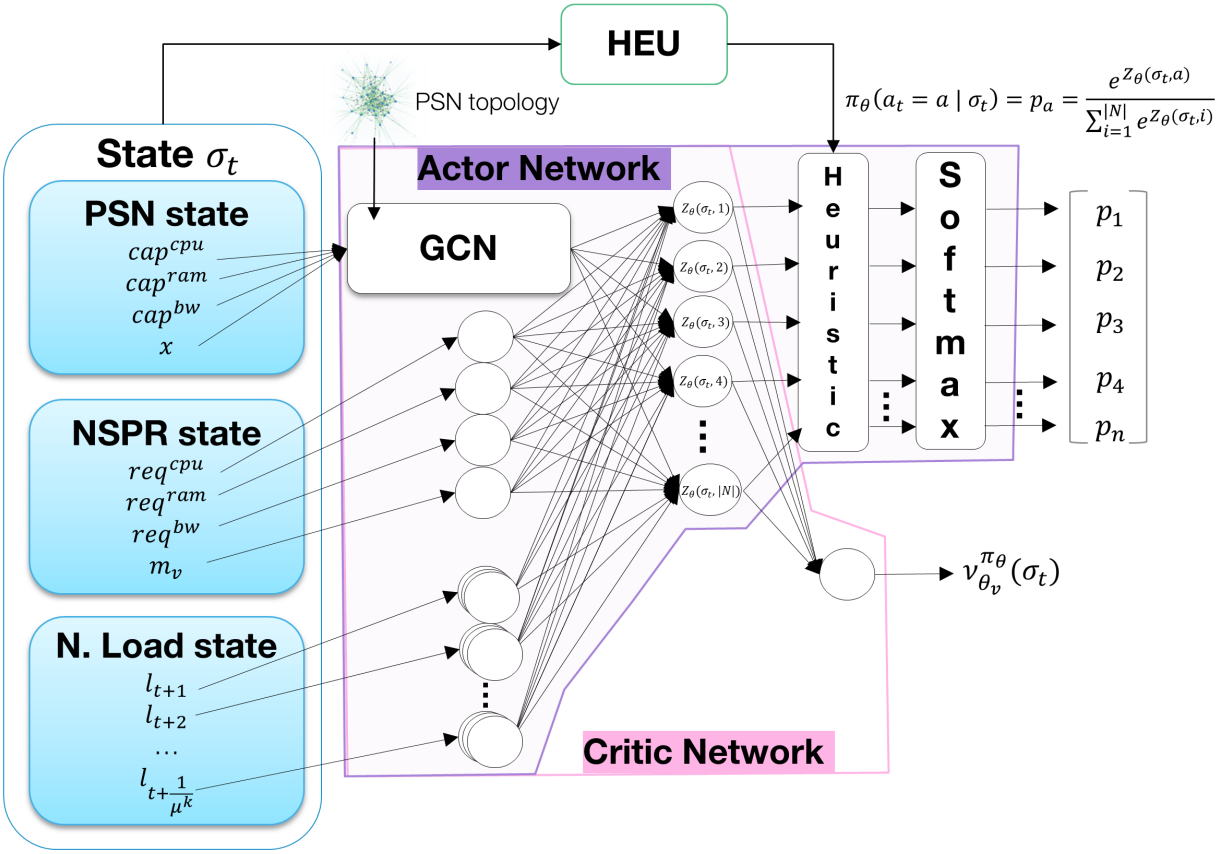


Figure 6.2: Reference architecture for the proposed learning algorithms.

In the Actor Network, the outputs represent the values of the function Z_θ introduced in Section 6.4.2. These values are injected into a Softmax layer that transforms them into a Softmax distribution that corresponds to the policy π_θ.

Deep Neural Networks Update

During the training phase, at each time step t, the A3C algorithm uses the Actor Network to calculate the policy $\pi_\theta(.|\sigma_t)$. An action a_t is sampled using the policy and performed on the environment. The Critic Network is used to calculate the state-value function approximation $\nu_{\theta_v}^{\pi_\theta}(\sigma_t)$. The agent receives then the reward r_{t+1} and next state σ_{t+1} from the environment and the placement process continues until a terminal state is reached, that is, until the Actor Network returns an unsuccessful action or until the current NSPR is completely placed. At the end of the training episode, the A3C algorithm updates parameters θ and θ_v adopting the rules given in Table 6.1, derived from the Policy Gradient Theorem [104].

$$J(\theta) = \sum_{t=t_0}^{T} \log\left(\pi_\theta(a_t|\sigma_t)\right) A^{\pi_\theta}(\sigma_t, a_t)$$
$$\delta(\theta) = \sum_{t=t_0}^{T} H(\pi_\theta(.|\sigma_t))$$
$$\theta \leftarrow \theta + \frac{\alpha}{T - t_0 + 1}\left(\nabla_\theta J(\theta) + \phi \nabla_\theta \delta(\theta)\right) \tag{6.9}$$

$$J(\theta_v) = \sum_{t=t_0}^{T} \left(r_{t+1} + \nu_{\theta_v}^{\pi_\theta}(\sigma_{t+1}) - \nu_{\theta_v}^{\pi_\theta}(\sigma_t)\right)^2$$
$$\theta_v \leftarrow \theta_v + \frac{\alpha'}{T - t_0 + 1} \nabla_{\theta_v} J(\theta_v) \tag{6.10}$$

Table 6.1: Updates of the control parameters θ and θ_v.

In Table 6.1, the term $A^{\pi_\theta}(\sigma_t, a_t)$ represents an estimate of the Advantage function, that indicates how better is the action a_t when compared against the "average action" from the corresponding policy under a certain state σ_t. By applying the Temporal Difference method [27], $A^{\pi_\theta}(\sigma_t, a_t) = r_{t+1} + \nu_{\theta_v}^{\pi_\theta}(\sigma_{t+1}) - \nu_{\theta_v}^{\pi_\theta}(\sigma_t)$.

The $J(\theta)$ function is the performance of the Actor Network. It is given by the Log-likelihood of the policy weighted by the Advantage Function. The $\delta(\theta)$ is the sum of the policy entropy $H(\pi_\theta(.|\sigma_t)$ used as a regularization term to discourage premature convergence. The function $J(\theta_v)$ is the performance of the Critic Network. It is given by the squared Temporal Difference error of $\nu_{\theta_v}^{\pi_\theta}(\sigma_{t+1})$. The gradients ∇_θ and ∇_{θ_v} are the vectors of partial derivatives w.r.t. θ and θ_v, respectively. The hyper-parameters α, α', and ϕ are the learning rates and heuristically fixed. t_0 is the first time step of the training episode and T is the last one. Note that $t_0 \leqslant T < t_0 + |V|$.

6.5.2 Introduction of a Heuristic

Motivation

For the proposed DRL agent to learn the optimal policy π_θ^* it needs to perform actions over various states and receive the respective rewards to improve the policy iteratively. However, this is not a straightforward process since the convergence depends on many hyper parameters. The agent may get stuck in sub-optimal policies or have trouble to learn good estimations for the state-value function.

Also, DRL approaches need to perform many actions and visit a huge number states to learn good policies. As a consequence, these approaches take much time to start providing good solutions. We propose then to reinforce and accelerate the DRL learning process using the heuristic described in Chapter 5.

Modified DRL algorithm

To guide the learning process, we use as the placement heuristic P2C 1 introduced in Chapter 5 and referred to here as HEU. This yields the HA-DRL algorithm. More precisely, from the reference framework shown in Figure 6.2, we proposed to include in the Actor Network the Heuristic layer that calculates a Heuristic Function $H : \Sigma \times \mathcal{A} \rightarrow \mathbb{R}$ based on external information provided by the heuristic method.

As described in [13], the Heuristic Function $H(\sigma_t, a_t)$ is a policy modifier that defines the importance of executing a certain action a_t in state σ_t. In what follows, we adapt the $H(\sigma_t, a_t)$ formulation proposed in [13] to our DRL algorithm and describe how we use $H(\sigma_t, a_t)$ as a modifier of the Actor Network output to give more importance to the action selected by the HEU method.

Heuristic Function Formulation

Let Z_θ be the function computed by the fully connected layer of the Actor Network that maps each state and action to a real value which is after converted by the Softmax layer into the selection probability of the respective action (see Section 6.4.2). Let $\bar{a}_t = \text{argmax}_{a \in \mathcal{A}} Z_\theta(\sigma_t, a)$ be the action with the highest Z_θ value for state σ_t. Let $a_t^* = HEU(\sigma_t)$ be the action derived by the HEU method at time step t and the preferred action to be chosen. $H(\sigma_t, a_t^*)$ is shaped to allow the value of $Z_\theta(\sigma_t, a_t^*)$ to become closer to the value of $Z_\theta(\sigma_t, \bar{a}_t)$. The aim is to turn a_t^* into one of the likeliest actions to be chosen by the policy.

The Heuristic Function is then formulated as

$$H(\sigma_t, a_t) = \begin{cases} Z_\theta(\sigma_t, \bar{a}_t) - Z_\theta(\sigma_t, a_t) + \eta, & \text{if } a_t = a_t^* \\ 0, & \text{otherwise} \end{cases} \tag{6.11}$$

where η parameter is a small real number. During the training process the Heuristic layer calculates $H(\sigma_t, .)$ and updates the $Z_\theta(\sigma_t, .)$ values by using the following equation:

$$Z_\theta(\sigma_t, .) = Z_\theta(\sigma_t, .) + \xi H(\sigma_t, .)^\beta \tag{6.12}$$

The Softmax layer then computes the policy using the modified Z_θ. Note the action returned by a_t^* will have a higher probability to be chosen. The ξ and β are parameters used to control how much HEU influence the policy.

6.5.3 Implementation remarks

Feature normalization

All resource-related characteristics are normalized to be in the $[0, 1]$ interval. This is done by dividing cap^j and req^j, $j \in$ {cpu, ram,bw} by $\max_{n \in N} M_n^j$.

Reward normalization

We have normalized positive global rewards to be in the $[0, 10]$ interval.

Deep Neural Networks implementation

With regard to the DNNs, we have implemented the Actor and Critic as two independent Neural Networks. Each neuron has a bias assigned.

Neurons activation functions

We have used the hyperbolic tangent (tanh) activation for non-output layers of the Actor Network and Rectified Linear Unit (ReLU) activation for all layers of the Critic Network.

Action sampling

During the training phase, we have considered the policy as a Categorical distribution and used it to sample the actions randomly. In the validation tests, we have always selected the action with higher probability, i.e., $a_t = \text{argmax}_{a \in N} \pi_\theta(a|\sigma_t)$.

Hyper-parameter setup

As in [46], we have taken $\phi = 0.5$. We performed hyper-parameter search to define α and α' values as described in Section 6.6.2.

Algorithms considered

We consider four learning algorithms based on the reference framework presented in Figure 6.2.

- **DRL:** This is the simpler algorithm, we call it the non-controlled DRL algorithm. The state representation does not include the network load state and the Actor Network does not contain the Heuristic layer.
- **eDRL:** This algorithm is an enhanced version of DRL in which the state representation includes the network load state and the Actor Network does not contain the Heuristic layer;
- **HA-DRL:** This algorithm embeds the heuristic but the state representation does not include the network load state while the Actor Network contain the Heuristic layer;
- **HA-eDRL:** The state representation includes the network load state and the Actor Network contains the Heuristic layer.

6.6 Experiments, evaluation results, and discussions

In this section, we present the implementation and experiments we conducted to evaluate the proposed DRL-based algorithms. First, we present in Section 6.6.1 the implementation details and experimentation settings. Then we present the experiments and evaluation results. The experiments were performed in three steps and at each step we considered a different network load scenario and looked to evaluate different performance metrics. In the first step (Section 6.6.2), we consider stationary network load scenario (see Definition 4.3.3). In the second step (Section 6.6.3), we consider a cycle-stationary network load scenario (see Definition 6.3.2). Finally, in the third step (Section 6.6.4), we considered a non-stationary network load scenario (see Definition 6.3.3).

6.6.1 Implementation details and experimentation settings

Proposal architecture description

The architecture of the proposed NSP solution is given in the Figure 6.3. The NSPR generator is used to generate NSPR arrivals according to a specific network load regime. The PSN database stores the data about the available resources of the PSN. NSPR requirements and PSN available resources data are used as input by the Placement module. This latter implements the DRL-based algorithms and also the P2C heuristic algorithm referred to here as HEU used by HA-DRL and HA-eDRL algorithms to accelerate convergence.

Both algorithms calculate: i) a VNF placement decision, that is, where each VNF of the NSPR is to be placed and ii) a VNF chaining decision, that is, which paths in the network to use to interconnect the different VNFs. The Placement module can be configured to use one of the Placement algorithms or both if comparison of Placement solutions is necessary.

Once the calculation of the Placement decision is done for one NSPR, an update of the available resources on the PSN is made and some key performance metrics are registered in the form of data series; the key metrics are: the acceptance ratio of network slices and the resource usage. These time series are used by the Data visualization component to build two dashboards: an acceptance ratio dashboard and a network load dashboard.

Both dashboards are used to show the performance of the algorithms according to variations on the network load in real time. Finally, the graph visualization component is used to allow the visualisations of the PSN and NSPR graphs.

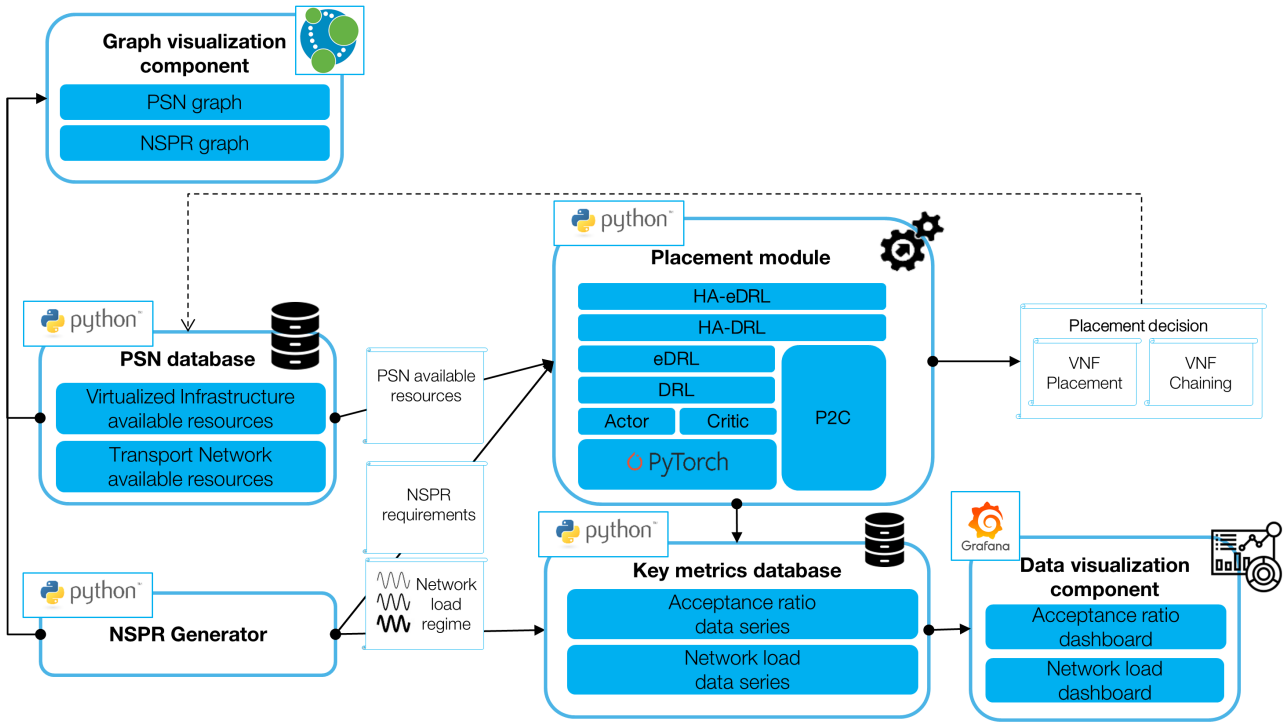


Figure 6.3: DRL-based NSP solution architecture

Proposal implementation description and tools

We have implemented the proposed NSP solution and we describe below the different tools used to implement its different components:

- **Python:** We use Python for implementing different elements of the proposed solution:

  1. the PSN database, that is actually represented by a series of data frames loaded from csv files,
  2. the NSPR generator that is actually a function that receives some parameters representing the CPU, RAM and bandwidth requirements of the NSPRs to be generated and also the parameters for the appropriated network load model,
  3. the placement algorithms.

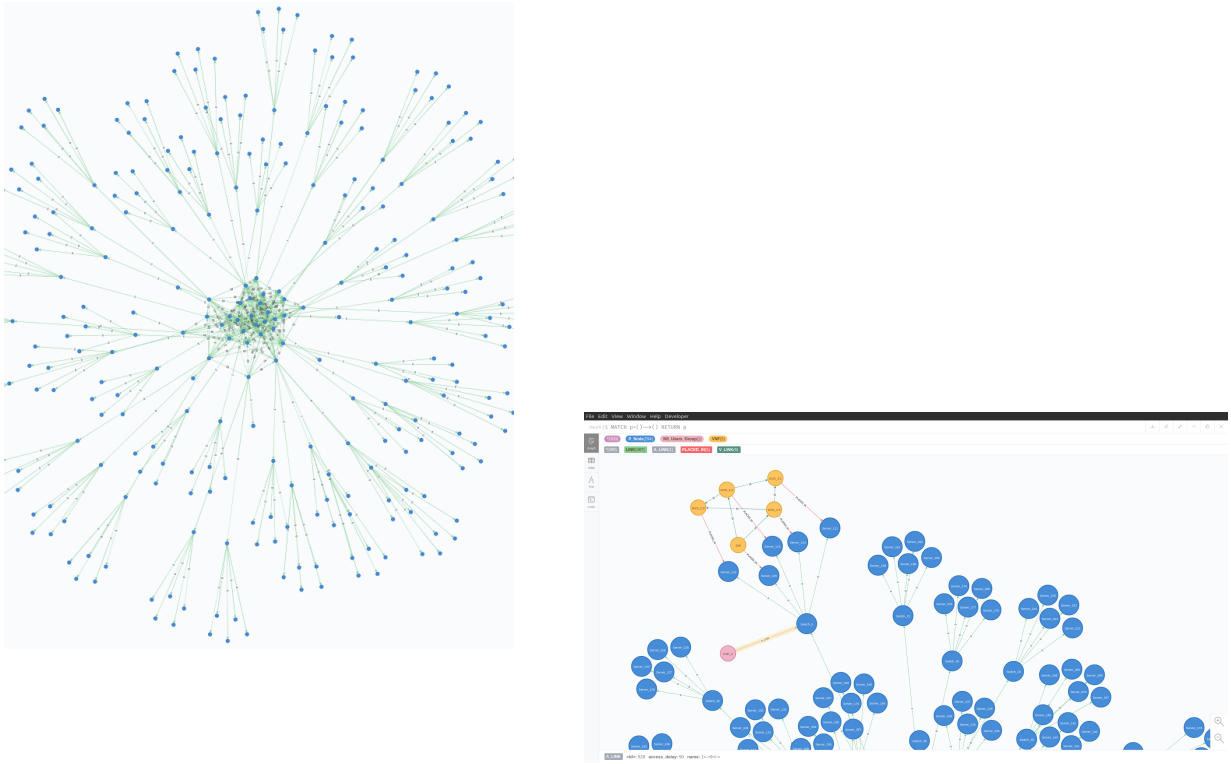
  The DRL and eDRL algorithms are built upon an implementation of the Actor and Critic DNNs using the PyTorch framework. The HA-DRL and HA-eDRL algorithms are implemented on top of these DNNs and also of different classes and functions used to implement the P2C heuristic designed and implemented from scratch using only default Python packages.

