UNIVERSIDAD DE GRANADA

MASTER'S THESIS

TELECOMMUNICATION ENGINEERING

Optimized radio resource allocation in 5G using Deep Q-Networks (DQN)

Author
David Fernández Martínez

Supervisors
Pablo José Ameigeiras Gutiérrez
Lorena Chinchilla Romero

Higher Technical School of Information Technology and Telecommunications Engineering

—

Granada, 5^{th} September 2024

Optimized radio resource allocation in 5G using Deep Q-Networks (DQN)

Author
David Fernández Martínez

Supervisors
Pablo José Ameigeiras Gutiérrez
Lorena Chinchilla Romero

Optimized radio resource allocation in 5G using Deep Q-Networks (DQN)

David Fernández Martínez

KeyWords: 5G, Radio access network, Physical resource block, Machine Learning, Neural networks, Deep reinforcement learning, Deep Q-Networks, ϵ-greedy

Abstract

The transition from 4G to 5G has not only been marked by growth in traffic, as has been usual in the last decade, but has also been accompanied by an increase in the number of devices and services. Services like Massive Internet of Things (mIoT), Massive Machine Type Communication (mMTC), ultra-Reliable Low Latency Communication (uRLLC), and Enhanced Mobile Broadband (eMBB) necessitate the implementation of network slicing, which considerably increase network complexity.

To accommodate emerging services and successfully transition to future networks, operators must tackle the challenge of managing and optimizing their network infrastructure. In this context, Machine Learning (ML) and Artificial Intelligence (AI) emerge as promising technologies capable of enhancing the efficiency of numerous processes that have traditionally relied on the expertise of human specialists. The integration of AI into 5G and future 6G networks, while promising, presents challenges, such as training models with the vast amounts of data generated by these networks. However, by leveraging this data, AI can detect patterns and trends that enable proactive resource allocation, load balancing, or energy-saving measures as some application examples.

Despite the clear potential of these technologies, the adoption of AI and ML methods in mobile networks remains in its early stages. The numerous challenges that lie ahead, coupled with the certainty that this technology will be transformative, have driven the initiation of this project. The goal is to make a contribution to the ongoing academic and industrial efforts in this field. To achieve this, a DQN-based radio resource allocation agent has been developed, designed to optimize network parameters to meet specific service requirements within a 5G network slice.

Asignación optimizada de recursos radio en 5G utilizando Deep Q-Networks (DQN)

David Fernández Martínez

Palabras clave: 5G, Red de acceso radio, Bloque de recursos físico, Aprendizaje máquina, redes neuronales, Aprendizaje por refuerzo profundo, Deep Q-Networks, ϵ-greedy,...

Resumen

La transición de 4G a 5G no ha estado marcada únicamente por un crecimiento en el tráfico, como venía siendo habitual en la última década, si no que ha venido acompañada de un aumento en el número de dispositivos y servicios. Servicios como Massive Internet of Things (mIoT), Massive Machine Type Communication (mMTC), ultra-Reliable Low Latency Communication (uRLLC), and Enhanced Mobile Broadband (eMBB) que requieren la implementación de network slicing aumentan significativamente la complejidad de la red.

Para responder a los nuevos servicios y transicionar hacia las redes del futuro de forma satisfactoria, los operadores se enfrentan al reto de gestionar y optimizar su infraestructura de red. La optimización de los recursos de la red nunca había sido un aspecto tan crítico como ahora. Los métodos tradicionales ya no son suficientes para gestionar la complejidad y la escala de las redes 5G. Ante esto el aprendizaje máquina y la inteligencia artificial se presentan como una prometedora tecnología capaz de mejorar la eficiencia de múltiples procesos, que tradicionalmente han sido llevados a cabo por humanos con un amplio conocimiento experto. La integración de la AI en las redes 5G y las futuras 6G, si bien es prometedora, presenta desafíos, como el entrenamiento de modelos con la gran cantidad de datos generados por estas redes. Sin embargo, al aprovechar estos datos, la AI puede detectar patrones y tendencias que permitan la asignación proactiva de recursos, el equilibrio de carga o medidas de ahorro de energía como algunos ejemplos de aplicación.

A pesar del claro potencial de estas tecnologías, la adopción de métodos de AI y ML en redes móviles aún se encuentra en sus primeras etapas. Los numerosos desafíos que se avecinan, junto con la certeza de que esta tecnología será transformadora, han impulsado el inicio de este proyecto. El objetivo es realizar una contribución a los esfuerzos académicos e industriales en este campo. Para lograrlo, se ha desarrollado un agente de asignación de recursos de radio basado en DQN, diseñado para optimizar los parámetros

de red cumpliendo con los requisitos de servicio específicos dentro de una slice 5G.

Pablo Ameigeiras Gutierrez, Catedrático de Universidad del Departamento de Teoría de la Señal, Telemática y Comunicaciones de la Universidad de Granada (Director del trabajo).

Lorena Chinchilla Romero, estudiante de doctorado (Mentora del trabajo).

Informan:

Que el presente trabajo, titulado **Asignación optimizada de recursos radio en 5G utilizando Deep Q-Networks (DQN)**, ha sido realizado bajo su supervision por David Fernández Martínez, y autoriza la defensa de dicho trabajo ante el tribunal que corresponda.

Y para que conste, expide y firma el presente informe en Granada, a 5 de Septiembre de 2024.

Los directores:

Pablo Ameigeiras Gutierrez **Lorena Chinchilla Romero**

Contents

List of Figures

List of Tables

Acronyms

1G First Generation

2G Second Generation

2D-CNN Two-dimensional convolutional neural networks

3G Third Generation

3GPP Third Generation Partnership Project

4G Fourth Generation

5G Fifth Generation

6G Sixth Generation

A2C Advantage Actor-Critic

AI Artificial Intelligence

CDF Cumulative Distribution Function

CNN Convolutional Neural Networks

CP-OFDM Cyclic Prefix OFDM

DFT-S-OFDM Discrete Fourier Transform Spread

DQN Deep Q-learning Network

DNN Deep Neural Networks

DRL Deep Reinforcement Learning

eMBB Enhanced Mobile Broadband

FDD Frequency Division Duplex

FFNN Feed-forward neural networks

gNB Next Generation Node B

GPU Graphic Processing Unit

GSM System for Mobile Communications

HSPA High Speed Packet Access

HetNets Heterogeneous networks

IoT Internet of Things

ITU International Telecommunications Union

LTE Long-term Evolution

KPI Key Performance Indicators

MDP Markov Decision Process

mIoT Massive Internet of Things

ML Machine Learning

MMS Multi-media Messaging Service

mMTC Massive Machine Type Communication

NFV Network Function virtualisation

NG-RAN New Generation Radio Access Network

NLOS Non-Line-of-Sight

NN Neural Networks

NR New Radio

OFDM Orthogonal Frequency Division Multiplexing

OFDMA Orthogonal Frequency Division Multiple Access

OTT Over the Top

PAPR Peak-to-Average Power Ratio

PDF Probability Density Function

PRB Physical Resource Block

PPO Proximal Policy Optimization

RAN Radio Acces Network

RL Reinforcement Learning

RNN Recurrent Neural Networks

RSS Received Signal Strength

SC Small Cell

SCN Small Cell Networks

SDN Software-defined networking

SINR Signal to Interference Noise Ratio

SMS Short Message Service

SOL Supplementary Downlink

SON Self-Organizing Networks

SUL Supplementary Uplink

TDD Time Division Duplex

TD Temporal Difference

UE User Equipment

UMTS Universal Mobile Telecommunication System

uRLLC ultra-Reliable Low Latency Communication

Chapter 1

Introduction

1.1 Context and motivation

Initial 4G network deployments focused on accommodating rapid traffic growth, a trend that has persisted over the past decade. In contrast, 5G has encountered a more complex and ambitious set of objectives. While traffic continues to rise, the number of connected devices is also increasing significantly. The proliferation of these new devices results from the diverse services that 5G aims to support, including Massive Internet of Things (mIoT)), Massive Machine Type Communication (mMTC), ultra-Reliable Low Latency Communication (uRLLC), and Enhanced Mobile Broadband (eMBB). This will be accomplished by simultaneously supporting multiple distinct logical networks, known as network slices, which will operate on the same physical infrastructure. Each network slice will be tailored to meet the specific requirements of different vertical industries.

Facing these new services presents various challenges, with one of the most significant being the optimization of network resources. Given the rapid evolution of mobile network technologies, traditional optimization methods are increasingly inadequate to manage the complexity and scale of modern networks. The introduction of network slicing, coupled with the diverse and stringent performance requirements of 5G and beyond, necessitates a more dynamic and intelligent approach to network management. This is where Artificial Intelligence (AI) and Machine Learning (ML) come into play as transformative technologies. AI-driven solutions have the potential to automate the optimization process, enabling networks to adapt in real-time to changing conditions, user demands, and service requirements. This adaptability is crucial for maintaining the quality of service (QoS) and ensuring efficient use of network resources across various slices.

The incorporation of AI into 5G and 6G networks offers significant

promise, but it also introduces various challenges. A key issue is the complexity involved in training AI models with the vast amounts of data produced by these networks. By analysing large quantities of network data, these algorithms can detect patterns and trends, which can be used to enable proactive resource allocation, load balancing, interference reduction, and energy-saving measures, among other applications. Furthermore, AI-powered predictive maintenance and self-healing capabilities can greatly minimize network downtimes and related costs, leading to improved operational efficiency.

Self-Organizing Networks (SON) have become increasingly critical. Traditional SON relied on automation algorithms designed by humans to optimize specific network parameters, aiming to minimize human intervention. While this approach reduced manual involvement, it still fell short of achieving full automation. SON now integrates machine learning algorithms to enable self-configuration, self-optimization, and self-healing within the network. With ML-driven self-configuration, radio access networks can automatically adjust parameters and settings based on real-time network conditions and user demands, significantly reducing the need for manual intervention. Self-optimization uses ML to continuously monitor and fine-tune network performance, optimizing resource use and adapting to evolving conditions.

Despite the clear potential of these technologies, the adoption of AI and ML methods in mobile networks remains in its early stages. The numerous challenges that lie ahead, coupled with the certainty that this technology will be transformative, have driven the initiation of this project. The goal is to make a contribution to the ongoing academic and industrial efforts in this field.

1.2 Scope and objectives

As discussed in the previous section, this thesis focuses on studying optimization in mobile networks using emerging AI and machine learning technologies. Specifically, the research focuses on the Radio Acces Network (RAN), optimizing resource allocation. A radio resource allocation agent utilizing Deep Reinforcement Learning (DRL) will be developed. This approach integrates reinforcement learning, a machine learning technique, with deep learning through neural networks.

The objective of this thesis is to study the behaviour of a radio resource allocation agent within a scenario that simulates a real Radio Acces Network (RAN). The aim of this study is to draw general conclusions that

will enhance our understanding of how to address these types of problems. Specifically, it seeks to identify factors that may be detrimental to agent performance and highlight design elements that contribute to optimal results. In other words, the use case developed in this project aims to serve as a practical example of applying DRL to radio resource allocation.

Also, the following specific objectives can be highlighted:

- Examine the theoretical foundation of DRL, with a particular focus on Deep Q-learning Network (DQN), to understand the key components and challenges associated with implementing this algorithm.

- Familiarize yourself with the unified interface of the Gym library for defining DRL environments, enabling the development of a customized environment tailored to the project's use case that adheres to this interface.

- Propose a realistic use case where the optimization carried out by the radio resource allocation agent is directly aligned with the operational interests of a real-world network operator.

- Develop a RAN simulator in Python that meets the requirements of the use case and is compatible with a Gym environment for effective communication.

- Adapt the actions that the agent can perform in the environment to the interface used by Gym. Optimizing this coding to favour agent learning.

- Construct a state space with representative variables that minimize dimensionality. This approach will enhance agent learning while reducing computational costs.

- Develop a reward algorithm that aligns with the optimization objectives, effectively guiding the agent toward achieving optimal solutions.

- Investigate the parameters influencing agent training and determine the optimal values for each.

- Conduct a detailed analysis of the agent's behaviour during the inference phase, using various metrics to assess its performance. Include an in-depth examination of the agent's interactions with the environment at each step.

1.3 Project planning and cost estimation

This section outlines the project timeline, detailing the sequence of tasks, the estimated time required for each, and the resources needed to complete them and reach the project's objectives. It is organized into three distinct subsections. The first subsection covers task planning, including descriptions and time estimates for each task. The second subsection focuses on the hardware, software, and human resources essential for project execution. Lastly, the third subsection presents the project budget.

1.3.1 Task planning

The tasks undertaken to complete the thesis are listed below, along with a brief description of them.

1. **Introduction to reinforcement learning**. During the first month, the mathematical foundations of reinforcement learning were thoroughly studied. Additionally, a practical project focused on developing a Q-learning agent was undertaken. This work laid the groundwork by establishing the essential concepts needed to begin working with DRL and DQN.

2. **Introduction to Gym and stable_baselines3**. The Python libraries Gym and stable_baselines3 are the core libraries in the developed solution. Gym provides the RL framework, while stable_baselines3 offers the implementation of the DQN algorithm. This task focuses on gaining familiarity and proficiency with these libraries.