- **Neo4j:** A Neo4j graph database represents and displays the PSN graph and the NSPR graph together with its requirements.

- **MySQL:** We use the MySQL database manager system to implement the Key metrics database with one table for the Acceptance ratio data series and another one for the Network load data series.

- **Grafana:** We use the Grafana tool to implement the Data visualization component in which we represent two dashboards using the MySQL database of Key metrics as data-source.

Experiments were run in a 2x6 cores @2.95Ghz 96GB machine. PSN and NSPR are displayed by using Neo4j in Figure 6.4a and Figure 6.4b, respectively.



(a) PSN view using the Graph-based Visualization Component

(b) View of a Placement Decision using the Graph visualization component

Figure 6.4: Example of view using Graph-based Visualization component

Physical Substrate Network settings

We consider the same PSN setting described in Section 5.3.1. In this network, three types of DCs are introduced as in Section 4.1. Each CDC is connected to three EDCs which are 100 km apart. CDCs are interconnected and connected to a CCP that is 300 km away.

We consider 15 EDCs each one with 4 servers, 5 CDCs each with 10 servers and 1 CCP with 16 servers. The CPU and RAM capacities of each server are 50 and 300 units, respectively. A bandwidth capacity of 100 Gbps is given to intra-DC links inside CDCs and CCP, 10Gbps being the bandwidth for intra-DC links inside EDCs. Transport links connected to EDCs have 10Gpbs of bandwidth capacity. Transport links between CDCs have 100Gpbs of bandwidth capacity as well for the ones between CDCs and the CCP.

Network Slice Placement Requests settings

We consider NSPRs to have the Enhanced Mobile Broadband (eMBB) setting described in Section 5.3.1. Each NSPR is composed of 5 VNFs. Each VNF requires 25 units of CPU and 150 units of RAM. Each VL requires 2 Gbps of bandwidth.

6.6.2 Stationary network load scenario

In this section, we evaluate the DRL and HA-DRL algorithms introduced in Section 6.5.3 under a stationary network load scenario.

Training process and hyper-parameters

We consider a training process with maximum duration of 24 hours for the DRL and HA-DRL agents with learning rates for the Actor and Critic networks set to $\alpha = 10^{-4}$ and $\alpha' = 2.5.10^{-3}$ (see Section 6.6.2 about hyper-parameters tuning). We program four versions of HA-DRL agent (HA-DRL, $\beta = 0.1$; HA-DRL, $\beta = 0.5$; HA-DRL, $\beta = 1.0$; HA-DRL, $\beta = 2.0$), each with a different value for the β parameter of the heuristic function formulation (see Section 6.5.2). We set the parameters $\xi = 1$ and $\eta = 0$.

Heuristic baseline

In this section we consider an implementation in Julia of the P2C 1 algorithm introduced in Chapter 5.2, referred to here as HEU, as a benchmark in the performance evaluations. Since the performance of the HEU algorithm does not depend on learning, we consider the placement of 100,000 NSPRs with the HEU algorithm and use its performance in the steady state as a benchmark.

Network load calculation

We use the stationary network load model introduced in Definition 4.3.3 to compute the NSPR arrival rates (λ^k) under the three network load conditions considered in the evaluation: underload ($\rho = 0.5$), normal load ($\rho = 0.8$ and $\rho = 0.9$), and critical load ($\rho = 1.0$). Network loads are calculated using CPU resources. In general, we set $1/\mu_k = 100$ time units for all $k \in \mathcal{K}$, where $\mathcal{K}$ is the set of NSPR classes.

Evaluation metrics

To characterize the performance of the placement algorithms, we consider 3 performance metrics:

1. **Average execution time:** the average execution time in seconds required to place 1 NSPR. This metric is calculated based on 100 NSPR placements;
2. **Acceptance Ratio per training phase**: the Acceptance Ratio obtained in each training phase, i.e., each part of the training process, corresponding to 1000 NSPR arrivals. It is calculated as follows: $\frac{\#\text{accepted NSPRs}}{1000}$. This metric is used to evaluate the convergence of the algorithms as it allows us to observe of the evolution of the agent's performance in time;
3. **Acceptance Ratio after training:** the Acceptance Ratio of the different tested algorithms after training computed after each arrival as follows: $\frac{\#\text{accepted NSPRs}}{\#\text{arrived NSPRs}}$. This metric is used in our validation test to compare the performance of the agents after training.

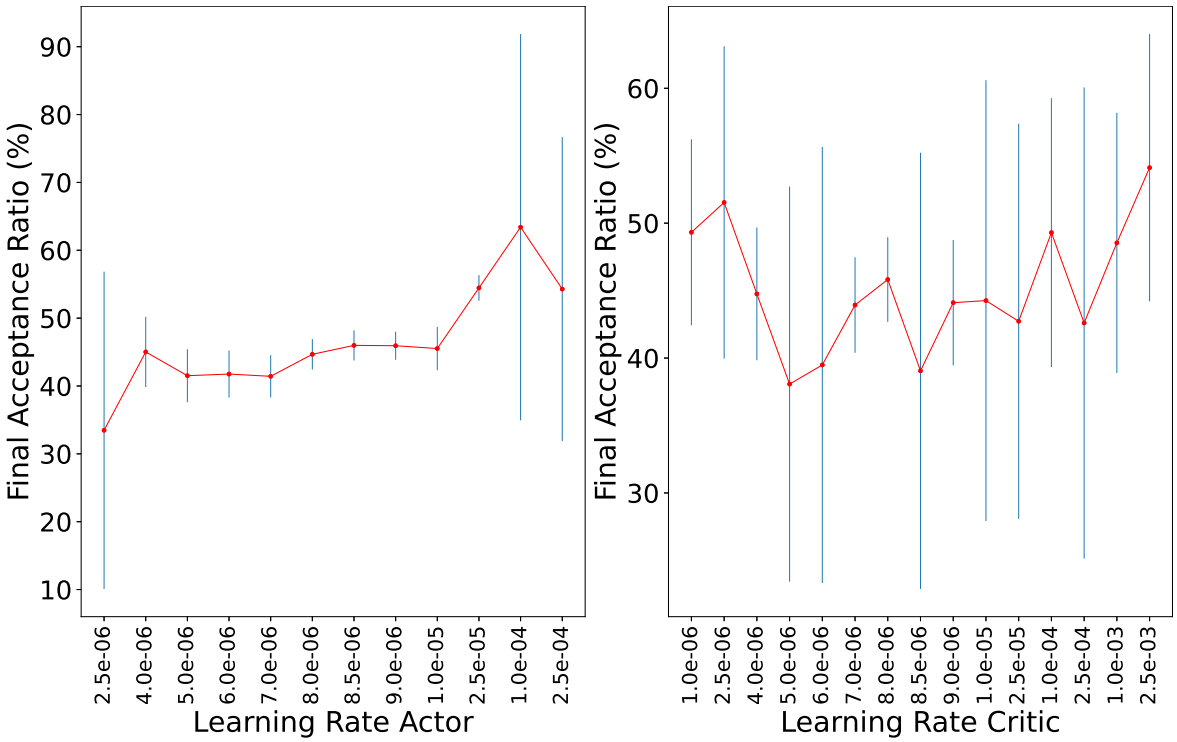
Hyper-parameter search results

Figures 6.5 and 6.6 present the hyper-parameter search results. To define the appropriate learning rates α and α' for training the Actor and Critic networks, respectively, we perform a classical hyper-parameter search procedure divided in 3 steps: i) we conduct several training experiments with different combinations of α and α' values, ii) we observe the final acceptance ratios, i.e., the acceptance ratios obtained in the last training phase of each experiment, and iii) we aggregate the final acceptance ratios by learning rate values.

The DNNs are trained for 4 hours and we consider α and α' to be in the following intervals: $\alpha \in [0.00001, 0.00025]$ and $\alpha' \in [0.000001, 0.0025]$, where upper bounds of the intervals are the learning rates used by [46]. Figures 6.5 and 6.6 present the average values and standard deviations of the final acceptance ratios aggregated by learning rate for the Actor and Critic networks considering two values for the network load parameter ρ: $\rho = 0.5$ (Figures 6.5a and 6.5b) and $\rho = 1.0$ (Figures 6.6a and 6.6b).

As shown in Figures 6.5a and 6.6a, under both network load conditions, the best average of final acceptance ratios was achieved when considering a learning rate $\alpha = 10^{-4}$ for the Actor Network. This fact can be observed even more clearly when $\rho = 1.0$ since we observe a peak in the curve when $\alpha = 10^{-4}$ in addition to a smaller standard deviation as shown by Figure 6.6a.

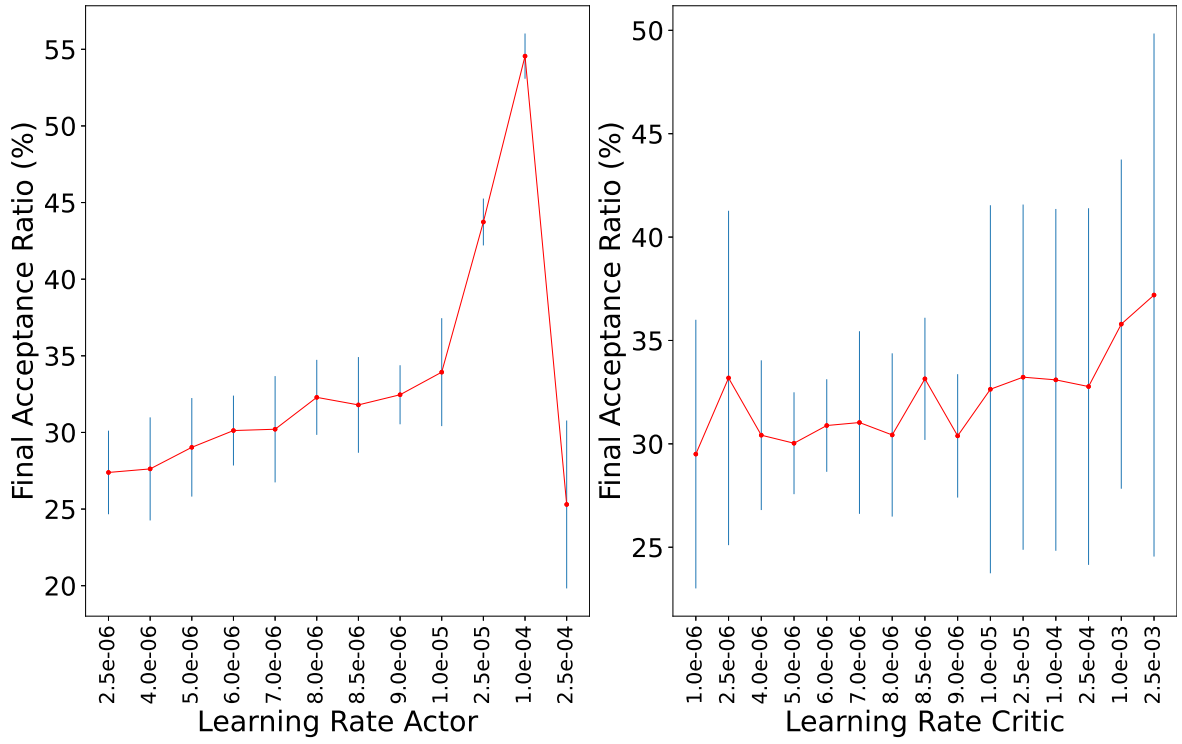
Figures 6.5b and 6.6b show that the best average of final acceptance rate was obtained when considering a learning rate $\alpha' = 2.5 \times 10^{-3}$ for the Critic Network. We also observe a higher standard deviation when aggregating the final acceptance ratios on the basis α' than on the basis of α. The parameter α thus seems to have a stronger impact on the final acceptance ratios obtained than obtained for α'.



(a) Actor Network, $\rho = 0.50$.

(b) Critic Network, $\rho = 0.50$.

Figure 6.5: Hyper-parameter search results: $\rho = 0.50$.



(a) Actor Network, $\rho = 1.0$.

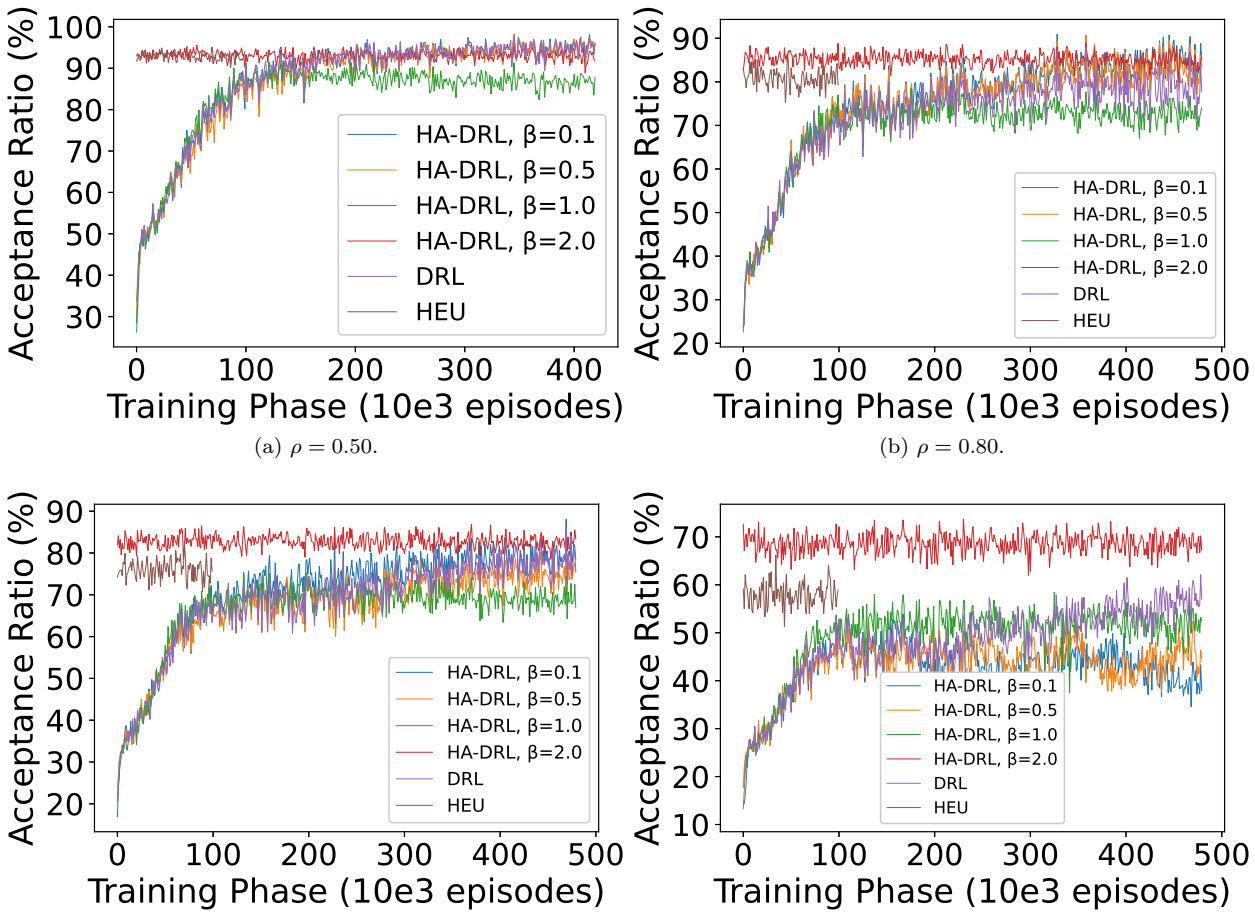
(b) Critic Network, $\rho = 1.0$.

Figure 6.6: Hyper-parameter search results: $\rho = 1.0$.

Acceptance ratio evaluation

Figure 6.7 and Tables 6.2-6.5 present the Acceptance Ratio per training phase obtained with the HA-DRL, DRL and HEU algorithms under different network loads.

HA-DRL with $\beta = 2.0$ exhibits the most robust performance with convergence after a few training phases for all values of ρ. This happens because when setting $\beta = 2.0$ the Heuristic Function computed on the basis of the HEU algorithm has strong influence on the actions chosen by the agent. Since the HEU algorithm often indicates a good action, this strong influence of the heuristic function helps the algorithm to become stable more quickly.



(a) $\rho = 0.50$.

(b) $\rho = 0.80$.

(c) $\rho = 0.90$.

(d) $\rho = 1.0$.

Figure 6.7: Acceptance ratio evaluation results.

Figure 6.7a and Table 6.2 reveal that DRL and HA-DRL algorithms with $\beta \in \{0.1, 0.5, 1.0\}$ converge within an interval of 200 to 300 training phases when $\rho = 0.5$ and that all algorithms except HA-DRL with $\beta = 1.0$ have an Acceptance Ratio higher than 94% in the last training phase. Figure 6.7b and Table 6.3 show that the performance of DRL and HA-DRL algorithms with $\beta \in \{0.1, 0.5, 1.0\}$ stabilizes only after more than 400 training phases when $\rho = 0.8$. They also show that when $\rho = 0.8$, only HA-DRL with $\beta \in \{0.1, 0.5, 2.0\}$ have an Acceptance Ratio higher than 83% in the last training phase.

Algorithms HA-DRL with $\beta = 0.1$ and HA-DRL with $\beta = 2.0$ have similar performance and they are about 2% better than HA-DRL for $\beta = 0.5$, 6% better than HEU, 8% better than DRL, and 11% better than HA-DRL with $\beta = 1.0$. As shown in Figure 6.7c and Table 6.4 when $\rho = 0.9$, only HA-DRL with $\beta = 2.0$ has an Acceptance Ratio higher than 83% in the last training phase.

HA-DRL with $\beta = 2.0$ is about 5% better HA-DRL with $\beta = 0.1$, 8% better than DRL, HEU and HA-DRL with $\beta = 0.5$, and 16% better than HA-DRL with $\beta = 1.0$.

Algorithm	Acceptance Ratio at different Training Phases (%)				
	25	100	200	300	400
HA-DRL,β=0.1	57.90	80.20	92.70	93.00	96.20
HA-DRL,β=0.5	58.50	83.00	90.20	94.80	96.30
HA-DRL,β=1.0	58.80	86.20	86.80	85.50	85.80
HA-DRL,β=2.0	93.50	90.30	93.10	92.20	94.80
DRL	57.10	83.60	91.20	94.80	95.70
HEU	93.50	94.00	94.00*	94.00*	94.00*

Table 6.2: Acceptance Ratio at different Training Phases, $\rho = 0.5$.

Algorithm	Acceptance Ratios at different Training Phases (%)					
	25	100	200	300	400	480
HA-DRL,β=0.1	44.10	74.60	77.80	82.80	87.50	85.30
HA-DRL,β=0.5	46.30	75.00	77.40	80.90	86.90	83.60
HA-DRL,β=1.0	46.00	77.00	74.90	72.40	74.80	74.10
HA-DRL,β=2.0	87.80	88.80	85.60	86.50	84.90	85.40
DRL	46.10	71.89	75.70	78.40	83.70	77.80
HEU	79.20	79.27	79.27*	79.27*	79.27*	79.27*

Table 6.3: Acceptance Ratio at different Training Phases, $\rho = 0.8$.

Algorithm	Acceptance Ratios at different Training Phases (%)					
	25	100	200	300	400	480
HA-DRL,β=0.1	42.40	66.20	75.00	73.00	81.70	78.50
HA-DRL,β=0.5	40.30	63.40	70.90	68.20	78.20	75.30
HA-DRL,β=1.0	42.69	66.00	70.90	68.40	71.10	66.90
HA-DRL,β=2.0	82.89	81.10	84.10	81.00	82.80	83.36
DRL	41.60	63.40	72.10	71.10	80.60	75.80
HEU	73.90	75.68	75.68*	75.68*	75.68*	75.68*

Table 6.4: Acceptance Ratio at different Training Phases, $\rho = 0.9$.

Algorithm	Acceptance Ratios at different Training Phases (%)					
	25	100	200	300	400	480
HA-DRL,β=0.1	30.10	49.70	45.30	45.0	41.00	37.90
HA-DRL,β=0.5	30.80	45.90	50.80	44.5	38.90	44.60
HA-DRL,β=1.0	27.40	52.00	55.50	55.60	49.00	52.50
HA-DRL,β=2.0	67.60	67.60	70.80	69.60	66.10	67.2
DRL	29.30	49.10	46.60	50.10	53.40	56.99
HEU	60.70	58.86	58.86*	58.86*	58.86*	58.86*

Table 6.5: Acceptance Ratio at different Training Phases, $\rho = 1.0$.

Figure 6.7c and Table 6.4 also reveal that the performance of algorithms DRL and HA-DRL with $\beta \in \{0.1, 0.5, 1.0\}$ only stabilizes after more than 400 training phases when $\rho = 0.9$. Figure 6.7d and Table 6.5 show that when $\rho = 1.0$, HA-DRL with $\beta = 2.0$ performs significantly better than the other algorithms at the end of the training. HA-DRL with $\beta = 2.0$ accepts 67.2 % of the NSPR arrivals in the last training phase, 8.34% more than HEU, 10.21% more than DRL, 14.7% more than HA-DRL with $\beta = 1.0$, 22.6% more than HA-DRL with $\beta = 0.5$, 29.3% more than HA-DRL with $\beta = 0.1$.

Figure 6.7d and Table 6.5 also show that performance of algorithms DRL and HA-DRL with $\beta \in \{0.1, 0.5, 1.0\}$ is still not stable at the end of the training process when $\rho = 1.0$.

Execution time evaluation

Figures 6.8a and 6.8b present the average execution time of the HEU, DRL and HA-DRL algorithms as a function of the number of VNFs in the NSPR and the number of servers in the PSN, respectively (see Section 6.6.2 for details on the metric calculation). In both evaluations, we start by the PSN and NSPR settings described in Section 6.6.1, and generate new settings by increasing either the number of VNFs per NSPR or the number of servers in the PSN.

The evaluation results confirmed our expectations by showing that the average execution times increase faster for heuristics than for a DRL approach. However, both HEU and DRL strategies have low execution times (less than 0.6s in the largest scenarios). The number of VNFs per NSPR has more impact on the average execution times of HEU and DRL algorithms than the number of servers on the PSN. The average execution time of HEU algorithm is more impacted than DRL by the number of servers in the PSN. The HA-DRL algorithm depends on a sequential execution of DRL and HEU. Therefore, the average execution time of HA-DRL is approximately to the sum of the execution times of HEU and DRL. Since DRL and HEU have small execution times, the average execution times of HA-DRL are also small (less than 1.0s for the largest NSPR setting and about 0.6s for the largest PSN setting).

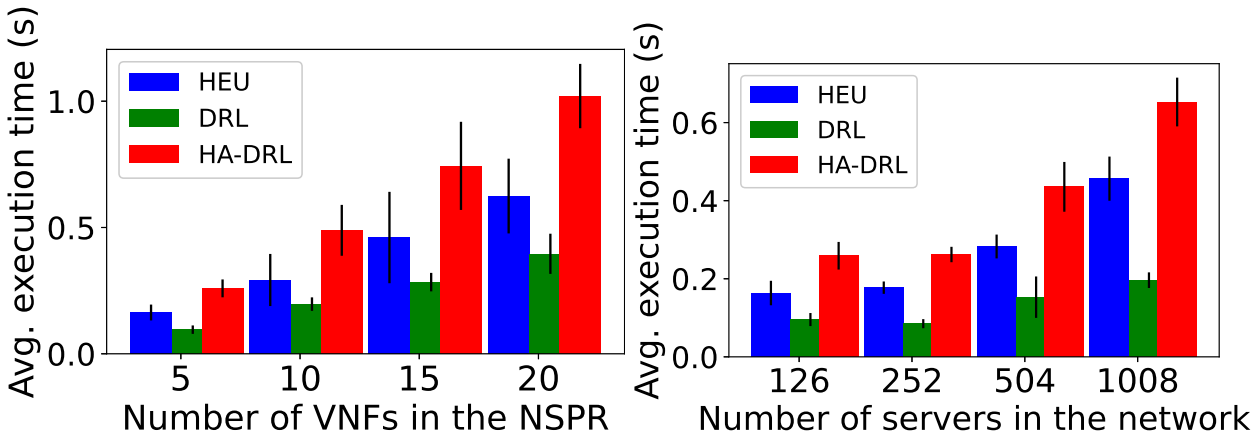


Figure 6.8: Average execution time evaluation.

Validation test

To validate the effectiveness of the different trained DRL agents, we perform a validation test. We consider the same PSN and NSPR settings described in Section 6.6.1 and the arrival of 10,000 NSPRs to be placed and generating a network load $\rho = 0.8$. We run a simulation with each one of the trained DRL agents as well with the HEU algorithm and compare the obtained Acceptance Ratios at the end of the simulations (see description of the Acceptance Ratio after training metric on Section 6.6.2). Figure 6.9 and Table 6.7 show that the HA-DRL agent with $\beta = 0.1$ is the one that better scales since it has the best Acceptance Ratio after training.

HA-DRL agent with $\beta = 2.0$ has the best Acceptance Ratio during training as described in Section 6.6.2 but is the one that scales the worst since it has poorest Acceptance Ratio after training performance. This is because with the use of a high β value, HA-DRL suffers from a strong dependence on the HEU algorithm. This prevents HA-DRL with $\beta = 2.0$ from being used with the HEU algorithm deactivated. Note, however, that HA-DRL remains the best solution even when HEU is disabled after 24 hours of training since HA-DRL with $\beta = 0.1$ has an Acceptance Ratio 7.34% higher than the non-controlled DRL and 4.42% higher than the HEU algorithm.

NSPR Classes (k)	# of VNFs per NSPR ($\|V^k\|$)	CPU requested per VNF ($req^{cpu,k}$)	NSPR lifetime ($\frac{1}{\mu^k}$)	NSPR arrival rate ($\lambda^k(t)$)	Total CPU capacity (C^{cpu})	Network Load ($\rho^k(t)$)
Volatile	5	25	20	$1.5\sin^2(\frac{\pi t}{96})$	6300	$1.5\sin^2(\frac{\pi t}{96})\frac{2500}{6300}$
Long term	10	25	500	0.02	6300	$0.02\frac{125000}{6300}$

Table 6.6: Network Load Calculation for both NSPR Classes

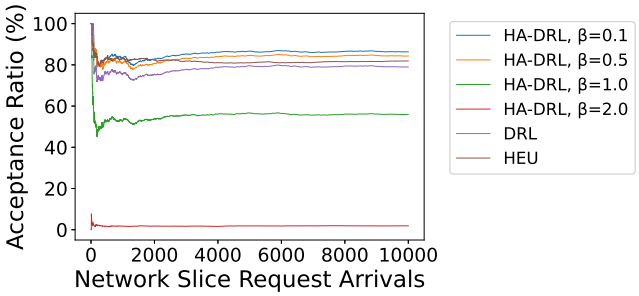


Figure 6.9: Acceptance Ratio at validation test, $\rho = 0.8$

HA-DRL,$\beta = 0.1$	86.31%
HA-DRL,$\beta = 0.5$	84.31%
HA-DRL,$\beta = 1.0$	55.95%
HA-DRL,$\beta = 2.0$	1.89%
DRL	78.97%
HEU	81.89%

Table 6.7: Values of Acceptance Ratio after Training

Optimality test

To evaluate the optimality of the proposed DRL-based algorithms, we consider a small scale network optimization scenario (i.e., with a 10 server PSN topology, illustrated in Figure 6.10). The CPU and RAM capacities of each server are 50 and 300 units respectively and the all the links have a bandwidth capacity of 100 Gbps. The NSPRs have the same settings described in Section 6.6.1.

We consider a Julia implementation of the offline ILP described in Chapter 4. This "oracle" ILP have complete knowledge about the future, i.e., it knows at time $t = 0$ all the arrival and exit dates for all NSPRs arriving to be placed during the whole time horizon $t = 0..., T$. We consider 10,000 NSPRs arriving to be placed during this time horizon and use the ILP to calculate the optimal solution for the problem in "one-shot" way.

We use the default branch-and-bound algorithm from `ILOG` CPLEX 12.9 to solve this ILP setting up 12 execution threads. It is worth noting that the average convergence time for the ILP in this illustrative scenario is 3.57 hours. We train our DRL-based algorithms during 2 hours in a data set having the same NSPR and PSN settings and calculate the gaps to optimal acceptance ratios using the following formula: GAP = algorithm acceptance ratio – optimal acceptance ratio.

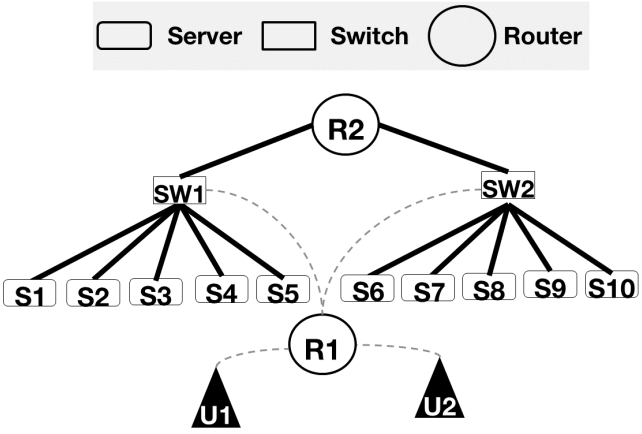


Figure 6.10: PSN topology used for optimality evaluation

Table 6.8 present the online learning results, i.e., the gap to the optimal acceptance ratio obtained at the end of the training phase. We can see that in only 2 hours of training, all the algorithms achieve near optimal performance.

In line with the results shown in the previous sections of this chapter, HA-DRL, $\beta = 2.0$ has the best performance with a gap always inferior to 10% and close to only 2% when $\rho = 0.5$. Table 6.9 presents the offline learning results, i.e., the gap to the optimal acceptance ratio obtained when executing the models already trained. In this scenario, all the algorithms except, HA-DRL, $\beta = 2.0$ have near optimal performance, with gaps to the optimal inferior to 10% after only 2 hours of training.

These results are in line with what is shown previously, since as discussed in Section 6.6.2, the poor performance of HA-DRL, $\beta = 2.0$ in offline mode is because with the use of a high β value, HA-DRL suffers from a strong dependence on the HEU algorithm. This prevents HA-DRL with $\beta = 2.0$ from being used with the HEU algorithm deactivated.

Tables 6.8 and 6.9 also show that, in this illustrative scenario, HA-DRL and DRL have close performance. However, it should be noted that the performance evaluations discussed previous sections show that, in more realistic scenarios (i.e., with a PSN with a higher number of nodes and links) the HA-DRL algorithm performs better than DRL and also than the HEU benchmark.

	$\rho = 0.5$	$\rho = 0.8$	$\rho = 0.9$	$\rho = 1.0$
DRL	4.29	8.50	9.39	11.34
HA-DRL, $\beta = 0.1$	4.64	8.84	9.53	12.15
HA-DRL, $\beta = 0.5$	4.27	8.59	9.36	11.37
HA-DRL, $\beta = 1.0$	4.82	8.76	10.18	12.10
HA-DRL, $\beta = 2.0$	2.32	6.75	7.68	9.87

Table 6.8: Gap to optimal acceptance ratio for different network load levels: online training (%)

	$\rho = 0.5$	$\rho = 0.8$	$\rho = 0.9$	$\rho = 1.0$
DRL	2.23	6.71	7.64	9.77
HA-DRL, $\beta = 0.1$	2.24	6.70	7.59	9.76
HA-DRL, $\beta = 0.5$	2.23	6.70	7.59	9.78
HA-DRL, $\beta = 1.0$	2.40	6.76	7.81	9.83
HA-DRL, $\beta = 2.0$	92.56	83.93	80.73	78.48

Table 6.9: Gap to optimal acceptance ratio for different network load levels: offline training (%)

6.6.3 Cycle-stationary network load scenario

In this section we evaluate the DRL, eDRL, HA-DRL and HA-eDRL algorithms introduced in Section 6.5.3 under a cycle-stationary network load scenario.

Training process and hyper-parameters

We consider a training process with maximum duration of 55 hours for the considered algorithms. We perform seven independent runs of each algorithm to assess their average and maximal performance in terms of metrics introduced below (see Section 6.6.3). After performing Hyper-parameter search, we set the learning rates for the Actor and Critic networks of DRL and HA-DRL algorithms to $\alpha = 5 \times 10^{-5}$ and $\alpha' = 1.25 \times 10^{-3}$, respectively.

For eDRL and HA-eDRL, we set the learning rates for the Actor and Critic networks to $\alpha = 5.7 \times 10^{-5}$ and $\alpha' = 1.4 \times 10^{-3}$, respectively. We implement four versions of HA-DRL and HA-eDRL agents, each with a different value for the β parameter of the heuristic function formulation (see Section 6.5.2). We set in addition the parameters $\xi = 1$ and $\eta = 0$.

Network load calculation

We consider two NSPR classes: i) a volatile class (referred to as Volatile), with a dynamic arrival process (see Definition 6.3.1) and ii) a static class (referred to as Long term), following a static arrival process (see Definition 4.3.2). Table 6.6 presents the network load models proposed for both classes. We consider that each simulation time unit corresponds to 15 minutes in reality. We set the period for the network load function to 96 simulation time units, i.e., one day. The network global network load $\rho(t)$ varies then between 0.3 and 1.0.

Evaluation metrics

To characterize the performance of the placement algorithms, we consider two performance metrics:

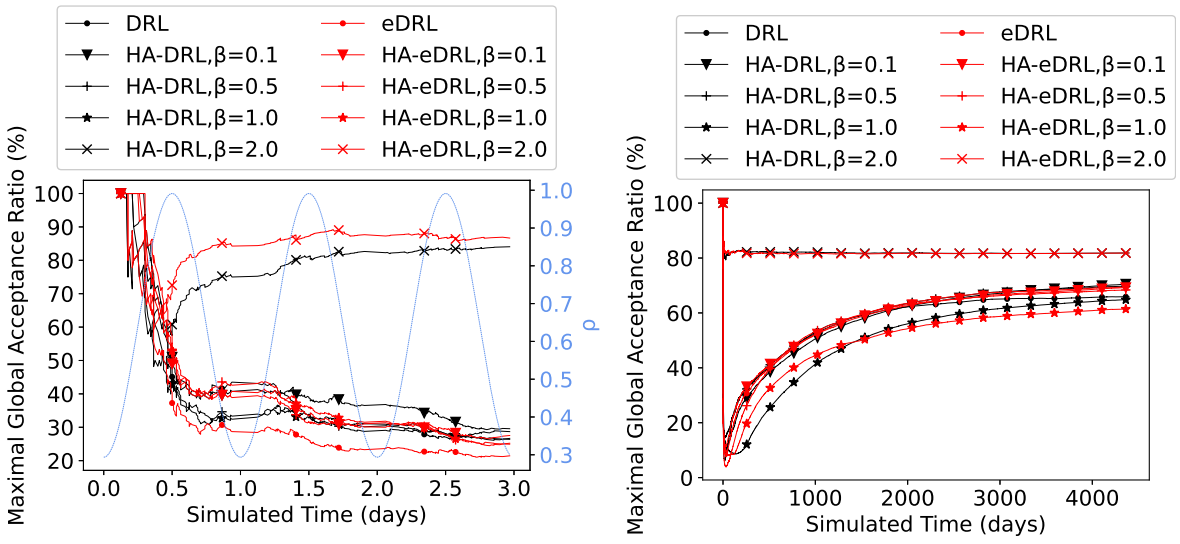
1. **Global Acceptance Ratio (GAR):** The Acceptance Ratio of the different tested algorithms during training computed after each arrival as follows: $\frac{\#\text{accepted NSPRs}}{\#\text{arrived NSPRs}}$. This metric is used to evaluate the accumulated acceptance ratio of the different algorithms as the learning process progresses.