3. **Comprehensive study of DQN**. The objective of this task is to thoroughly examine the limitations, challenges, and implementation aspects of the DQN algorithm. The primary bibliography for this study includes references [9] and [11].

4. **DRL optimization review in radio access networks**. The application of DRL optimization in RAN encompasses a wide range of use cases. This task aims to evaluate the current state of the art and explore various approaches to identify the potential limitations and challenges associated with the project.

5. **RAN simulator in Python**. Define the network model and implement it in Python as a radio access simulator.

6. **Use case definition**. Once the network model is established, the optimization problem is defined. This encompasses the mathematical formulation of the optimization conditions and the identification of the relevant network parameters involved.

7. **Design action space**. Model the actions the agent can perform on the network to achieve the optimization objective.

8. **Design state space**. Model the variables that make up the state space to find the best configurations.

9. **Design reward algorithm**. Develop various strategies within the reward algorithm to guide the agent toward the most optimal solution possible.

10. **Train and evaluate agents**. This task integrates the previous objectives of modelling the agent's actions and developing strategies within the reward algorithm. It involves configuring the state space and reward algorithm, followed by training the agent and evaluating its behavior. This iterative process allows for continuous refinement, where conclusions are drawn, and the design is adjusted to enhance its effectiveness.

11. **Hyperparameter configuration**. After finalizing the design, the impact of the training configuration parameters is analysed. The goal of this task is to identify the optimal agent within the established design framework.

12. **Inference phase**. Analysis of the agent's performance metrics in the environment across various scenarios.

13. **Experimental evaluation**. Analyse the results and draw conclusions by visualizing the agent's step-by-step operations.

14. **Document drafting**. Lastly, this task is the only one that spans the entire duration of the project. In this document, all the information related with this project is collected.

Then, in Table 1.1 the time invested on each task of the project described previously is included. In Figure 1.1, the Gantt diagram is represented. It shows the project tasks development along time.

1.3.2 Resource planning

In this section, resource planning is carried out. For the development of this project, the necessary resources can be classified into three different types: physical resources or hardware, software resources and human resources.

1. **Hardware resources**.

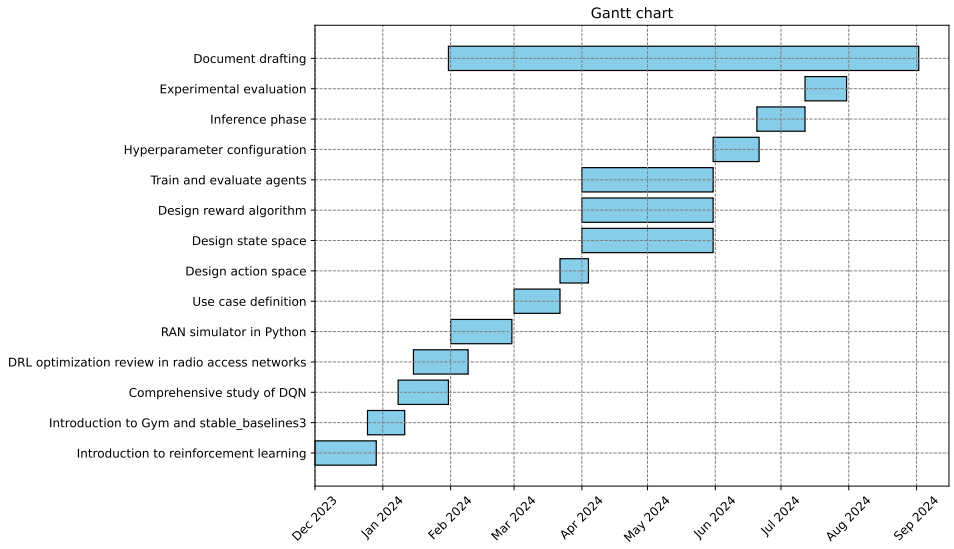


Fig. 1.1: Gantt chart

Task	Time (h)
Introduction to reinforcement learning	40
Introduction to Gym and stable_baselines3	30
Comprehensive study of DQN	60
DRL optimization review in radio access networks	25
RAN simulator in Python	60
Use case definition	40
Design action space	25
Design state space	80
Design reward algorithm	80
Train and evaluate agents	80
Hyperparameter configuration	60
Inference phase	60
Experimental evaluation	40
Document drafting	360
Total Time	1040

Table 1.1: Project temporal planning.

For hardware resources, a Lenovo IdeaPad 330 laptop was utilized for software development and document preparation. Additionally, a computer equipped with a Graphic Processing Unit (GPU) was utilized for training and inference phases. The GPU significantly enhances computational efficiency, meeting the intensive demands of these processes.

The technical characteristics of the hardware components are shown below.

- **Lenovo ideapad 330 technical characteristics**. Intel Core i7-8550U processor, 256 GB SATA3 SSD storage, 256 GB SATA3 SSD storage, RAM memory of 8 GB and UHD Graphics 620 Intel graphics card.
- **HP Z4 G4 Workstation**. Intel Core i9-10920X processor and NVIDIA GeForce RTX 3090.

2. **Software resources**.

Software resources encompass all the programs and applications utilized at any stage of the project's development. In Table 1.2 all of them are shown:

Software resource
SO Windows 10 64 bit
Overleaf online LaTeX editor
IEEE Xplore
Lucidchart
Python 3.8
PyCharm IDE

Table 1.2: Software resources

3. **Human resources**.
Human resources are quantified by the number of people involved in the project and the time they dedicate to it. This includes the working hours of both students and supervisors, which are accounted for in this section. Table 1.3 presents an estimate of the time invested by the student and supervisors. It is worth mentioning that the time

Person	**Invested time (h)**
Student	1040
Supervisor 1	40
Supervisor 2	40

Table 1.3: Human resources

spent by the student is the time that has been calculated in the Table 1.1 of tasks planning subsection. The time invested by the supervisors encompasses not only the tutoring hours but also the time spent sourcing materials, such as bibliographic references, as well as the time dedicated to reviewing and evaluating the student's work.

1.3.3 Project Budget

In this subsection, the total cost of the project is estimated, assuming that none of the required resources were available before the start of the project.

Regarding software resources, only non-open-source programs and applications will be accounted for as expenditures, as they require a license. The price of the student working hours, considering that he is a newly telecommunications engineer, is set at 25 euros; mean while the price of the supervisors working hours is set at 50 euros. The detailed pricing breakdown for each item and the total project cost are presented in Table 1.4.

Resource	**Units**	**Unit Cost (€)**	**Subtotal Cost (€)**
Laptop	1	550.00	550.00
Computer with GPU	1	2329.99	2329.99
Windows 10 Home OS	1	145.00	145.00
IEE Xplore monthly subscription	5 months	40.56	202.8
Supervisor 1 labor	40 hours	50.00	2000.00
Supervisor 2 labor	40 hours	50.00	2000.00
Student labor	1040 hours	25.00	26000.00
TOTAL(€)			33227.79

Table 1.4: Project total cost.