2. **Acceptance Ratio per training phase (TAR):** The Acceptance Ratio obtained in each training phase, i.e., each part of the training process, corresponding to 10^4 NSPR arrivals or 10^4 episodes. It is calculated as follows: $\frac{\#\text{accepted NSPRs}}{10^4}$. This metric allows us to better observe the evolution of algorithm performance over time since it measures algorithm performance in independent parts (phases) of the training process without accumulating the performance of previous training phases.

Global acceptance ratio evaluation

Figures 6.11 and 6.12 show the progression of the GAR over time for the considered algorithms. Figure 6.11a and Figure 6.12a show that after 3 simulated days, the average and maximal GARs of HA-DRL and HA-eDRL with $\beta = 2.0$ are convergent and higher than 80% while for the other algorithms, the GARs remain between 20% and 40% and are not stable.

Figure 6.11b and Figure 6.12b exhibit the progression of the GARs over a longer simulated time horizon of 4500 simulated days. We observe that while GARs of HA-DRL and HA-eDRL with $\beta = 2.0$ remain stable after the convergence reached in the first 3 simulated days, for the other algorithms, these performance metrics start to stabilize only after 2000 simulated days.

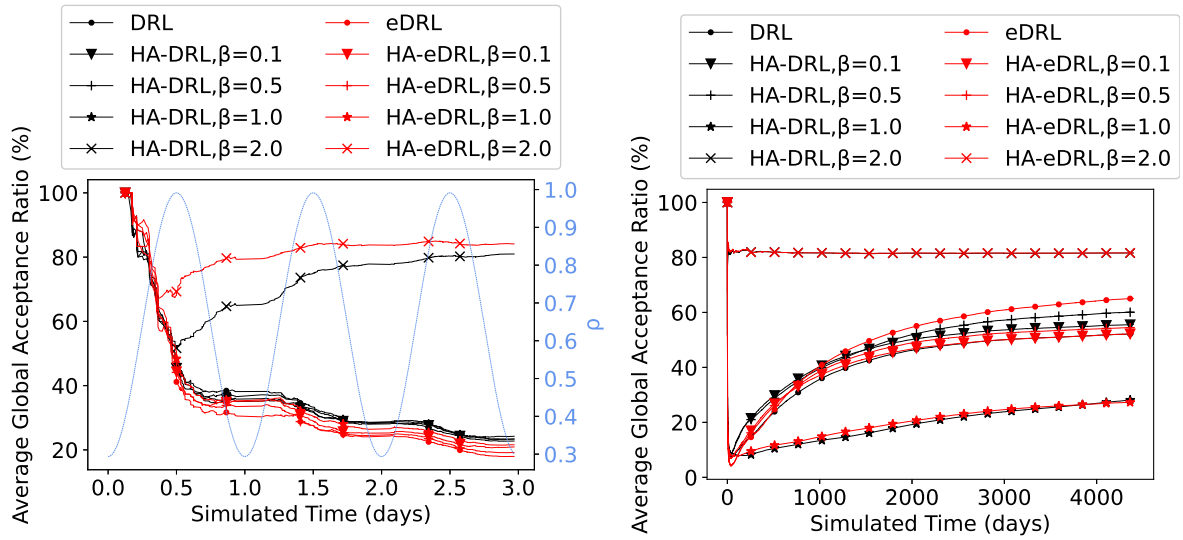


(a) Maximal performance in 3 simulated days.

(b) Maximal performance in 4500 simulated days.

Figure 6.11: Maximal Global Acceptance ratio results.

Table 6.10 shows that both the maximal and average final GARs, i.e., maximal and average GARs achieved at the end of training, are close to 82% for both HA-DRL and HA-eDRL with $\beta = 2.0$.



(a) Average performance in 3 simulated days.

(b) Average performance in 4500 simulated days.

Figure 6.12: Average Global Acceptance ratio results.

For the other algorithms, maximal final GARs are between 61% and 71% and average final GARs are between 52% and 65%, except for HA-DRL and HA-eDRL with $\beta = 1.0$. Table 6.10 also shows that maximal and average final GARs of HA-DRL algorithms are generally higher than the equivalent HA-eDRL versions, the gap being never higher than 6%.

Algorithm	Maximal Final GAR (%)	Average Final GAR. (%)	Maximal Final TAR (%)	Average Final TAR (%)
DRL	65.90	52.15	77.69	55.82
HA-DRL,$\beta = 0.1$	70.56	55.57	81.45	59.83
HA-DRL,$\beta = 0.5$	69.80	60.12	73.56	65.33
HA-DRL,$\beta = 1.0$	64.89	28.23	72.14	41.69
HA-DRL,$\beta = 2.0$	81.84	81.62	83.91	82.21
eDRL	69.36	65.10	77.39	58.61
HA-eDRL,$\beta = 0.1$	69.25	52.22	76.34	48.45
HA-eDRL,$\beta = 0.5$	68.40	54.49	71.91	57.57
HA-eDRL,$\beta = 1.0$	61.46	27.33	67.05	33.10
HA-eDRL,$\beta = 2.0$	81.82	81.68	81.37	81.95

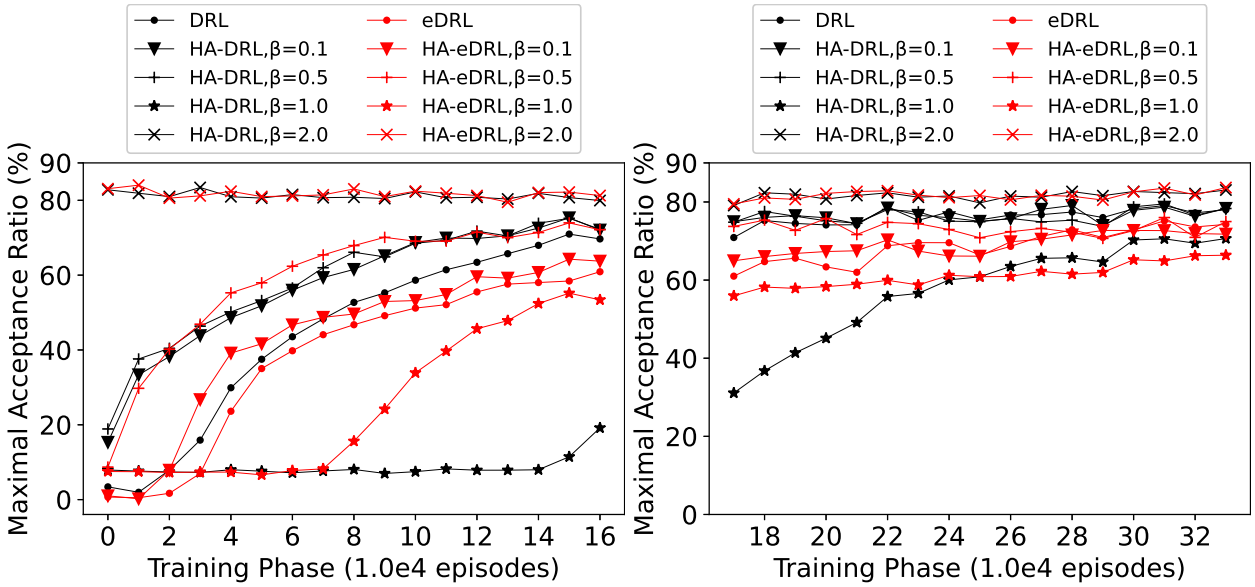
Table 6.10: Summary of evaluation results

The above results show that HA-DRL and HA-eDRL with $\beta = 2.0$ exhibit more robust performance with faster convergence and handle better network load variations than the other evaluated algorithms, notably when compared with the classical DRL approach. Indeed, when setting $\beta = 2.0$, the Heuristic Function computed on the basis of the HEU algorithm has strong influence on the actions chosen by the agent. Since the HEU algorithm often indicates a good action, this strong influence of the heuristic function helps the algorithms to become stable more quickly.

The addition of network load related features to the state observed by the eDRL algorithm helps improve the maximal and average final GAR when compared with DRL. However, this improvement is less significant than the improvement brought by the Heuristic Function acceleration as the eDRL algorithm does not achieve the same fast convergence and robustness than HA-DRL and HA-eDRL with $\beta = 2.0$.

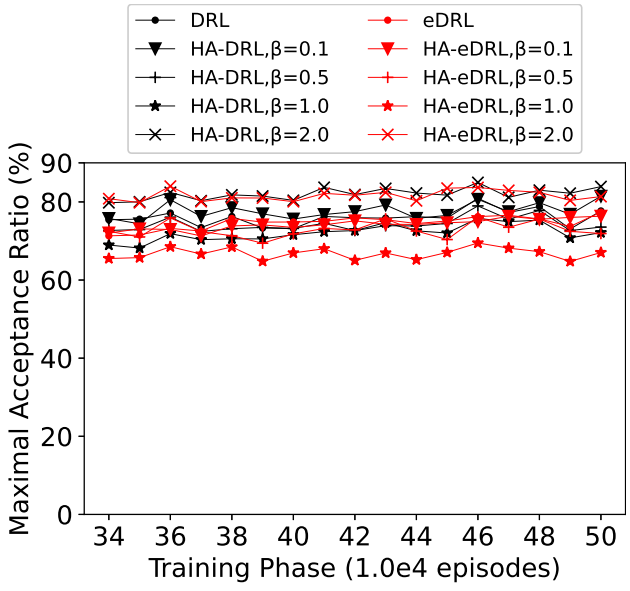
Acceptance ratio per training phase evaluation

Figure 6.13 and Figure 6.14 show that all algorithms need at least 16 training phases to reach a convergent maximal and average TAR, except for HA-DRL and HA-eDRL with $\beta = 2.0$ which need only one training phase (see Fig 6.13a and Fig 6.14a).



(a) Training Phases 0–16.

(b) Training Phases 17–33

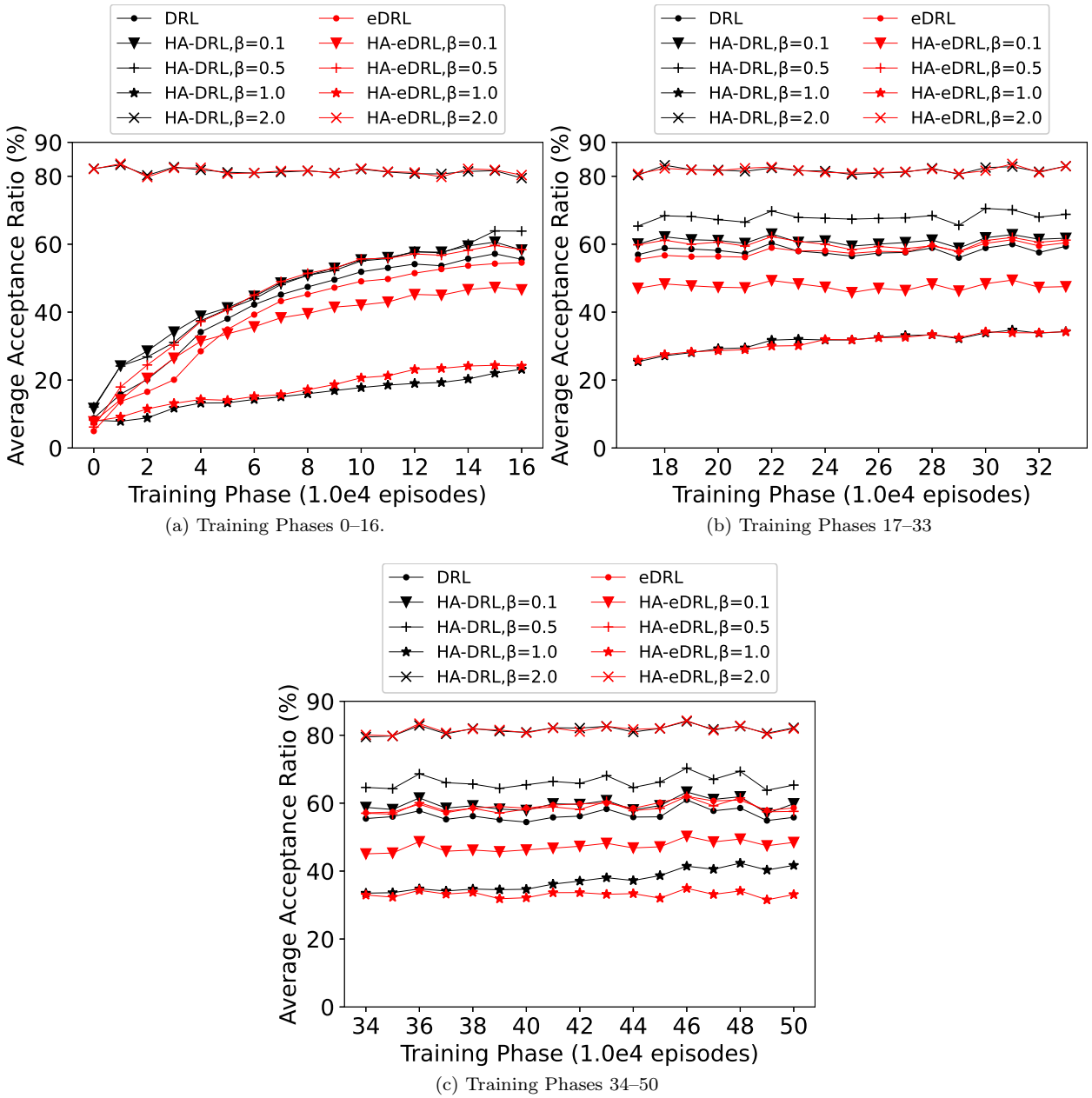


(c) Training Phases 34–50

Figure 6.13: Maximal Acceptance Ratio per training phase results.

The fast convergence of HA-DRL and HA-eDRL with $\beta = 2.0$ is due to the strong influence of the Heuristic Function in the choice of actions as explained in Section 6.6.3.

Figure 6.13a and Figure 6.13b illustrate that algorithms HA-DRL and HA-eDRL with $\beta = 0.5$, HA-DRL with $\beta = 0.1$ and DRL need around 16 phases to reach a convergent maximal TAR while Figure 6.13b and Figure 6.13c demonstrate that eDRL, HA-eDRL with $\beta \in \{0.1, 1.0\}$ and HA-DRL with $\beta = 1.0$ need around 34 phases.

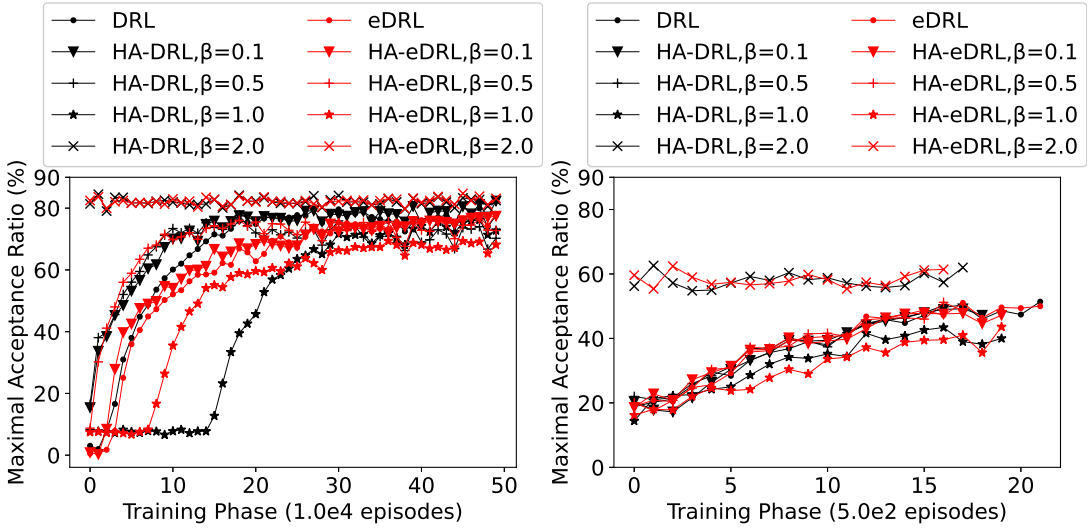


(a) Training Phases 0–16.

(b) Training Phases 17–33

(c) Training Phases 34–50

Figure 6.14: Average Acceptance Ratio per training phase results.



(a) Maximal performance for volatile NSPRs.

(b) Maximal performance for longterm NSPRs.

Figure 6.15: Acceptance Ratio per training phase results (per NSPR class).

Finally, Figure 6.14a and Figure 6.14b show that HA-DRL and HA-eDRL with $\beta \in \{0.1, 0.5\}$, eDRL, and DRL need around 16 phases to obtain a convergent average TAR while Figure 6.14b and Figure 6.14c show that HA-DRL and HA-eDRL with $\beta = 1.0$ need around 34 phases.

Table 6.10 shows that the only algorithms that reach a maximal and average final TAR (i.e, maximal and average TAR achieved at the end of training) higher than 80% are HA-DRL and HA-eDRL with $\beta = 2.0$. HA-DRL with $\beta = 0.1$ also attains a maximal final TAR higher than 80% but its average final TAR is less than 60%. HA-DRL with $\beta = 0.5$ has average final TAR higher than 65%, but its maximal final TAR is around 74%.

In addition, Table 6.10 shows that: HA-DRL algorithms have maximal final TAR generally higher than the equivalent HA-eDRL versions, the gap being never higher than 6%; eDRL and DRL have equivalent maximal final TAR but eDRL has average final TAR 3% higher. Figure 6.15b and Figure 6.15a show that all algorithms have better maximal and average TAR on volatile NSPRs than on long-term NSPRs.

This difference is arguably related to the number of arrivals of each requests and, therefore, the amount of training performed on each class of requests, since many more volatile requests arrive to be placed during the simulated period than long-term requests. These results reinforce the conclusion introduced in Section 6.6.3. The maximal TAR progression results (i.e., Figure 6.13) allow us to observe that in the best case, if a large number of training phases is given, all the algorithms attain a good performance.

However, the average TAR progression results (see Figure 6.14) show that only HA-DRL and HA-eDRL with $\beta = 2.0$ have robust performance. These algorithms are also the only one to have quick convergence being, among those evaluated, the most adapted to be used in practice.

6.6.4 Non-stationary network load scenario

In this section we evaluate the DRL and HA-eDRL algorithms introduced in section 6.5.3 under a non-stationary network load scenario.

Training process and hyper-parameters

We consider a training process with maximum duration of 6 hours for the considered algorithms. We perform seven independent runs of each algorithm to assess their average performance in terms of the metrics introduced below (see Section 6.6.4).

After performing Hyper-parameter search, we set the learning rates for the Actor and Critic networks of DRL and HA-DRL algorithms to $\alpha = 5 \times 10^{-5}$ and $\alpha' = 1.25 \times 10^{-3}$, respectively.

We program four versions of HA-DRL agents, each with a different value for the β parameter of the heuristic function formulation (see Section 6.5.2). We set in addition the parameters $\xi = 1$ and $\eta = 0$.

Network load calculation

Network loads are calculated using CPU resource but the analysis could easily be applied to RAM; we use the non-stationary network load model introduced by Definition 6.3.3. We consider two NSPR classes: i) a Volatile class and ii) a Long term class.

The differences between the two classes are related to their resource requirements and their lifespans as Volatile requests have 5 VNFs and a life-span of 20 simulation time units and Long-term requests have 10 VNFs and a life span of 500 simulation time units.

Network load change scenarios

We consider that the network runs in a standard regime under a network load being equal to 40% (i.e., $\rho = 0.4$) and that the NSPRs of each class generate half of the total load.

In each experiment, the learning agent is trained during approximately 4 hours for this network load regime. Then a stair-stepped network load change occurs. We simulated eight different network load change levels. Each network load change level is characterized by the addition of a certain amount of extra network load ranging from 10% to 80% (causing system overload).

Evaluation metrics

To characterize the performance of the placement algorithms, we consider the TAR metric introduced in Section 6.6.3. However, in this part of the evaluation, each training phase corresponds to 500 NSPR arrivals or 500 episodes. The metric is then calculated as follows: $\frac{\#\text{accepted NSPRs}}{500}$. This metric is retained here as it allows us to better observe the evolution of algorithm performance over time since it measures algorithm performance in independent parts (phases) of the training process without accumulating the performance of previous training phases.

Based on this metric, we identify three other important metrics used in our results discussion:

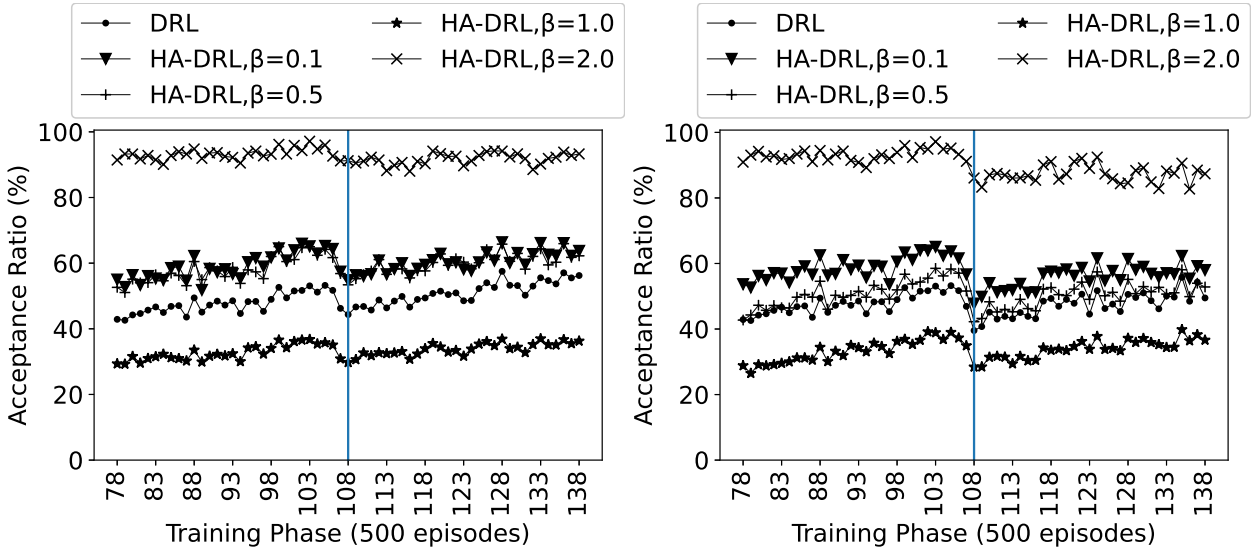
1. **Rupture TAR:** it is the TAR obtained in the training phase where the network load change occurs, i.e., the rupture phase;
2. **Last TAR:** it is the TAR obtained in the training phase that is prior to the rupture phase;
3. **Average TAR:** it is the average of the TARs obtained in the 30 phases preceding the rupture phase;
4. **TAR standard deviation:** it is the standard deviation of the TARs obtained in the 30 phases preceding the rupture phase;

Evaluation of the impact of network load change

Figures 6.16, 6.17, 6.18 and 6.19 capture the impact of different network load change levels on the TARs obtained by the different evaluated algorithms. The rupture phase is identified by a blue vertical line in the various figures. The performance of the algorithms seems to drop before the blue line. This impression is caused by expected performance variations since the training process is not yet completed and the performances of the algorithms are not yet stable after only a few hours of training.

We can observe in Figures 6.16, 6.17, 6.18 and 6.19 that with the reduced training time of 6 hours the only algorithm that has near optimal performance after 108 training phases is HA-DRL, with $\beta = 2.0$. This is due to the fact that the strong influence of the Heuristic Function helps the algorithm to become stable more quickly as already discussed in sections 6.6.2 and 6.6.3.

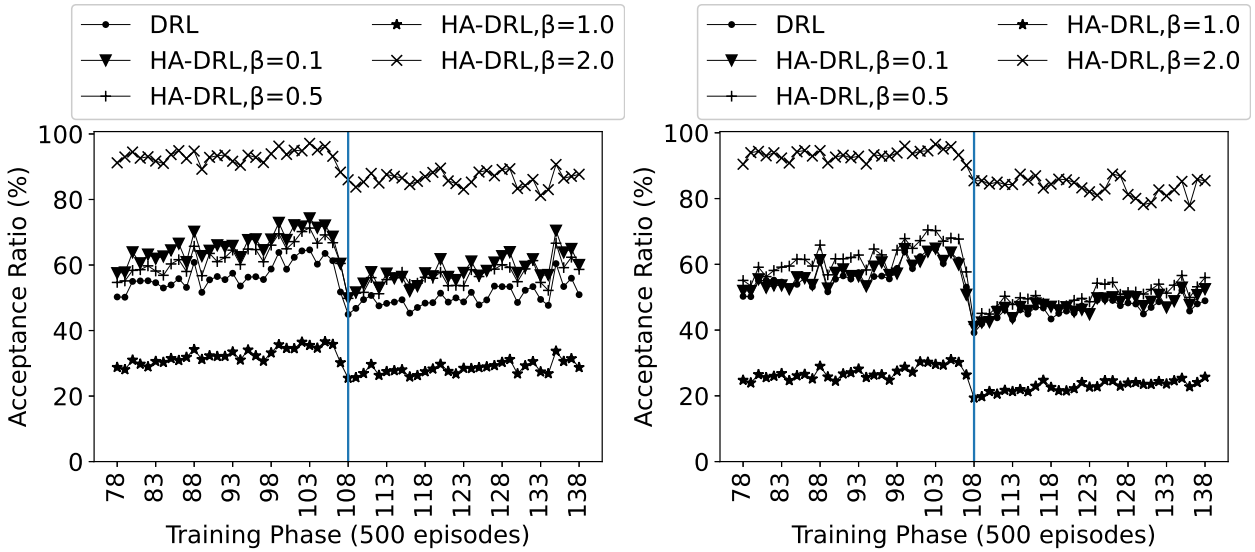
We can also observe by the shape of the different curves in Figures 6.16, 6.17, 6.18, and 6.19 that, as expected, all the algorithms have some variability in their performance during the training phases. In addition, these figures show that the performance of all the algorithms is affected at various levels by the network load change and that, generally speaking, the higher the amount of extra network load added, the lower is the TAR after the change.



(a) Addition of 10% of network load.

(b) Addition of 20% of network load.

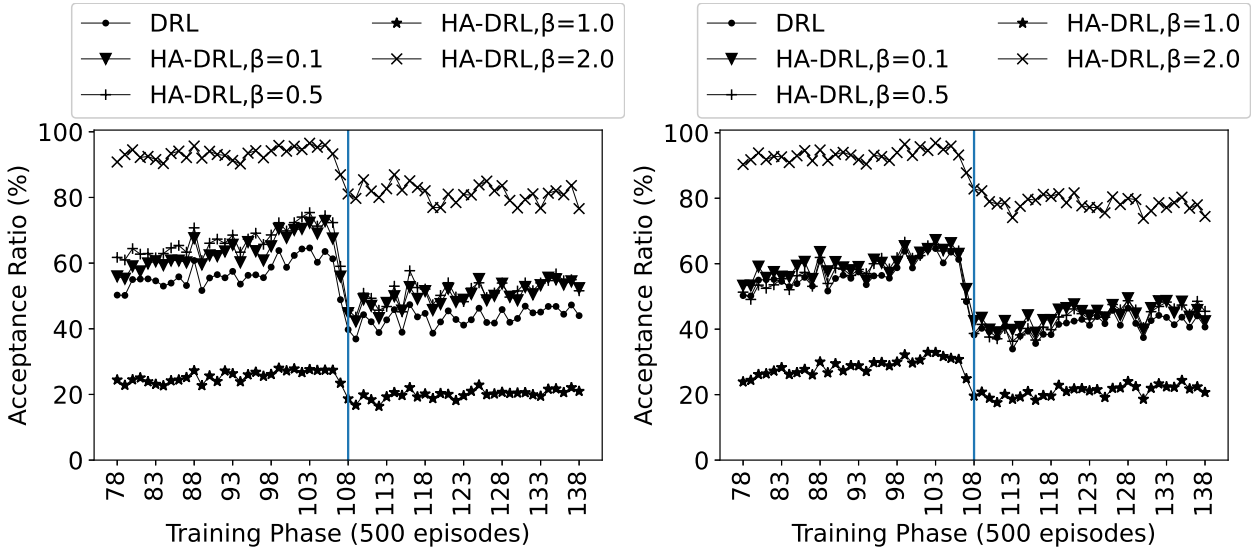
Figure 6.16: Evaluation of impact of network load disruption on TAR: under-loaded scenarios (1)



(a) Addition of 30% of network load.

(b) Addition of 40% of network load.

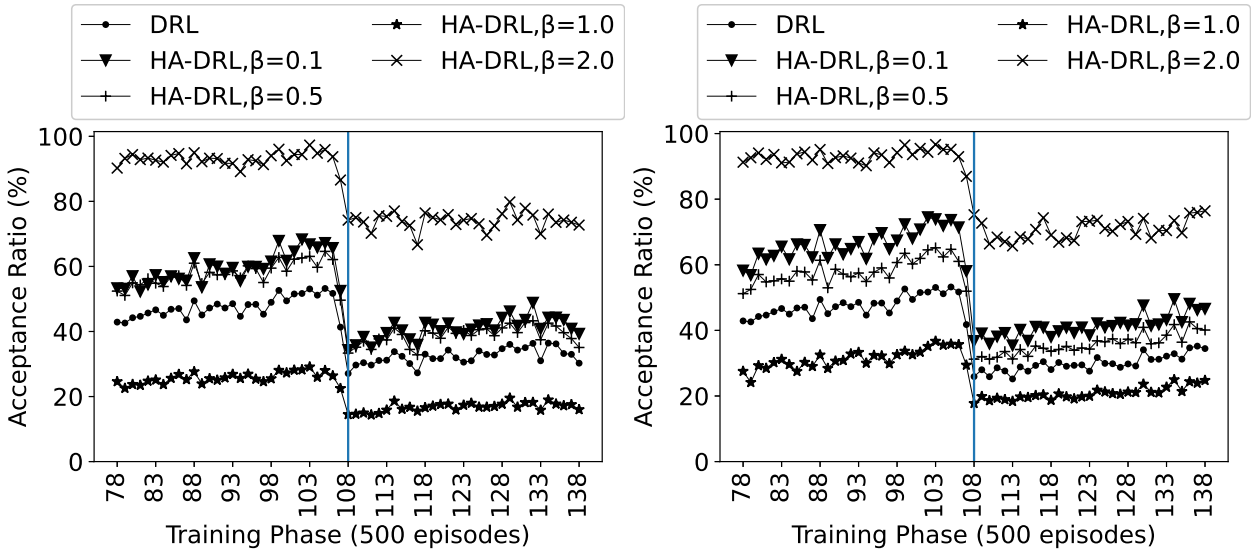
Figure 6.17: Evaluation of impact of network load disruption on TAR: under-loaded scenarios (2)



(a) Addition of 50% of network load.

(b) Addition of 60% of network load.

Figure 6.18: Evaluation of impact of network load disruption on TAR: critical scenarios



(a) Addition of 70% of network load.

(b) Addition of 80% of network load.

Figure 6.19: Evaluation of impact of network load disruption on TAR: overloaded scenarios

Finally, we can also see that the only algorithm to keep a near optimal performance even in overloaded scenarios shown in Figure 6.19 is HA-DRL, with $\beta = 2.0$. Tables 6.11, 6.12,6.13, 6.14, and 6.15 present other performance metrics related to the various evaluated algorithms. The columns "Rupture TAR - Avg. TAR" and "Rupture TAR - Last TAR" indicate how much the performance of the algorithms drops in the rupture phase when compared with the Average TAR and Last TAR, respectively.

The TAR Standard Deviation column indicates the TAR Standard Deviation metric described in Section 6.6.4. Those tables confirm that in general the performance gaps, i.e., the gaps between the Rupture TAR and Average or Last TAR, grow with the level of disruption for all algorithms. For instance, in the disruption level "+10", the performance gaps are never higher than 5%. But, in the change level "+80" the performance gap are never lower than 11%.

Network Load Disruption Level (%)	Rupture TAR - Avg. TAR (%)	Rupture TAR - Last TAR (%)	TAR Standard Deviation (%)
+10	-3.37	-1.89	3.10
+20	-8.19	-7.37	3.09
+30	-11.89	-6.83	4.17
+40	-17.68	-13.8	4.12
+50	-17.00	-9.11	4.32
+60	-18.50	-10.20	4.35
+70	-20.46	-14.26	3.30
+80	-21.65	-15.86	3.27

Table 6.11: DRL algorithm results

Network Load Disruption Level (%)	Rupture TAR - Avg. TAR (%)	Rupture TAR - Last TAR (%)	TAR Standard Deviation (%)
+10	-4.13	-2.60	4.07
+20	-11.02	-8.91	3.51
+30	-16.00	-10.54	4.50
+40	-16.28	-9.83	4.13
+50	-18.66	-11.14	5.05
+60	-17.02	-9.80	3.99
+70	-25.13	-18.20	4.93
+80	-29.41	-21.31	4.85

Table 6.12: HA-DRL, $\beta = 0.1$ algorithm results

In all the evaluated cases, the difference between the Rupture TAR and the Average TAR is higher than the TAR standard deviation. For instance, for the DRL algorithm, in network load disruption level of +50%, rupture TAR is 17% lower than Average TAR which is 3.94 times the TAR standard deviation.

The algorithm with the lower performance gaps is HA-DRL with $\beta = 1.0$ as we can see in columns "Rupture TAR - Avg. TAR" and "Rupture TAR - Last TAR" of Table 6.14. We can state that this algorithm has significantly better robustness then all the others as its performance gaps are significantly lower. However, HA-DRL with $\beta = 1.0$ has the worst TAR performance as shown in Figures 6.16, 6.17, 6.18 and 6.19, which reduces its applicability.

HA-DRL with $\beta = 2.0$ has the second better robustness and DRL the third as we can see on "Rupture TAR - Avg. TAR" and "Rupture TAR - Last TAR" columns of Tables 6.15 and 6.11, respectively. Even if the usage of the Heuristic Function has helped HA-DRL, with $\beta \in \{0.1, 0.5\}$ to achieve significantly better TARs than DRL, the influence of the Heuristic Function in these algorithms was not sufficient to allow to improve the robustness of the DRL algorithm against unpredictable network load disruptions (see Tables 6.12 and 6.13, respectively).