Thus, as it can be seen in Table 1.4, the total project expenditure is 33227.79 €.

1.4 Project Structure

This section provides an overview of the structure and contents of this Master's Thesis, offering the reader a clear understanding of its organization. The dissertation is composed of six chapters and four appendices, with a brief summary of each chapter presented below to guide the reader through the document.

- **Chapter 1: Introduction**. Sets the stage for the project by providing an overview of the emerging possibilities enabled by the integration of AI and ML in mobile networks. It outlines the motivation and objectives of the work, offering the reader a clear understanding of its purpose. Additionally, this chapter includes a section on project planning, detailing the tasks required for the project's execution over time.

The resource planning and the estimated total cost of this project are also included in the mentioned section.

- **Chapter 2: State of the Art**. Describe all the key enablers on which this project is based. The chapter is clearly divided into three main sections. The first section offers a general overview of Fifth Generation (5G), focusing on key radio access concepts such as spectrum and overall transmission structure. It concludes with a discussion on Network Slicing technology. The second section reviews the Self-Organizing Networks (SON) paradigm and its operation within 5G networks. This review includes an examination of the most popular AI/ML optimization methods applied to SON networks. Finally, the third section presents the theoretical foundations of Deep Reinforcement Learning (DRL), with a detailed exploration of Deep Q-learning Network (DQN) and the implementation keys of this algorithm.

- **Chapter 3: System model and problem definition**. This chapter describes the system model adopted for this project, describ- ing the network design. It concludes by defining the problem statement addressed within this system.

- **Chapter 4: Solution Design**. It presents the solution developed to address the use case outlined in this Master's Thesis. First, the solution is outlined and introduced at a high level, followed by a detailed explanation of each of its components (action space, state space, reward algorithm). Lastly, the implementation of the solution using the Gym reinforcement learning framework and the interaction with the DQN agent are described.

- **Chapter 5: Experimental evaluation and results**. This chapter analyses the results obtained for the DQN agent developed in the project. The experimental setup in which the agent was tested is detailed, including two primary experiments. The first experiment examines the impact of various parameters on the agent's training process. The second experiment provides a step-by-step demonstration of the agent's performance under various conditions, highlighting the effects of its actions on network parameters.

- **Chapter 6: Conclusions and future works**. At the end of this document, the conclusions drawn from the development of this work are presented. Additionally, this chapter offers insights into potential future research directions on the topic.

- **Appendix A: UEs assignment to serving SCs**. This appendix outlines the procedure used in the proposed system model to assign cells to UEs. For this purpose, the implemented algorithm is presented.

- **Appendix B: RAN simulator.** It describes how the network model is implemented as a RAN simulator in Python. It provides a detailed explanation of the developed pseudocode.
- **Appendix C: Advantages of discretizing throughput as a state variable**. The content of this appendix relates to Chapter 4, demonstrating why the design choices made for the state space in that chapter are optimal for facilitating the agent's learning.
- **Appendix D: Code repository**. This appendix details the structure of the code developed for this project, which is available in GitHub.

Chapter 2

State of the Art

This chapter presents the essential concepts and technologies necessary to understand the solution developed and implemented in this thesis.

First, an overview of Fifth Generation (5G) and the main radio access concepts, such as spectrum and overall transmission structure, is provided. This section concludes with a discussion on Network Slicing technology, a key component introduced in 5G and a critical aspect of the solution designed in this project.

The second section reviews the Self-Organizing Networks (SON) paradigm and its operation within 5G networks. This review includes an examination of the most popular AI/ML optimization methods applied to SON networks.

Finally, the theoretical basis of Deep Reinforcement Learning (DRL) is presented. The basic concepts of Reinforcement Learning (RL) are introduced, along with the mathematical formalism that leads its behaviour, specifically Markov Decision Process (MDP). Building on these foundations, the chapter explains how Deep Learning is applied to RL, resulting in DRL. The theoretical foundation of the algorithm used in this project, Deep Q-learning Network (DQN), is discussed in the final section

2.1 Fifth Generation of mobile networks (5G)

Since the first generation of mobile communications was introduced in the early 1980s, a new generation has appeared every 10 years. The First Generation (1G) systems used analogue communications techniques and did not use the available radio spectrum efficiently. The Second Generation (2G) turned mobile telecommunications into a consumer product due to the decreasing price of the devices, the most popular 2G system was System for Mobile Communications (GSM). It was originally designed for voice, but

later allowed the sending of short messages known as Short Message Service (SMS). Due to the success of 2G, the International Telecommunications Union (ITU) specified a number of requirements that led to a Third Generation (3G). The most popular third generation system was Universal Mobile Telecommunication System (UMTS) which had a more efficient use of spectrum than GSM. The most important further improvement in 3G is known as High Speed Packet Access (HSPA), which improves the performance of data applications by increasing the average transmission speed. The emergence of HSPA coincided with the first smartphones that would change the paradigm up to that point due to the massive growth of mobile data traffic. As a result, Fourth Generation (4G) systems are optimized to meet these new challenges. By far the most popular is the 3GPP System known as Long-term Evolution (LTE). Over the last decade, the fourth generation has incorporated improvements that have allowed it to respond to traditional mobile broadband. This brings us to the Fifth Generation (5G), known as New Radio (NR).

5G radio will take the traditional mobile broadband to the extreme in terms of data rates, capacity, and availability. In addition, 5G will enable new services including industrial Internet of Things (IoT) connectivity and critical communication. 4G networks were designed for the use case of smartphones, so far mobile networks have primarily focused on connecting people. A number of new use cases and applications can be run on top of 5G mobile networks. It is expected that 5G can fundamentally impact all sections of society by improving efficiency, productivity, and safety through advancements such as remote surgery, Industry 4.0, smart cities, autonomous vehicles, and augmented reality [4].

In September 2015 the ITU published ITU-R M.2083 "IMT Vision – Framework and overall objectives of the future development of IMT for 2020 and beyond", in which it identified key capabilities of IMT-2020 and outlined three main scenarios of use:

- **Enhanced Mobile Broadband (eMBB).** Human-centric use cases for accessing multimedia content, services and data. But with improved performance and an increasingly seamless user experience compared to existing mobile broadband applications. For example, by supporting even higher end-user data rates.

- **Massive Machine Type Communication (mMTC).** Communication services in which devices communicate with each other without human intervention. Key requirements for such services entail minimal device expenses and energy consumption, facilitating extended battery life of several years at the least. Typically, each device handles and produces only a modest amount of data, making support for high data rates less crucial.