Network Load Disruption Level (%)	Rupture TAR - Avg. TAR (%)	Rupture TAR - Last TAR (%)	TAR Standard Deviation (%)
+10	-4.55	-3.43	3.95
+20	-8.80	-9.37	4.21
+30	-12.78	-10.66	4.59
+40	-20.33	-15.94	4.61
+50	-21.24	-13.43	4.56
+60	-19.46	-10.46	5.08
+70	-24.28	-16.26	3.75
+80	-26.78	-20.71	3.88

Table 6.13: HA-DRL, $\beta = 0.5$ algorithm results

Network Load Disruption Level (%)	Rupture TAR - Avg. TAR (%)	Rupture TAR - Last TAR (%)	TAR Standard Deviation (%)
+10	-2.96	-1.11	2.37
+20	-4.94	-6.49	3.50
+30	-6.93	-4.71	2.37
+40	-7.67	-7.00	1.97
+50	-6.80	-4.77	1.72
+60	-8.95	-5.29	2.45
+70	-11.25	-8.00	1.73
+80	-13.62	-11.69	2.89

Table 6.14: HA-DRL, $\beta = 1.0$ algorithm results

Network Load Disruption Level (%)	Rupture TAR - Avg. TAR (%)	Rupture TAR - Last TAR (%)	TAR Standard Deviation (%)
+10	-2.04	0.09	2.37
+20	-7.01	-5.09	3.50
+30	-7.15	-2.31	2.37
+40	-7.90	-4.69	1.97
+50	-12.13	-5.86	1.72
+60	-10.24	-4.94	2.45
+70	-18.83	-12.34	1.73
+80	-17.79	-11.69	2.89

Table 6.15: HA-DRL, $\beta = 2.0$ algorithm results

We can observe, however, that HA-DRL with $\beta = 2.0$ has better robustness against unpredictable network load changes than DRL as the performance gaps obtained with HA-DRL with $\beta = 2.0$ are significantly lower than the ones obtained with DRL as can be observed in columns "Rupture TAR - Avg. TAR" and "Rupture TAR - Last TAR" of Tables 6.15 and 6.11, respectively. These results confirm that HA-DRL with $\beta = 2.0$ is the algorithm among those evaluated that is the most adapted to be used in practice. Indeed, the algorithm presents not only the better TAR results and quick convergence but also robust performance.

Conclusion

In this Chapter, we present the second contribution of this PhD thesis, namely, *Automating Multi-objective Network Slice Placement with Machine Learning: a Heuristically Assisted Deep Reinforcement Learning solution*. The proposed DRL-heuristic algorithm proposed here gathers 6 main contributions: the proposed method i) enhances the scalability of existing ILP and heuristic approaches, ii) can cope with multiple optimization criteria, iii) combines DRL with GCN to automate feature extraction, iv) strengthens and accelerates the DRL learning process using an efficient placement optimization heuristic, v) supports multi-domain slice placement, and vi) supports the variations of network load in the placement of network slice requests.

We performed extensive evaluations considering different network load scenarios to assess the performance and robustness of the proposed DRL-heuristic algorithms. We started by evaluating the performance of the algorithm in a stationary network load scenario (see Section 6.6.2). The results of this evaluation first shown that the proposed HA-DRL approach yields good placement solutions in nearly real time, converges significantly faster than non-controlled DRL approaches, and yields better performance in terms of acceptance ratio than state-of-the-art heuristics and non-controlled DRL algorithms during and after the training phase.

We then introduced network load feature into the states observed by the DRL-based algorithms and evaluated them considering cycle-stationary network load conditions (see Section 6.6.3). In this phase we have considered four families of algorithms, pure-DRL and and HA-DRL algorithms but also eDRL and HA-eDRL (see Section 6.5.3) and by assuming that network load has periodic fluctuations we have shown how introducing network load states into the DRL algorithm together with a combination with the heuristic function yields very good results in terms of different performance metrics that remain stable in time.

As a final step, we have specifically evaluated the performance of DRL and HA-DRL algorithms in a non-stationary network load scenario with unpredictable changes. In line with the precedent conclusions, the numerical experiments performed in this step show that coupling DRL and heuristic functions yields good and stable results even under non stationary load conditions. Therefore, we believe that such an approach is relevant in real networks that are subject to unpredictable network load changes.

Chapter 7

Conclusion

Contents

This chapter concludes the PhD thesis manuscript with a summary of the main contributions in Section 7.1 and some perspectives for future work in Section 7.2.

7.1 Thesis contributions: a summary

In this PhD thesis, we have studied the optimization of NSP. This is an optimization problem which becomes very important for network operators with the emergence of Network Slicing. We have focused on three research questions defined after carefully analysing the state-of-the-art. The first research question is related to orchestration mechanisms with latency constraints: **How can we ensure QoS/QoE for NSUs in a converged network-Edge/MEC context?** Research on this topic has led us to propose the first contribution of this PhD thesis, which is described in **Chapter 4**, namely, *Enabling on-Edge and on-Network Slice Placement: an ILP-based Solution*. We have studied the NSP problem, particularly focusing on constraints implied by user location.

The proposed ILP-based solution gathers four main contributions. First, introducing a new E2E latency model that integrates the user location as an end-point of the Network Slice. This is done without the assumption that the preferred location for each Virtual Node is known (generalizing the hypothesis introduced by [59]). Then, tackling the scalability issues, we have proposed a model that can consider a higher number of NSUs and explore the possibility of grouping them (unlike [64]). Moreover, we have investigated to which extent Network Slices can be viewed as a generalization of SFCs since they can have a cyclic topology (generalizing at least [72] and [64] hypothesis). Finally, we did not set any restrictions on the placement location of two VNFs of the same Network Slice (generalizing [60] hypothesis).

We have implemented the proposed ILP inside an NSP solution [8] and evaluated the value of our proposition trough simulations [9]. The results show the relevance of the proposed E2E latency model since the simulations have shown that taking into account the user location as an end-point of the communication is essential in order to ensure the fulfillment of the E2E latency requirements. The results also show that a more scalable approach is needed to overcome complexity explosion when solving large-scale scenarios.

The scalability issue is covered by the second research question of this PhD thesis which is related to modeling and definition of large-scale network optimization algorithms: **What are the accurate optimization models and algorithms for Edge-enabled Network Slice placement in large-scale networks?** We have proposed the second contribution of this PhD thesis, which is described in **Chapter 5**, namely, *Optimizing Large-scale Network Slice Placement: a Heuristic using the Power of Two Choices*. The core of this contribution is an online heuristic to optimize NSP which gathers three main contributions: i) adaption to slice placement

requests on large-scale networks, ii) integration of Edge-specific and QoS constraints (i.e., E2E latency), and iii) reuse of the strength of P2C heuristic to implement selection policies.

We have implemented the proposed heuristic inside an NSP solution [11] and assessed its performance trough simulations [12]. Evaluation results have shown that the heuristic yields good solutions within a small execution time (1.96s for a PSN of 16128 nodes) and that the selection policies improve load balancing and reduce load of EDCs which improves the acceptance ratio in most simulation scenarios comparing to an online ILP-based placement algorithm. However, even if heuristic approaches have the advantage of being more scalable than ILP, they also show some limitations as they give sub-optimal solutions, they lack flexibility due to manual design of features and they have difficulties to handle multiple-optimization criteria.

To remedy these problems, we have investigated the potential of ML methods, specially DRL to overcome the convergence issues of ILPs while being more accurate and flexible than heuristics. More specifically, the third research question of this PhD thesis is related to AI/ML for automated and reliable network management: **How do we build automated and scalable algorithms to compute optimal placement decisions that are robust to changes in network behaviour?**

We have proposed then the third and final contribution of this PhD thesis, which is described in **Chapter 6**, namely, *Automating Multi-objective Network Slice Placement with Machine Learning: a Heuristically Assisted Deep Reinforcement Learning solution.* We have first designed a DRL approach to NSP Optimization which contains four main contributions: the proposed method i) enhances the scalability of existing ILP and heuristic approaches, ii) can cope with multiple optimization criteria, iii) combines DRL with GCN to automate feature extraction, and iv) supports multi-domain slice placement.

In order to increase robustness and reliability of the proposed DRL algorithm, based on the concept of Heuristically Accelerated Reinforcement Learning [13], we have introduced the concept of HA-DRL which strengthens and accelerates the DRL learning process using the efficient placement optimization heuristic described in Chapter 5 as an external controller. The first evaluation results performed under stationary network load conditions (see Section 4.3.6) have shown that the proposed HA-DRL approach yields good placement solutions in nearly real time, converges significantly faster than the proposed DRL approach, and yields better performance in terms of acceptance ratio than the proposed heuristic and DRL algorithms during and after the training phase.

To complement the study, we have introduced a DRL algorithm which considers the variations of network load in the placement of network slice requests in cycle-stationary fashion (see Section 6.3.2). We have shown that introducing network load states into the DRL algorithm together with a combination with the heuristic function yields very good and stable results in the cycle-stationary conditions.

Finally, we evaluated the proposed HA-DRL approach under non-stationary network load conditions (see Section 6.3.3). In line with the conclusions of the previous numerical experiments, this evaluation has also shown that coupling DRL and heuristic functions yields good and stable results even under non stationary load conditions. Therefore, we believe that such an approach is relevant in real networks that are subject to unpredictable network load changes.

7.2 Future work and perspectives

In this section, we present directions for future work as a perspective for the work conducted during this PhD thesis. Actually, many of the directions given below have been already started as part of the contribution to MON-B5G European project.

We provide 2 main directions for future work:

- Network Slice Placement over multiple technological domains
- Multi-agent Distributed DRL for Network Slice Placement

7.2.1 Network Slice Placement over multiple technological domains

To recall, we have considered a network divided in three domains: Edge, Core and Cloud as depicted by Figure 5.3 relying on PSN. What differentiates these three domains is the amount of resources they provide: EDCs are local DCs with small resources capacities, CDCs are regional DCs with medium resource capacities, and CCPs are national DCs with big resource capacities. These resource capacities are the capacities of servers inside the DCs but also the capacities of DC links.

In addition, as EDCs are closer to NSUs, they provide lower latency then CDCs which provide lower latency than CCPs. But, even though these three proposed levels of the network incorporate constraints related to the heterogeneity of resource capacities, we consider that they all provide the same type of resources. In other words, these three domains are heterogeneous with respect to their resource capacities but they are homogeneous with respect with the type of resources they provide.

To go beyond this proposal, one approach would be to consider multiple and heterogeneous domains within the PSN. It may also be multiple PSN with different natures, i.e., different technologies and network functions (Edge, RAN, Core, Cloud) which we believe is an important future work for this PhD thesis. It is important to investigate to which extent the inclusion of other technical domains such as RAN will impact the problem modeling and solving.

This extension is important in order to provide cross-domain network slice management, i.e., providing management of slices considering multiple heterogeneous technical domains, which is under study in the framework of MON-B5G European project. The proposed concept considers two management levels: i) intra-slice management, based on advanced ML-based algorithms that automate and optimize the placement, chaining, scaling, and migration of VNFs related to each slice; and ii) inter-slice control, to enable the coordination across multiple slices and domains.

7.2.2 Multi-agent Distributed DRL for Network Slice Placement

An interesting perspective for the work performed in this PhD thesis consists in studying the use of multi-agent DRL for NSP. Recent literature has shown the advantages of applying multi-agent DRL when solving a variety of problems [131]–[134]. In the case of the NSP problem, the use of a multi-agent DRL approach may be interesting in order to increase scalabilty of the system since this kind of approach allows to distribute the decision calculation and by consequent to reduce the amount of monitoring data transferred trough the network.

Two important aspects of this kind of methods are the protocol of communication/collaboration among the agents and the type of training performed. For instance [131] and [133] propose to rely on centralized controller to coordinate communication between agents—this controller is used only during training, which is performed offline—while [132] and [134] propose to rely on a peer-to-peer communication protocol and to perform online training.

An interesting direction for the work proposed in this PhD thesis is to extend the HA-DRL approach proposed in Chapter 6 to a multi-agent approach taking inspiration from the multi-agent actor-critic approach proposed in [131]. The authors of [131] design their solution using the Markov Decision Games framework [135] which can be summarized as follows:

- Let N be the number of agents, S be the number of states, A_1...,A_N be the sets of actions for each agent and O_1,...,O_N the sets of observations for each agent;
- Each agent use a stochastic policy $\pi_{\theta_i} : O_i \times A_i \rightarrow [0, 1]$ to choose the next action which produces the next state according to a state transition function $\tau : S \times A_1, ..., A_N \rightarrow S$;
- Each agent i receives a reward as a function of the state and agent's action $r_i : S \times A_i \rightarrow \mathbb{R}$ and receive a private observation $o_i : S \rightarrow O_i$;
- Each agent seeks to maximize its own total expected return (weighted sum of rewards).

We provide in the following a list of guidelines which can be used in order to design a multi-agent version of the HA-DRL algorithm described in Chapter 6 considering the multi-domain PSN studied in this PhD thesis.

1. Divide the PSN nodes in placement nodes (nodes in which VNFs can be placed such as servers) and routing nodes (nodes in which VNFs cannot be placed such as switches and routers);
2. Transform bandwidth capacities between servers and switches into local parameters which are stored independently in each node side. As a consequence, the connection between servers and switches are not seen by the routing algorithms;
3. Define Core, Cloud and Edge domains as only containing placement nodes;

4. Define Transport domain as only containing routing nodes and links/edges between these nodes;
5. Assign one decision engine to each technical domain, which are called the DE on Figure 7.1. Each decision engine has only access to local data (i.e., data associated to node or link/Edge located inside the domain).

If we take the PSN described in Figure 5.3 as an example and apply this guidelines we will have the following domains:

- Cloud domain: nodes $s1$ to $s16$;
- Core domain: nodes $s17$ to $s66$;
- Edge domain: nodes $s67$ to $s126$;
- Transport domain: nodes $r1$ to $r25$, $sw1$ to $sw21$ and existing links between these nodes.

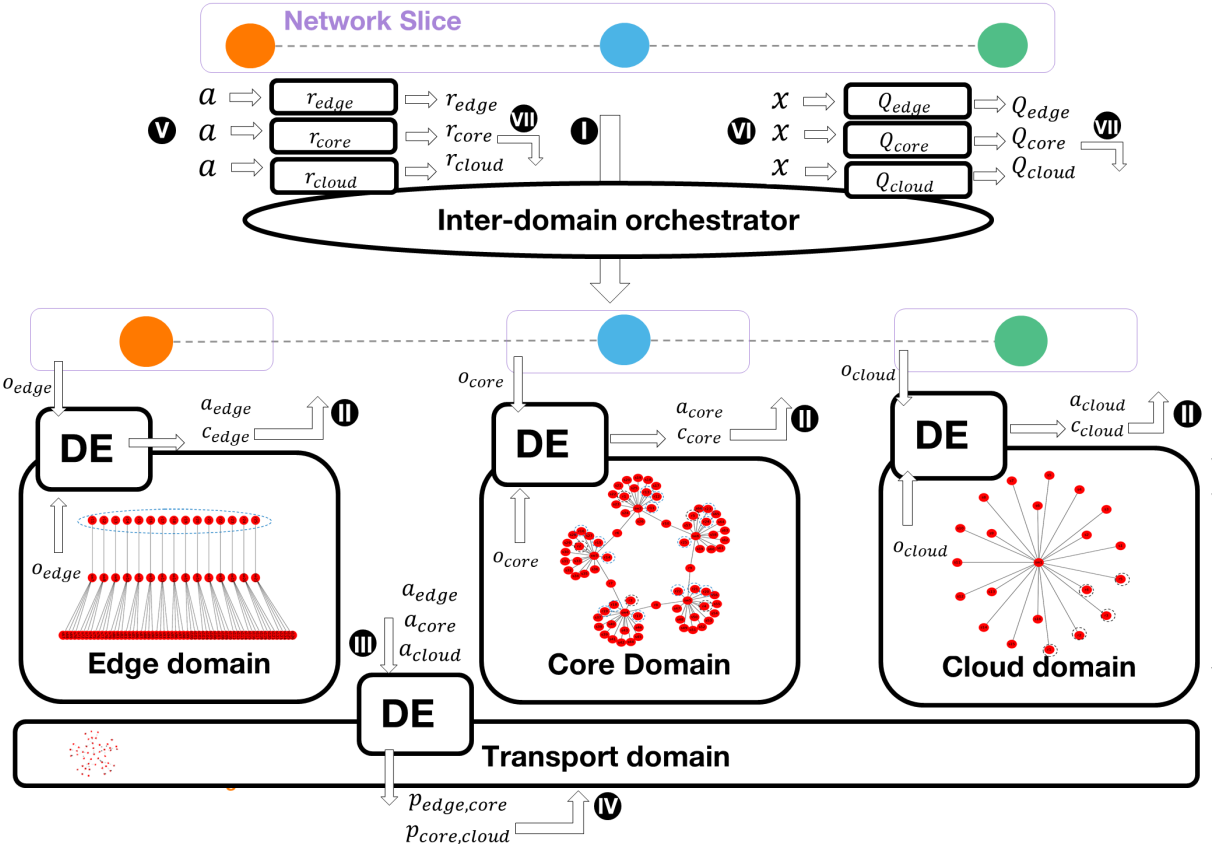


Figure 7.1: Archicture of a multi-agent DRL algorithm for NSP

In Figure 7.1 we provide an initial architecture for a multi-agent HA-DRL algorithm. We consider that in Edge, Core and Cloud domains, the decision engine computes a local policy, a local action, and a local constraint satisfaction rule. In the Transport domain, the decision engine computes a path to interconnect the VNF placed in distant nodes.

The inter-domain orchestrator is the one which ensures the coordination necessary between domains and it has different roles during training and execution phases. In the training phase the inter-domain orchestrator is responsible for (the numerals the items correspond to the numerals indicated on Figure 7.1):

i Decompose slice in sub-slices and forward data to domains;

ii Get local actions (a_d), get local constraint satisfaction variables (c_d) of all the domains. If $c_d = 1$ for all $d \in \{edge,\ core,\ cloud\}$:

iii Transfer a_d values to transport domain

iv Recuperate paths ($p_{edge,core}$, $p_{core,cloud}$) calculated by transport domain

v Compute reward for each decision engine

vi Compute state-value functions using input vector x with centralized critic networks where

$$x = \{o_{edge}, o_{core}, o_{transp}, o_{cloud}, a_{edge}, a_{core}, a_{tranp}, a_{cloud}\}$$

vii Transfer state-value functions and reward computations to local decision engine

During execution, the inter-domain orchestrator have a more reduced scope since no state-value function is computed. Only the actor network with training weights are used inside the decision engine. Hence, steps vi and vii are not needed. In order to develop the proposed strategy it is important to define an efficient way for decomposing a slice in sub-slices. An strategy based on the calculation of an optimal offloading threshold by a Genetic Algorithm was proposed in [136] and could be an interesting option.

Another interesting direction for the work proposed in this PhD thesis is to integrate the implementation of the proposed algorithms into a larger testbed with real 5G network functions.

The integration of the proposed HA-DRL algorithm with other components of slice orchestration process (e.g., monitoring system, analytics engine) and the test-bed implementation of the HA-DRL algorithm integrated with these other mechanisms is being studied as part of the MON-B5G project.

This is an important step to be done in order to understand how the algorithms perform when used in a real system.

7.3 Personal note and perspectives

To conclude my thesis, I would like to give a personal note about the key factors that I believe made my PhD such a rich and unique experience. These factors go from the technical knowledge I have gained to the soft and personal skills I have developed.

The first factor would be the technical knowledge. During my PhD, I was not only able to use my knowledge on optimization from the previous steps of my studies, but also to dig deep into related fields and acquire and extend the application of my knowledge to new areas in the larger telecommunications field. I would mention three fields here: Telco networks, software development, and machine learning.

Indeed, I had the opportunity to perform a large number of trainings, mostly provided by Orange and Sorbonne University, but also courses that I pursued on my own with the support and encouragement of my supervisors, such as the Machine Learning course provided by Stanford University via the Coursera platform.

It would also worth noting the number of hours spent reading not only research papers but also books in these fields, tutorials, participation to workshops and lectures, the time spent reading technical documents and project deliverables from different projects, and hours of discussions with colleagues from Orange and external partners that have contributed significantly to the extension of my knowledge in different technical fields. The second factor is related to the professional skills. In addition to the technical knowledge, as a researcher, I had also needed to develop a set of skills that come from a combination of theoretical training, practical experience, and interaction with other researchers. I had the opportunity to develop many of the skills I can show today during my PhD. I have learned to rely on scientific methods in my work. Indeed, throughout my PhD, I had the opportunity to practice my ability to address the following challenges: analyze the state-of-the-art and identify gaps, formulate a specific research question from an initial idea; develop a set of hypotheses related to that research question; formulate the question and set up experiments to evaluate those hypotheses; analyze and evaluate the results of the experiments; and finally, share the obtained results in an scientific manner through conferences. Learning this methodology as a systematic way of working in research was very important for me.

I also developed my scientific rigor and better understood how to analyze, criticize and evaluate not only the different technical and scientific documents related to my field of expertise but also my own productions

in terms of quality and added value. This skill was very important when analyzing the state-of-the-art and formulating my research questions, since my PhD was on a topic widely studied by the research community. It also allowed me to be accurate in my evaluations when I had the opportunity to review papers submitted by other researchers at different conferences.

Another important skill for a researcher is collaboration and teamwork. I had the opportunity to be part of two research teams (at Orange and at LIP6); to work on an internal research project at Orange and on two external collaborative research projects: the MON-B5G European project and the INFLUENCE research project in collaboration with Nokia Bell Labs. I was not only a PhD student participating in the projects but an active force of proposal. I participated in the technical deliverables, presented my work during the project meetings and improved it according to the partners' feedbacks. In the case of MON-B5G, there were 12 partners from different countries in Europe, so the project was also an opportunity to learn to work in collaboration with other researchers in an international environment.

Communication is also a key skill for a researcher and I was able to work on it during my PhD. I wrote papers for conferences and journals, and contributed to technical deliverables of the MON-B5G project. These were excellent opportunities to improve my writing efficiency, improve my ability to prioritize the information presented according to the objectives, and improve my ability to step back and synthesize. I have also presented works orally at different events (conferences, workshops and seminars). In these events, I had the opportunity to better understand how to effectively present my work by adapting my language and presentation format to different types of presentations such as technical sessions, demonstrations or scientific pitches.

By pursuing my PhD at Orange, I better understood the context of a research and development department within a large telecommunications company. It was a great opportunity to see the major stakes of a research activity conducted in this environment. Indeed, the research conducted in a research and development department of a company such as Orange is closely linked to the business and ecosystem challenges of the industry, which is in itself an environment under major transformations.

I would also like to talk about soft skills. I can't stress enough how important it was for me to improve my ability to manage stress. This skill was necessary to deal with the stress generated by the fact that research is inherently an uncertain activity in which we explore avenues with no guarantee on positive results. In addition, there is no doubt that stress management was very important to work effectively during the COVID pandemic. It was a very uncertain and stressful time and the fact that all activities, including conferences and meetings, were conducted remotely required a strong effort to deal with the generated stress and maintain a sufficient level of motivation.

Last but not least, I would like to highlight at least two other experiences that the PhD has given me and that I value. That is mainly living in France! It was an opportunity to meet incredible people and colleagues and open up to cultural exchanges that go from practicing French on a daily basis to becoming more familiar with French culture at work but also in other informal occasions. Second, to experience the daily life of a researcher. This was an excellent opportunity for me to learn many valuable professional lessons, both formally and informally, especially from my supervisors but also from my teammates.

These and many other factors made my PhD a superb experience. It required a lot of effort and resilience, but it gave me the opportunity to explore my potential, discover the exciting world of research, and confirm my tastes for long term exploratory activities in innovation. The PhD is definitely the experience of a lifetime!

After my PhD defense, I am planning to keep working as a research and development engineer. My current objective is to continue developing my expertise on Operations Research and Machine Learning focusing on the research and development of innovative data-driven decision support solutions.

Bibliography

[1] ETSI NFV ISG, "GS NFV 002 V1.2.1 Network Functions Virtualization (NFV);Architectural Framework", 2014. [Online]. Available: https://www.etsi.org/technologies-clusters/technologies/nfv.

[2] I. Afolabi, T. Taleb, K. Samdanis, A. Ksentini, and H. Flinck, "Network slicing and softwarization: a survey on principles, enabling technologies, and solutions", *IEEE Commun. Surveys Tuts.*, vol. 20, no. 3, pp. 2429–2453, 2018.

[3] ETSI NFV ISG, "GS NFV-MAN 001 V1.1.1 Network Functions Virtualisation (NFV); Management and Orchestration", 2014. [Online]. Available: https://www.etsi.org/technologies-clusters/technologies/nfv.

[4] N. Alliance, "Description of network slicing concept", *NGMN 5G P*, 2016.

[5] 3GPP, "TR 28.801 (V15.0.0): Telecommunication management; Study on management and orchestration of network slicing for next generation network", 2017. [Online]. Available: https://www.3gpp.org/specifications.

[6] ETSI NFV ISG, "GR NFV-EVE 012 V3.1.1 Network Functions Virtualisation (NFV); Evolution and Ecosystem; Report on Network Slicing Support", 2017. [Online]. Available: https://www.etsi.org/technologies-clusters/technologies/nfv.

[7] J. Gil Herrera and J. F. Botero, "Resource allocation in NFV: a comprehensive survey", *IEEE Trans. Netw. Service Manag.*, vol. 13, no. 3, pp. 518–532, Sep. 2016.

[8] J. J. Alves Esteves, A. Boubendir, F. Guillemin, and P. Sens, "Optimized network slicing proof-of-concept with interactive gaming use case", in *Proc. 2020 23rd Conf. Innovation Clouds, Int. Netw. Workshops (ICIN)*, 2020, pp. 150–152.

[9] ——, "Location-based data model for optimized network slice placement", in *Proc. 6th IEEE Conf. Netw. Softwarization (NetSoft)*, 2020, pp. 404–412.

[10] M. Mitzenmacher, "The power of two choices in randomized load balancing", *IEEE Trans. Parallel Distrib. Syst.*, vol. 12, no. 10, pp. 1094–1104, 2001.

[11] J. J. Alves Esteves, A. Boubendir, F. Guillemin, and P. Sens, "Edge-enabled optimized network slicing in large scale networks", in *Proc. 2020 11th Int. Conf. Netw. Future (NoF)*, 2020, pp. 129–131.

[12] ——, "Heuristic for edge-enabled network slicing optimization using the "power of two choices"", in *Proc. 2020 16th Int. Conf. Netw. Service Manag. (CNSM)*, 2020, pp. 1–9.

[13] R. A. Bianchi, C. H. Ribeiro, and A. H. R. Costa, "Heuristically accelerated reinforcement learning: theoretical and experimental results.", in *Proc. 20th Eur. Conf. Artif. Intell. (Frontiers in Artificial Intelligence and Applications, vol. 242)*., 2012, pp. 169–174.

[14] J. J. Alves Esteves, A. Boubendir, F. Guillemin, and P. Sens, "A heuristically assisted deep reinforcement learning approach for network slice placement", *Accepted to IEEE Trans. Netw. Service Manag.*, 2021. [Online]. Available: https://arxiv.org/pdf/2105.06741.pdf.

[15] ——, "Drl-based slice placement under realistic network load conditions", in *Proc. 2021 17th Int. Conf. Netw. Service Manag. (CNSM)*, 2021, pp. 1–3. [Online]. Available: https://arxiv.org/pdf/2109.12857.pdf.

[16] ——, "Drl-based slice placement under non-stationary conditions", in *Proc. 2021 17th Int. Conf. Netw. Service Manag. (CNSM)*, 2021, pp. 1–9. [Online]. Available: https://arxiv.org/pdf/2108.02495.pdf.

[17] ——, "On the robustness of controlled deep reinforcement learning for slice placement", *Accepted to Journal of Network and Systems Management*, Aug. 2021. [Online]. Available: https://arxiv.org/pdf/2108.02505.pdf.

[18] *MON-B5G web page*, https://www.monb5g.eu/, Accessed: 2021-09-30.

[19] ——, "Controlled deep reinforcement learning for optimized slice placement", in *2021 IEEE Int. Mediterranean Conf. Commun. Netw. (MeditCom)*, 2021, pp. 1–3. [Online]. Available: https://arxiv.org/pdf/2108.01544.pdf.

[20] ETSI NFV ISG, "GS NFV 003 V1.4.1 Network Functions Virtualization (NFV);Terminology for Main Concepts in NFV", 2018. [Online]. Available: https://www.etsi.org/technologies-clusters/technologies/nfv.

[21] R. J. Wilson, *Introduction to graph theory*. Pearson Education India, 1979.

[22] L. Taccari, "Integer programming formulations for the elementary shortest path problem", *Eur. J. Oper. Res.*, vol. 252, no. 1, pp. 122–130, Jul. 2016.

[23] E. W. Dijkstra *et al.*, "A note on two problems in connexion with graphs", *Numerische mathematik*, vol. 1, no. 1, pp. 269–271, 1959.

[24] F. S. Hillier, *Introduction to operations research*. Tata McGraw-Hill Education, 2012.

[25] P. J. Kolesar, "A branch and bound algorithm for the knapsack problem", *Manag. Sci.*, vol. 13, no. 9, pp. 723–735, May 1967.

[26] E.-G. Talbi, *Metaheuristics: from design to implementation*. John Wiley & Sons, 2009, vol. 74.

[27] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. Cambridge, MA, USA: MIT press, 2015.

[28] A. Fischer, J. F. Botero Vega, M. Duelli, D. Schlosser, X. Hesselbach Serra, and H. De Meer, "ALEVIN – a framework to develop, compare, and analyze virtual network embedding algorithms", *Open Access J. Electron. Commun. EASST*, pp. 1–12, Mar. 2011.

[29] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: a survey", *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 1888–1906, 2013.

[30] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A survey on service function chaining", *J. Netw. Comput. Appl.*, vol. 75, pp. 138–155, Nov. 2016.

[31] A. Laghrissi and T. Taleb, "A survey on the placement of virtual resources and virtual network functions", *IEEE Commun. Surveys Tuts.*, vol. 21, no. 2, pp. 1409–1434, 2018.

[32] A. Fischer, J. F. Botero, M. T. Beck, H. de Meer, and X. Hesselbach, "Virtual network embedding: a survey", *IEEE Commun. Surveys Tuts.*, vol. 15, no. 4, pp. 1888–1906, Feb. 2013.

[33] A. Banchs, G. de Veciana, V. Sciancalepore, and X. Costa-Perez, "Resource allocation for network slicing in mobile networks", *IEEE Access*, vol. 8, pp. 214 696–214 706, Nov. 2020.

[34] A. Othman and N. A. Nayan, "Efficient admission control and resource allocation mechanisms for public safety communications over 5g network slice", *Telecommun. Syst.*, vol. 72, no. 4, pp. 595–607, Jul. 2019.

[35] W. Kellerer, P. Kalmbach, A. Blenk, A. Basta, M. Reisslein, and S. Schmid, "Adaptable and data-driven softwarized networks: review, opportunities, and challenges", *Proc. IEEE*, vol. 107, no. 4, pp. 711–731, Feb. 2019.

[36] O. Soualah, M. Mechtri, C. Ghribi, and D. Zeghlache, "An efficient algorithm for virtual network function placement and chaining", in *Proc. 2017 14th IEEE Annu. Consum. Commun. Netw. Conf. (CCNC)*, IEEE, 2017, pp. 647–652.

[37] S. Khebbache, M. Hadji, and D. Zeghlache, "A multi-objective non-dominated sorting genetic algorithm for vnf chains placement", in *Proc. 2018 15th IEEE Annu. Cons. Commun. Netw. Conf. (CCNC)*, IEEE, 2018, pp. 1–4.

[38] C. Pham, N. H. Tran, S. Ren, W. Saad, and C. S. Hong, "Traffic-aware and energy-efficient vnf placement for service chaining: joint sampling and matching approach", *IEEE Trans. Serv. Comput.*, Feb. 2017.

[39] H. Yao, X. Chen, M. Li, P. Zhang, and L. Wang, "A novel reinforcement learning algorithm for virtual network embedding", *Neurocomputing*, vol. 284, pp. 1–9, Apr. 2018.

[40] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: substrate support for path splitting and migration", *ACM SIGCOMM Comput. Commun. Rev.*, vol. 38, no. 2, pp. 17–29, 2008.

[41] H. A. Shah and L. Zhao, "Multiagent deep-reinforcement-learning-based virtual resource allocation through network function virtualization in internet of things", *IEEE Internet Things J.*, vol. 8, no. 5, pp. 3410–3421, Sep. 2020.

[42] M. Dolati, S. B. Hassanpour, M. Ghaderi, and A. Khonsari, "DeepViNE: virtual network embedding with deep reinforcement learning", in *Proc. IEEE INFOCOM 2019 - IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2019, pp. 879–885.

[43] S. Oechsner and A. Ripke, "Flexible support of vnf placement functions in openstack", in *Proc. 2015 1st IEEE Conf. Netw. Softwarization (NetSoft)*, IEEE, 2015, pp. 1–6.

[44] Y. Xiao, Q. Zhang, F. Liu, *et al.*, "NFVdeep: adaptive online service function chain deployment with deep reinforcement learning", in *Proc. 2019 IEEE/ACM 27th Int. Symp. Qual. Service (IWQoS)*, 2019, pp. 1–10.

[45] F. Carpio, W. Bziuk, and A. Jukan, "Replication of virtual network functions: optimizing link utilization and resource costs", in *Proc. 2017 40th Int. Conv. Inf. Commun. Technol., Electron., Microelectronics (MIPRO)*, IEEE, 2017, pp. 521–526.

[46] Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li, "Automatic virtual network embedding: a deep reinforcement learning approach with graph convolutional networks", *"IEEE J. Sel. Areas Commun."*, vol. 38, no. 6, pp. 1040–1057, Jun. 2020.

[47] J.-J. Kuo, S.-H. Shen, H.-Y. Kang, D.-N. Yang, M.-J. Tsai, and W.-T. Chen, "Service chain embedding with maximum flow in software defined network and application to the next-generation cellular network architecture", in *Proc. IEEE INFOCOM 2017-IEEE Conf. Comput. Commun.*, IEEE, 2017, pp. 1–9.

[48] M. A. T. Nejad, S. Parsaeefard, M. A. Maddah-Ali, T. Mahmoodi, and B. H. Khalaj, "Vspace: vnf simultaneous placement, admission control and embedding", *IEEE J. Sel. Areas Commun.*, vol. 36, no. 3, pp. 542–557, Mar. 2018.

[49] L. Gao and G. N. Rouskas, "On congestion minimization for service chain routing problems", in *Proc. 2019 IEEE IEEE Int. Conf. Commun. (ICC)*, IEEE, 2019, pp. 1–6.

[50] F. Carpio, S. Dhahri, and A. Jukan, "Vnf placement with replication for load balancing in nfv networks", in *Proc. 2017 IEEE IEEE Int. Conf. Commun. (ICC)*, IEEE, 2017, pp. 1–6.

[51] A. Alleg, T. Ahmed, M. Mosbah, R. Riggio, and R. Boutaba, "Delay-aware vnf placement and chaining based on a flexible resource allocation approach", in *Proc. 2017 13th Int. Conf. Netw. Service Manag. (CNSM)*, IEEE, 2017, pp. 1–7.

[52] A. Marotta, E. Zola, F. D'Andreagiovanni, and A. Kassler, "A fast robust optimization-based heuristic for the deployment of green virtual network functions", *J. Netw. Comput. Applications*, vol. 95, pp. 42–53, Jul. 2017.

[53] A. Marotta, F. D'andreagiovanni, A. Kassler, and E. Zola, "On the energy cost of robustness for green virtual network function placement in 5g virtualized infrastructures", *Comput. Netw.*, vol. 125, pp. 64–75, Apr. 2017.

[54] P. Vizarreta, M. Condoluci, C. M. Machuca, T. Mahmoodi, and W. Kellerer, "Qos-driven function placement reducing expenditures in nfv deployments", in *Proc. 2017 IEEE Int. Conf. Commun. (ICC)*, IEEE, 2017, pp. 1–7.

[55] A. Fendt, S. Lohmuller, L. C. Schmelz, and B. Bauer, "A network slice resource allocation and optimization model for end-to-end mobile networks", in *Proc. 2018 IEEE 5G World Forum (5GWF)*, IEEE, 2018, pp. 262–267.

[56] V. S. Reddy, A. Baumgartner, and T. Bauschert, "Robust embedding of vnf/service chains with delay bounds", in *Proc. 2016 IEEE Conf. Netw. Funct. Virtualization Softw. Defined Netw. (NFV-SDN)*, IEEE, 2016, pp. 93–99.

[57] G. Wang, G. Feng, W. Tan, S. Qin, R. Wen, and S. Sun, "Resource allocation for network slices in 5g with network resource pricing", in *Proc. GLOBECOM 2017-IEEE Global Commun. Conf.*, IEEE, 2017, pp. 1–6.

[58] A. Rkhami, Y. Hadjadj-Aoul, and A. Outtagarts, "Learn to improve: a novel deep reinforcement learning approach for beyond 5G network slicing", in *Proc. 2021 IEEE 18th Annu. Consum. Commun. Netw. Conf. (CCNC)*, 2021, pp. 1–6.

[59] N. M. K. Chowdhury, M. R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping", in *Proc. IEEE INFOCOM 2009*, IEEE, 2009, pp. 783–791.