- **ultra-Reliable Low Latency Communication (uRLLC).** Communication services with strict requirements in terms of latency and reliability. In addition, these services cover both human-centric and machine-centric communications. Examples hereof are traffic safety, automatic control, and factory automation

5G radio can bring major benefits in terms of network performance and efficiency compared to LTE radio. Minimum technical requirements for 5G are shown in Figure 2.1, the values are given next to the most relevant use case. There is no need to meet all requirements simultaneously, whilst all key ca-

KPI	Key Use Case	Values
Peak Data Rate	eMBB	DL: 20 Gbps, UL: 10 Gbps
Peak Spectral Efficiency	eMBB	DL: 30 bps/Hz, UL: 15 bps/Hz
User Experienced Data Rate	eMBB	DL: 100 Mbps, UL: 50 Mbps (Dense Urban)
5% User Spectral Efficiency	eMBB	DL: 0.3 bps/Hz, UL: 0.21 bps/Hz (Indoor Hotspot); DL: 0.225 bps/Hz, UL: 0.15 bps/Hz (Dense Urban); DL: 0.12 bps/Hz, UL: 0.045 bps/Hz (Rural)
Average Spectral Efficiency	eMBB	DL: 9 bps/Hz/TRxP, UL: 6.75 bps/Hz/TRxP (Indoor Hotspot); DL: 7.8 bps/Hz/TRxP, UL: 5.4 bps/Hz/TRxP (Dense Urban); DL: 3.3 bps/Hz/TRxP, UL: 1.6 bps/Hz/TRxP (Rural)
Area Traffic Capacity	eMBB	DL: 10 Mbps/m^2 (Indoor Hotspot)
User Plane Latency	eMBB, URLLC	4 ms for eMBB and 1 ms for URLLC
Control Plane Latency	eMBB, URLLC	20 ms for eMBB and URLLC
Connection Density	mMTC	1,000,000 devices/km^2
Energy Efficiency	eMBB	Capability to support high sleep ratio and long sleep duration to enable low energy consumption when there is no data
Reliability	URLLC	$1–10^{-5}$ success probability of transmitting a layer 2 protocol data unit of 32 bytes within 1 ms in channel quality of coverage edge
Mobility	eMBB	Up to 500 km/h
Mobility Interruption Time	eMBB, URLLC	0 ms
Bandwidth	eMBB	At least 100 MHz; Up to 1 GHz for operation in higher frequency bands (e.g., above 6 GHz)

Fig. 2.1: Minimum technical performance requirements of IMT 2020 [1]

pabilities may to some extent be important for most use cases, the relevance of certain key capabilities may be significantly different, depending on the use cases/scenario. The importance of each requirement in the three use case scenarios is shown in Figure 2.2. The eMBB scenario is the one with the most stringent requirements: user experienced data rate, area traffic capacity, peak data rate, mobility, energy efficiency and spectrum efficiency. However, it does not need such extreme values in connection density and latency, which are the key requirements of the other use cases. For example, in uRLLC scenarios low latency is very important, so that safety critical applications can operate. In the context of mMTC, a high connection density is essential to accommodate a vast number of devices in the network. These devices might transmit only sporadically, at low bit rates, and with very low mobility.

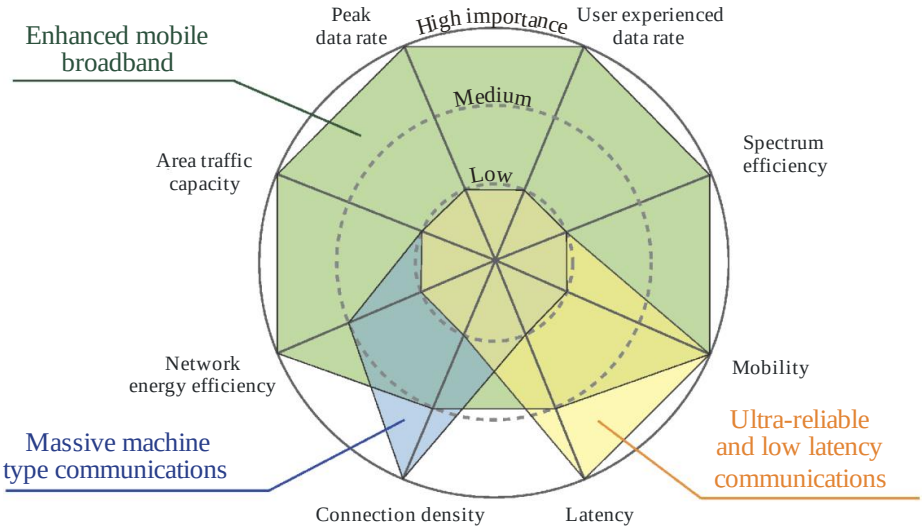


Fig. 2.2: Key capabilities in different usage scenarios [2]

2.1.1 Spectrum

Spectrum is the most important resource for operators. The available spectrum defines both the capacity and the range of coverage that the mobile network can offer. Congestion in the low bands, combined with higher bandwidth requirements, has led to the use of both low and high bands in 5G. Figure 2.3 shows the 5G target frequency range together with some characteristics of the bands. The high bands at millimeter wave have lot of

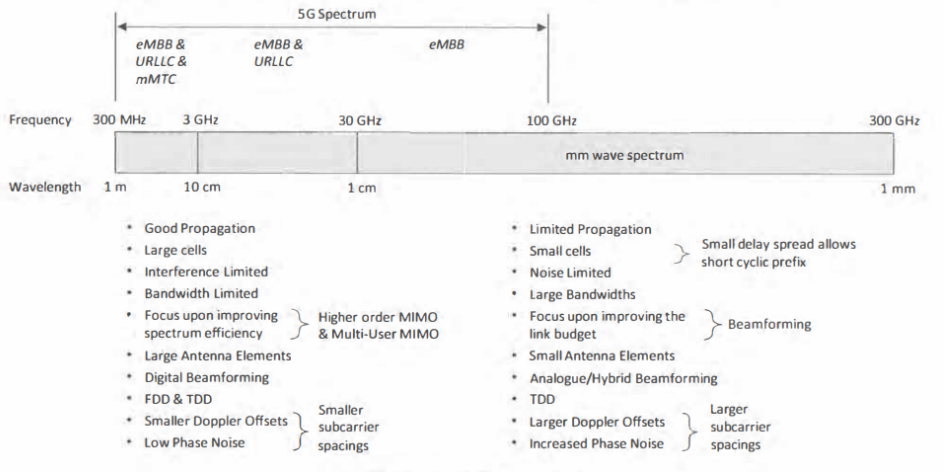


Fig. 2.3: Range of 5G Spectrum [3]

spectrum and enable high capacity and data rates. The low bands have

great propagation and provide wide area coverage. The challenge with high spectrum is the short propagation. Millimeter wave signals attenuate fast, and the cell range is limited to a few hundred meters. Third Generation Partnership Project (3GPP) has specified 2 frequency ranges:

- Frequency Range 1 (450MHz - 6GHz).
- Frequency Range 2 (24.25GHz - 52.60GHz).

This division allows a different set of requirements to be specified for each frequency range. Bands from frequency range 1 are shown in Figure 2.4. These bands allow Frequency Division Duplex (FDD), Time Division Duplex (TDD), Supplementary Downlink (SOL) and Supplementary Uplink (SUL). Bands n1 to n76 are re-farming bands, i.e. they have been specified for 4G but can be reused for 5G. The highest bands are the new 5G TDD bands (n77, n78 and n79). These bands have the ability to provide high throughput at relatively low frequencies.

Figure 2.5 shows the bands of the frequency range 2. This part of the spectrum is essential in 5G in order to achieve increased capacity and data rates according to requirements. Bandwidths are much higher than FR1, these bands only allow TDD. Each operating band is specified to support a specific set of channel bandwidths that depend on the subcarrier spacing. FR1 allows 15, 30 and 60kHz subcarrier spacing, up to 100MHz total bandwidth can be achieved with 30kHz and 60kHz. In FR2, a bandwidth of 400MHz can be achieved with 120kHz subcarrier spacing.

Figure 2.6 shows a typical spectrum usage in 5G. The main spectrum below 6 GHz worldwide will be 3.5 GHz, covering up to 400 MHz from 3.4 to 3.8 GHz with possible further extension up to 4.2 GHz. The spectrum around 3.5 GHz is attractive for 5G because it is available worldwide and the amount of spectrum is relatively high. 5G still requires low bands, below 1GHz, to ensure wide coverage and indoor penetration. Extensive coverage is important for the new use cases like Internet of Things (IoT) and critical communication. Finally, millimeter waves can provide high data rates, this is particularly interesting for fixed wireless users.

2.1.2 Overall Transmission Structure

This section describes the overall transmission structure in NR, providing a complete overview of its key components. Fundamental aspects of waveform, numerology, time domain structure, and frequency domain structure are presented.

Operating Band	Duplex Mode	Uplink Band (MHz)	Downlink Band (MHz)	Bandwidth (MHz)	Category
n1	FDD	1920 - 1980	2110 - 2170	2 × 60	Re-farmed 4G bands
n2		1850 - 1910	1930 - 1990	2 × 60	
n3		1710 - 1785	1805 - 1880	2 × 75	
n5		824 - 849	869 - 894	2 × 25	
n7		2500 - 2570	2620 - 2690	2 × 70	
n8		880 - 915	925 - 960	2 × 35	
n12		699 - 716	729 - 746	2 × 17	
n20		832 - 862	791 - 821	2 × 30	
n25		1850 - 1915	1930 - 1995	2 × 65	
n28		703 - 748	758 - 803	2 × 45	
n34	TDD	2010 - 2025		15	
n38		2570 - 2620		50	
n39		1880 - 1920		40	
n40		2300 - 2400		100	
n41		2496 - 2690		194	
n50		1432 - 1517		85	
n51		1427 - 1432		5	
n65	FDD	1920 - 2010	2110 2200	2 × 90	
n66		1710 - 1780	2110 - 2200	70 + 90	
n70		1695 - 1710	1995 - 2020	15 + 25	
n71		663 - 698	617 - 652	2 × 35	
n74		1427 - 1470	1475 - 1518	2 × 43	
n75	SDL	Not Applicable	1432 - 1517	85	
n76			1427 - 1432	5	
n77	TDD	3300 - 4200		900	New 5G bands
n78		3300 - 3800		500	
n79		4400 - 5000		600	
n80	SUL	1710 - 1785	Not Applicable	75	
n81		880 - 915		35	
n82		832 - 862		30	
n83		703 - 748		45	
n84		1920 - 1980		60	
n86		1710 - 1780		70	

Fig. 2.4: 5G operating bands for Frequency Range I (450 MHz to 6 GHz) [3]

Waveform

3GPP has defined two possible waveforms in, NR based on Orthogonal Frequency Division Multiplexing (OFDM):

- Cyclic Prefix OFDM (CP-OFDM). Both uplink and downlink.
- Discrete Fourier Transform Spread (DFT-S-OFDM). Only in the uplink.

CP-OFDM is used as the basic scheme, but the possibility of implementing DFT-S-OFDM in the uplink is allowed. The reason for this is the same as

Operating Band	Duplex Mode	Uplink Band (MHz)	Downlink Band (MHz)	Bandwidth (MHz)	Category
n257	TDD	26 500 - 29 500		3 000	New 5G bands
n258		24 250 - 27 500		3 250	
n260		37 000 - 40 000		3 000	
n261		27 500 – 28 350		850	

Fig. 2.5: 5G operating bands for Frequency Range 2 (24.25 to 52.60 GHz) [3]

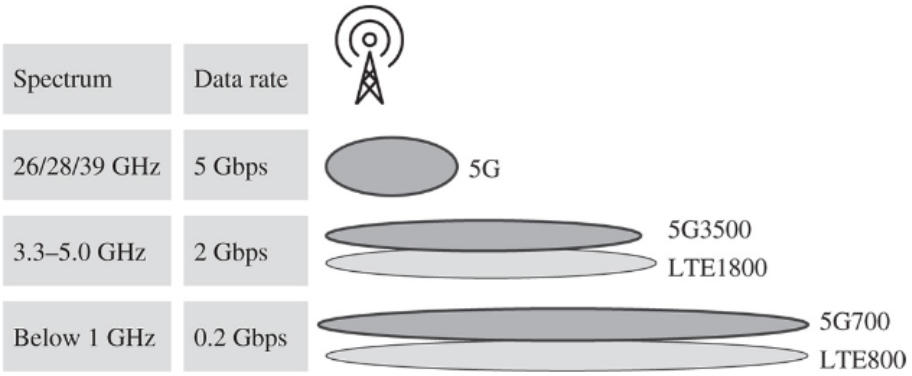


Fig. 2.6: Example of early phase 5G spectrum usage and coverage. [4]

in LTE, to obtain a higher power amplifier efficiency. DFT-S-OFDM has a lower Peak-to-Average Power Ratio (PAPR) compared to CP-OFDM. This allows the User Equipment (UE) to transmit with a higher average power, improving uplink coverage performance. However, it has some disadvantages, for example in MIMO. Spatial multiplexing receivers become more complex and 3GPP has specified that DFT-S-OFDM only supports single stream transmission [3]. When DFT-S-OFDM is used, uplink transmissions are restricted to a single layer only, while uplink transmissions of up to four layers are possible with CP-OFDM. In other words, DFT-S-OFDM provides better troughput and capacity, while CP-OFDM provides better coverage.