[60] L. Gong, H. Jiang, Y. Wang, and Z. Zhu, "Novel location-constrained virtual network embedding lc-vne algorithms towards integrated node and link mapping", *IEEE/ACM Trans. Netw.*, vol. 24, no. 6, pp. 3648–3661, 2016.

[61] C. Papagianni, P. Papadimitriou, and J. S. Baras, "Rethinking service chain embedding for cellular network slicing", in *Proc. 2018 IFIP Netw. Conf. (IFIP Networking) Workshops*, IEEE, 2018, pp. 1–9.

[62] D. Bhamare, M. Samaka, A. Erbad, R. Jain, L. Gupta, and H. A. Chan, "Optimal virtual network function placement in multi-cloud service function chaining architecture", *Comput. Commun.*, vol. 102, pp. 1–16, 2017.

[63] S. Khebbache, M. Hadji, and D. Zeghlache, "Virtualized network functions chaining and routing algorithms", *Comput. Netw.*, vol. 114, pp. 95–110, 2017.

[64] D. Harutyunyan, N. Shahriar, R. Boutaba, and R. Riggio, "Latency-aware service function chain placement in 5g mobile networks", in *Proc. IEEE Conf. Netw. Softwarization (NetSoft)*, 2019.

[65] M. Mechtri, C. Ghribi, and D. Zeghlache, "Vnf placement and chaining in distributed cloud", in *Proc. 2016 IEEE 9th Int. Conf. Cloud Comput. (CLOUD)*, IEEE, 2016, pp. 376–383.

[66] P. T. A. Quang, A. Bradai, K. D. Singh, and Y. Hadjadj-Aoul, "Multi-domain non-cooperative VNF-FG embedding: a deep reinforcement learning approach", in *Proc. IEEE INFOCOM 2019 - IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2019, pp. 886–891.

[67] O. Soualah, M. Mechtri, C. Ghribi, and D. Zeghlache, "A green vnf-fg embedding algorithm", in *Proc. 2018 4th IEEE Conf. Netw. Softwarization and Workshops (NetSoft)*, IEEE, 2018, pp. 141–149.

[68] B. Addis, D. Belabed, M. Bouet, and S. Secci, "Virtual network functions placement and routing optimization", in *Proc. 2015 IEEE 4th Int. Conf. Cloud Netw. (CloudNet)*, IEEE, 2015, pp. 171–177.

[69] P. T. A. Quang, Y. Hadjadj-Aoul, and A. Outtagarts, "A deep reinforcement learning approach for vnf forwarding graph embedding", *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 4, pp. 1318–1331, Dec. 2019.

[70] S. Khebbache, M. Hadji, and D. Zeghlache, "Scalable and cost-efficient algorithms for vnf chaining and placement problem", in *Proc. 2017 20th Conf. Innov. Clouds Internet Netw. (ICIN)*, IEEE, 2017, pp. 92–99.

[71] J. Cao, Y. Zhang, W. An, X. Chen, J. Sun, and Y. Han, "Vnf-fg design and vnf placement for 5g mobile networks", *Sci. China Info. Sci.*, vol. 60, no. 4, p. 040 302, 2017.

[72] B. Spinnewyn, P. H. Isolani, C. Donato, J. F. Botero, and S. Latré, "Coordinated service composition and embedding of 5g location-constrained network functions", *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 4, pp. 1488–1502, 2018.

[73] A. C. Adamuthe, R. M. Pandharpatte, and G. T. Thampi, "Multiobjective virtual machine placement in cloud environment", in *Proc. 2013 Int. Conf. Cloud Ubiquitous Comput. Emerg. Technol.*, IEEE, 2013, pp. 8–13.

[74] D. Dietrich, C. Papagianni, P. Papadimitriou, and J. S. Baras, "Network function placement on virtualized cellular cores", in *Proc. 2017 9th Int. Conf. Commun. Syst. Netw. (COMSNETS)*, IEEE, 2017, pp. 259–266.

[75] V. Sciancalepore, F. Z. Yousaf, and X. Costa-Perez, "Z-torch: an automated nfv orchestration and monitoring solution", *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 4, pp. 1292–1306, 2018.

[76] P. T. A. Quang, A. Bradai, K. D. Singh, G. Picard, and R. Riggio, "Single and multi-domain adaptive allocation algorithms for vnf forwarding graph embedding", *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 1, pp. 98–112, 2018.

[77] F. B. Jemaa, G. Pujolle, and M. Pariente, "Qos-aware vnf placement optimization in edge-central carrier cloud architecture", in *Proc. 2016 IEEE Global Commun. Conf. (GLOBECOM)*, IEEE, 2016, pp. 1–7.

[78] K. Li, J. Wu, and A. Blaisse, "Elasticity-aware virtual machine placement for cloud datacenters", in *Proc. 2013 IEEE 2nd Int. Conf. Cloud Netw. (CloudNet)*, IEEE, 2013, pp. 99–107.

[79] M. Bagaa, T. Taleb, A. Laghrissi, and A. Ksentini, "Efficient virtual evolved packet core deployment across multiple cloud domains", in *Proc. 2018 IEEE Wireless Comun. Netw. Conf. (WCNC)*, IEEE, 2018, pp. 1–6.

[80] S. Agarwal, F. Malandrino, C. F. Chiasserini, and S. De, "Vnf placement and resource allocation for the support of vertical services in 5g networks", *IEEE/ACM Trans. Netw.*, vol. 27, no. 1, pp. 433–446, 2019.

[81] M. Bunyakitanon, A. P. Da Silva, X. Vasilakos, R. Nejabati, and D. Simeonidou, "Auto-3p: an autonomous vnf performance prediction & placement framework based on machine learning", *Comput. Netw.*, vol. 181, p. 107 433, Nov. 2020.

[82] R. Zhou, Z. Li, and C. Wu, "An efficient online placement scheme for cloud container clusters", *IEEE J. Sel. Areas Commun.*, vol. 37, no. 5, pp. 1046–1058, 2019.

[83] S. Messaoud, A. Bradai, O. B. Ahmed, P. T. A. Quang, M. Atri, and M. S. Hossain, "Deep federated q-learning-based network slicing for industrial iot", *IEEE Trans. Ind. Informat.*, vol. 17, no. 8, pp. 5572–5582, Oct. 2020.

[84] Y. Sun, G. Feng, L. Zhang, P. V. Klaine, M. A. Iinran, and Y.-C. Liang, "Distributed learning based handoff mechanism for radio access network slicing with data sharing", in *Proc. 2019 IEEE Int. Conf. Commun. (ICC)*, IEEE, 2019, pp. 1–6.

[85] H. Wang, Y. Wu, G. Min, J. Xu, and P. Tang, "Data-driven dynamic resource scheduling for network slicing: a deep reinforcement learning approach", *Inf. Sci.*, vol. 498, pp. 106–116, Sep. 2019.

[86] W. da Silva Coelho, A. Benhamiche, N. Perrot, and S. Secci, "On the impact of novel function mappings, sharing policies, and split settings in network slice design", in *Proc. 2020 16th Int. Conf. Netw. Service Manag. (CNSM)*, 2020, pp. 1–9.

[87] Y. L. Lee, J. Loo, T. C. Chuah, and L.-C. Wang, "Dynamic network slicing for multitenant heterogeneous cloud radio access networks", *IEEE Trans. Wireless Commun.*, vol. 17, no. 4, pp. 2146–2161, Jan. 2018.

[88] A. Baumgartner, T. Bauschert, A. A. Blzarour, and V. S. Reddy, "Network slice embedding under traffic uncertainties — a light robust approach", in *Proc. 2017 13th Int. Conf on Netw. Service Manag. (CNSM)*, IEEE, 2017, pp. 1–5.

[89] S. Khebbache, M. Hadji and D. Zeghlache, "Dynamic placement of extended service function chains: steiner-based approximation algorithms", in *Proc. 2018 IEEE 43rd Conf. Local Comput. Netw. (LCN)*, IEEE, 2018, pp. 307–310.

[90] F. Esposito, I. Matta, and V. Ishakian, "Slice embedding solutions for distributed service architectures", *ACM Comput. Surv. (CSUR)*, vol. 46, no. 1, pp. 1–29, 2013.

[91] O. Houidi, O. Soualah, W. Louati, M. Mechtri, D. Zeghlache, and F. Kamoun, "An efficient algorithm for virtual network function scaling", in *Proc. GLOBECOM 2017 IEEE Global Commun. Conf.*, IEEE, 2017, pp. 1–7.

[92] H. Lee, S. H. Lee, and T. Q. Quek, "Deep learning for distributed optimization: applications to wireless resource management", *IEEE J. Sel. Areas Commun.*, vol. 37, no. 10, pp. 2251–2266, Aug. 2019.

[93] S. Wang, R. Urgaonkar, M. Zafer, T. He, K. Chan, and K. K. Leung, "Dynamic service migration in mobile edge-clouds", in *Proc. 2015 IFIP Netw. Conf. (IFIP Networking)*, IEEE, 2015, pp. 1–9.

[94] B. Han, V. Sciancalepore, D. Feng, X. Costa-Perez, and H. D. Schotten, "A utility-driven multi-queue admission control solution for network slicing", in *Proc. IEEE INFOCOM 2019-IEEE Conf. Comput. Commun.*, IEEE, 2019, pp. 55–63.

[95] B. Han, D. Feng, and H. D. Schotten, "A markov model of slice admission control", *IEEE Netw. Lett.*, vol. 1, no. 1, pp. 2–5, Oct. 2018.

[96] E. Amaldi, S. Coniglio, A. M. Koster, and M. Tieves, "On the computational complexity of the virtual network embedding problem", *Electron. Notes Discrete Math.*, vol. 52, pp. 213–220, Jun. 2016.

[97] N. Alliance, "5g white paper", *Next generation mobile networks, white paper*, vol. 1, 2015.

[98] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to model an internetwork", in *Proc. IEEE INFOCOM'96. Conf. Comput. Commun.*, IEEE, vol. 2, 1996, pp. 594–602.

[99] S. Orlowski, R. Wessäly, M. Pióro, and A. Tomaszewski, "Sndlib 1.0—survivable network design library", *Netw. Int. J.*, vol. 55, no. 3, pp. 276–286, 2010.

[100] P. T. A. Quang, A. Bradai, K. D. Singh, G. Picard, and R. Riggio, "Single and multi-domain adaptive allocation algorithms for vnf forwarding graph embedding", *IEEE Trans. Netw. Service Manag.*, vol. 16, no. 1, pp. 98–112, 2018.

[101] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering", in *Proc. 30th Int. Conf. Neural Processing Syst. (NIPS)*, 2016, pp. 3844–3852.

[102] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks", in *Proc. 5th Int. Conf. Learn. Representations (ICLR)*, 2017, pp. 1–14.

[103] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. Van Den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks", in *Proc. European Semantic Web Conf.*, Springer, 2018, pp. 593–607.

[104] V. Mnih, A. P. Badia, M. Mirza, *et al.*, "Asynchronous methods for deep reinforcement learning", in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2016, pp. 1928–1937.

[105] R. A. Bianchi, C. H. Ribeiro, and A. H. Costa, "Accelerating autonomous learning by using heuristic selection of actions", *J. Heuristics*, vol. 14, no. 2, pp. 135–168, May 2008.

[106] J. Supancic and D. Ramanan, "Tracking as online decision-making: learning a policy from streaming videos with reinforcement learning", in *Proc. 2017 IEEE Int. Conf. on Comput. Vision (ICCV)*, 2017, pp. 322–331.

[107] R. A. Bianchi, M. F. Martins, C. H. Ribeiro, and A. H. Costa, "Heuristically-accelerated multiagent reinforcement learning", *IEEE Trans. Cybern.*, vol. 44, no. 2, pp. 252–265, Feb. 2014.

[108] R. A. Bianchi, P. E. Santos, I. J. Da Silva, L. A. Celiberto, and R. L. de Mantaras, "Heuristically accelerated reinforcement learning by means of case-based reasoning and transfer learning", *J. Intelligent Robotic Syst.*, vol. 91, no. 2, pp. 301–312, Oct. 2017.

[109] N. Morozs, T. Clarke, and D. Grace, "Heuristically accelerated reinforcement learning for dynamic secondary spectrum sharing", *IEEE Access*, vol. 3, pp. 2771–2783, Dec. 2015.

[110] D. Bertsimas and A. Thiele, "Robust and data-driven optimization: modern decision making under uncertainty", in *Models, methods, and appli. innovative decision making*, INFORMS, 2006, pp. 95–122.

[111] D. Bertsimas and M. Sim, "The price of robustness", *Operations Res.*, vol. 52, no. 1, pp. 35–53, Feb. 2004.

[112] A. M. Koster, M. Kutschka, and C. Raack, "On the robustness of optimal network designs", in *Proc. 2011 IEEE Int. Conf. Commun. (ICC)*, IEEE, 2011, pp. 1–5.

[113] M. Fischetti and M. Monaci, "Light robustness", in *Robust online large-scale optim.* Springer, 2009, pp. 61–84.

[114] A. Schöbel, "Generalized light robustness and the trade-off between robustness and nominal quality", *Mathematical Methods Operations Res.*, vol. 80, no. 2, pp. 161–191, Jun. 2014.

[115] M. Shafique, M. Naseer, T. Theocharides, *et al.*, "Robust machine learning systems: challenges, current trends, perspectives, and the road ahead", *IEEE Design & Test*, vol. 37, no. 2, pp. 30–57, Apr. 2020.

[116] R. R. O. Al-Nima, T. Han, S. A. M. Al-Sumaidaee, T. Chen, and W. L. Woo, "Robustness and performance of deep reinforcement learning", *Applied Soft Comput.*, vol. 105, p. 107 295, Jul. 2021.

[117] P. Sun, J. Lan, J. Li, Z. Guo, and Y. Hu, "Combining deep reinforcement learning with graph neural networks for optimal vnf placement", *IEEE Commun. Letters*, vol. 25, no. 1, pp. 176–180, Jan. 2021.

[118] J. Pei, P. Hong, M. Pan, J. Liu, and J. Zhou, *IEEE J. Sel. Areas Commun.*,

[119] D. Bega, M. Gramaglia, M. Fiore, A. Banchs, and X. Costa-Perez, "Deepcog: cognitive network management in sliced 5g networks with deep learning", in *Proc. IEEE INFOCOM 2019 - IEEE Conf. Comput. Commun. Workshops (INFOCOM WKSHPS)*, 2019, pp. 280–288.

[120] I. Alawe, Y. Hadjadj-Aoul, A. Ksentinit, P. Bertin, C. Viho, and D. Darche, "An efficient and lightweight load forecasting for proactive scaling in 5g mobile networks", in *Proc. 2018 IEEE Conf. Standards Commun. Netw. (CSCN)*, 2018, pp. 1–6.

[121] G. M. Dias, M. Nurchis, and B. Bellalta, "Adapting sampling interval of sensor networks using on-line reinforcement learning", in *Proc. 2016 IEEE 3rd World Forum Internet Things (WF-IoT)*, IEEE, 2016, pp. 460–465.

[122] G. M. Dias, T. Adame, B. Bellalta, and S. Oechsner, "A self-managed architecture for sensor networks based on real time data analysis", in *Proc. 2016 Future Technol. Conf. (FTC)*, IEEE, 2016, pp. 1297–1299.

[123] H. Malik, A. S. Malik, and C. K. Roy, "A methodology to optimize query in wireless sensor networks using historical data", *J. Ambient Intell. Humanized Comput.*, vol. 2, no. 3, p. 227, Jun. 2011.

[124] Y.-A. Le Borgne, S. Santini, and G. Bontempi, "Adaptive model selection for time series prediction in wireless sensor networks", *Signal Process.*, vol. 87, no. 12, pp. 3010–3020, Dec. 2007.

[125] ETSI ISG NFV, "GS ZSM 001 V1.1.1 Zero-touch network and Service Management (ZSM);Requirements based on documented scenarios", 2019. [Online]. Available: `https://www.etsi.org/technologies-clusters/technologies/nfv`.

[126] F. Slim, F. Guillemin, and Y. Hadjadj-Aoul, "CLOSE: a costless service offloading strategy for distributed edge cloud", in *Proc. 2018 15th IEEE Annu. Cons. Commun. Netw. Conf. (CCNC)*, 2018, pp. 1–6.

[127] F. Slim, F. Guillemin, A. Gravey, and Y. Hadjadj-Aoul, "Towards a dynamic adaptive placement of virtual network functions under ONAP", in *Proc. 2017 IEEE Conf. on Netw. Function Virtualization Softw. Defined Netw. (NFV-SDN)*, 2017, pp. 210–215.

[128] I. ILOG, "Cplex optimization studio", 2014.

[129] J. Bezanson, S. Karpinski, V. B. Shah, and A. Edelman, "Julia: a fast dynamic language for technical computing", *arXiv preprint arXiv:1209.5145*, 2012.

[130] E. W. Zegura, K. L. Calvert, and M. J. Donahoo, "A quantitative comparison of graph-based models for internet topology", *IEEE/ACM Trans. Netw.*, vol. 5, no. 6, pp. 770–783, 1997.

[131] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch, "Multi-agent actor-critic for mixed cooperative-competitive environments", in *Proc. 31st Int. Conf. Neural Inf. Process. Syst.*, ser. NIPS'17, 2017, pp. 6382–6393.

[132] K. Zhang, Z. Yang, H. Liu, T. Zhang, and T. Basar, "Fully decentralized multi-agent reinforcement learning with networked agents", in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2018, pp. 5872–5881.

[133] S. Iqbal and F. Sha, "Actor-attention-critic for multi-agent reinforcement learning", in *Proc. Int. Conf. Mach. Learn.*, PMLR, 2019, pp. 2961–2970.

[134] H.-T. Wai, Z. Yang, Z. Wang, and M. Hong, "Multi-agent reinforcement learning via double averaging primal-dual optimization", in *Proc. Neural Info. Process. Syst. (NeurIPS)*, 2018.

[135] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning", in *Mach. Learn. Proc. 1994*, Elsevier, 1994, pp. 157–163.

[136] F. Slim, "Design and implementation of resource allocation algorithms for a network operating", Ph.D. dissertation, Ecole nationale supérieure Mines-Télécom Atlantique, 2018.