Numerology

The numerology defines the subcarrier spacing and the cyclic prefix length. In LTE a subcarrier spacing of 15KHz and a cyclic prefix of approximately $4.7\mu s$ has been used because it gives a good result for the scenario it was designed for. NR must have the flexibility to support different spectrum and service options. Having a single numerology value for all scenarios is not possible from a requirements point of view. LTE subcarrier spacing (15 kHz) was selected as the baseline for NR. From this value, larger subcarrier

spacing are obtained according to the following relationship:

$$\triangle f = 2^{\mu} \cdot 15KHz \tag{2.1}$$

Table 2.1 shows the subcarrier spacing, the symbol duration and type of cyclic prefix.

μ	$\triangle f$ (KHz)	Symbol duration (μs)	Cyclic prefix
0	15	66.67	Normal
1	30	33.33	Normal
2	60	16.67	Normal, Extended
3	120	8.33	Normal
4	240	4.17	Normal
5	480	2.08	Normal

Table 2.1: Subcarrier Spacings Supported by NR.

Time-Domain Structure

Figure 2.7 illustrates the time domain structure. NR transmissions are orga-

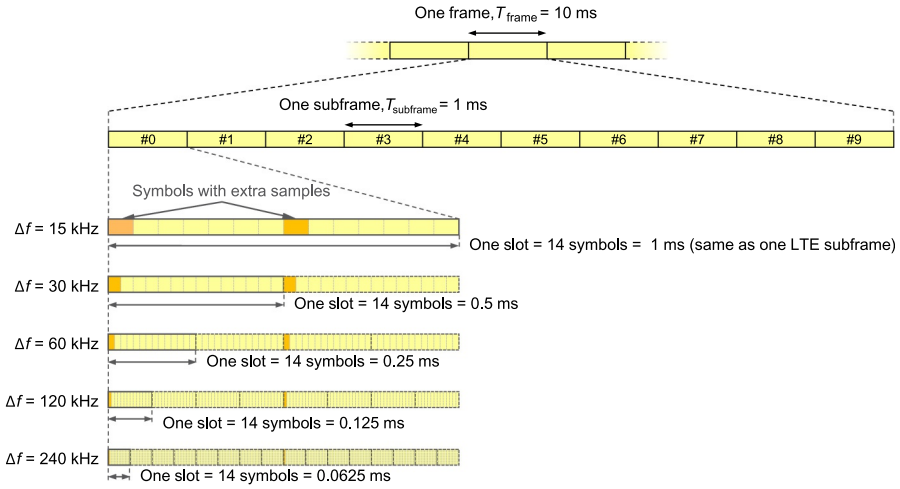


Fig. 2.7: Frames, subframes, and slots in NR. [5]

nized into frames of length 10ms, each of which is divided into 10 subframes of length 1ms. Each subframe is divided into slots of 14 OFDM symbols, the slot duration depends on the numerology. For the 15KHz subcarrier spacing, the subframe corresponds to a slot and has the same structure as a LTE subframe. The time-domain structure for higher subcarrier spacings in NR is then derived by scaling the baseline 15 kHz structure by powers of

two, the number of slots per subframe is 2^{μ}.

A higher subcarrier spacing implies a shorter slot duration. This can be beneficial for low delay transmissions, but it also reduces the cyclic prefix, so it is not a feasible approach in all scenarios.

Frequency-Domain Structure

Figure 2.8 illustrates the frequency domain structure. A resource element,

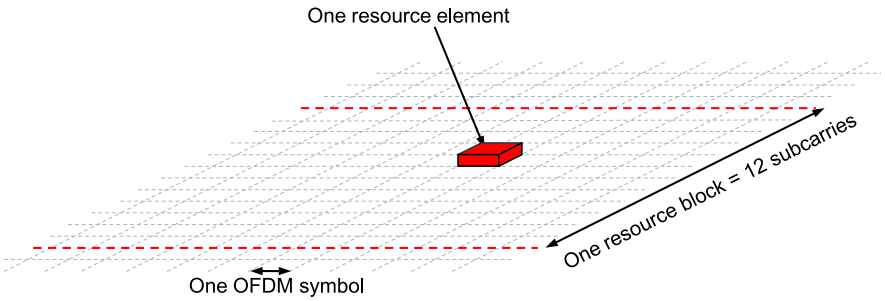


Fig. 2.8: Resource element and resource block. [5]

consisting of one subcarrier during one OFDM symbol, is the smallest physical resource in NR. 12 consecutive subcarriers in the frequency domain are called a resource block. Due to multiple numerology that NR supports, the bandwidth of a resource block is variable depending on said numerology. Two resource blocks at a subcarrier spacing $\triangle f$ occupy the same frequency range as one resource block at a subcarrier spacing of $2 \triangle f$. This concept is illustrated in Figure 2.9.

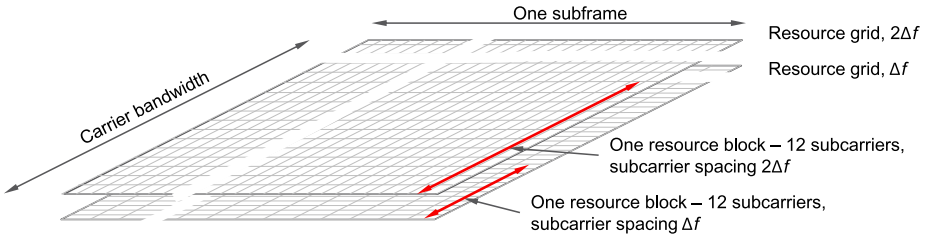


Fig. 2.9: Resource grids for two different subcarrier spacings. [5]

2.1.3 Network Slicing

The introduction to this section has presented the main use case scenarios that 5G should be able to address. In order to satisfy different (and

potentially conflicting) requirements of different services and/or customers in a cost-effective manner, operators must convert their networks into programmable multi-service platforms, adopting the infrastructure and functional sharing mechanisms commonly known as network slices. With network slicing, the operator's network can be logically divided into a set of programmable network partitions (i.e. network segments), each designed to satisfy a particular set of service requirements [12].

To achieve this, there are two fundamental technologies that make up the 5G network architecture: Software-defined networking (SDN) and Network Function virtualisation (NFV) [13].

SDN is a key component of the 5G networking architecture, designed to overcome the limitations imposed by the use of hardware. The primary objective of implementing SDN is to achieve fully automated network administration, enabling the administrator to efficiently manage the network through a centralized control plane. This simplifies network management and facilitates the introduction of changes to the network. NFV architecture is ideal for efficiently managing the lifecycle of network services and their constituent resources. A virtualized network function consists of one or more virtual machines, each running various software and processes, operating on servers, switches, storage devices, and cloud infrastructure.

Through these two technologies, network softwarization can provide the flexibility and modularity that is required to create multiple logical (virtual) networks on top of a common network. We define network slices as end-to-end (E2E) logical networks that operate on a shared underlying network, whether physical or virtual. These slices are mutually isolated, independently controlled and managed, and can be created on demand [6]. An illustrative example of this concept is shown in Figure 2.10.

2.2 Self-Organizing Networks (SON)

Optimization of the mobile network and consequently a good use of available resources has always been a major objective of operators. During GSM and UMTS traffic was dominated by SMS and Multi-media Messaging Service (MMS), being this kind of traffic relatively predictable. However, the emergence of smartphones led to an increase in mobile traffic that changed the paradigm of network optimization.

The rapid expansion of mobile broadband significantly strains wireless radio networks and the underlying transport infrastructure. Operators have

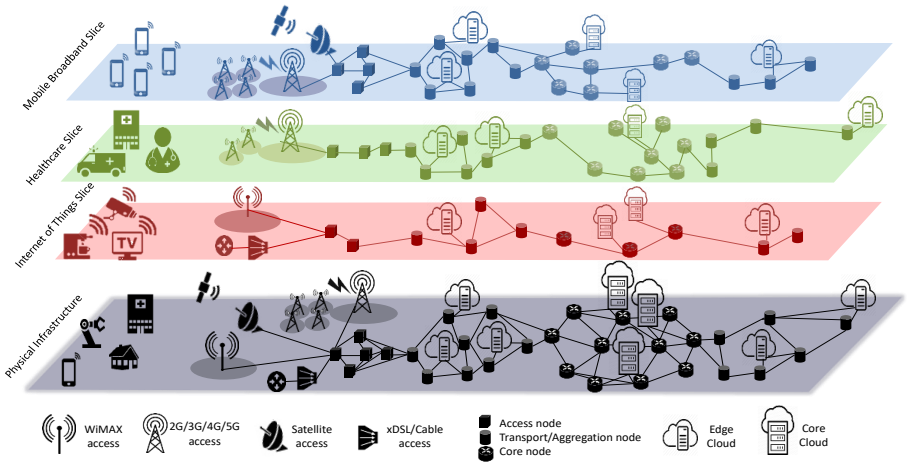


Fig. 2.10: 5G network slices running on a common underlying multi-vendor and multi-access network. Each slice is independently managed and addresses a particular use case. [6]

implemented various solutions to cope with this rapid traffic growth. First, operators can employ economic incentives to modify user behaviour by adjusting tariff structures. Another approach is to improve network capacity by deploying advanced radio access technologies, as was done for example with LTE deployments. Optimization of protocol design and traffic shaping methods is also developed, together with the deployment of advanced source coding techniques [14].

Many of these alternatives involve high capital outlays as the network infrastructure needs to be upgraded. It is therefore interesting to ask whether the available resources are being used to their full potential before considering network expansions or evolutions. This is where network optimisation and in particular the SON paradigm comes into play.

2.2.1 Motivation of the SON paradigm

Managing mobile networks is a complex task due to the number of network elements to be deployed and managed but also the interdependency of their configurations. Complexity increases when different 2G, 3G, 4G and 5G networks must coexist. In such Heterogeneous Network scenarios the variety of deployed technologies and their specific operational paradigms will be difficult to handle.

Traditionally, network management tasks are supervised by humans.

This manual effort by the human operator is time-consuming, expensive, error-prone and requires a high degree of expertise. Networks are operated so that the human operator re-plans network configuration based on analysed performance data [15]. The high complexity and cost of the manual procedure can be reduced with SON networks. The target is to gradually move towards a pure monitoring and steering of the SON-enabled system from manual planning and configuration. Therefore, the human's role is not to carry out frequent routine work anymore. Instead, the role is to design and decide policies that guide SON functioning [16]. The design of optimization algorithms in SON networks started out based on human expert guidelines, but in recent years the use of Machine Learning (ML) and Artificial Intelligence (AI) techniques has gained momentum. In the following sections we will discuss the different techniques that are being implemented.

2.2.2 SON for 5G Mobile Networks

SON originally appeared in 3GPP Release 8 [16] in response to the emergence of smartphones and the exponential growth of traffic, during the last decade, it has proven its usefulness in traditional macro networks. With the advent of 5G, the trend in traffic growth has not changed. The ITU has projected that the exponential increase in mobile data traffic will persist, anticipating that by 2030, the total monthly mobile data traffic will soar to an incredible 5 zettabytes (ZB), as shown in Figure 2.11. 5G is expected to

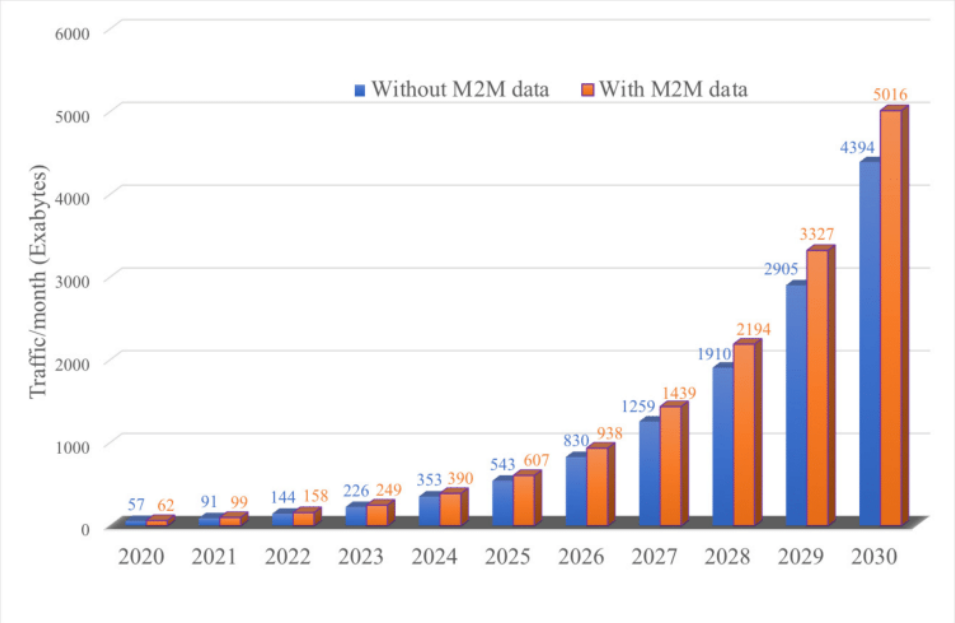


Fig. 2.11: Global mobile data traffic forecast by ITU. [6]

reach its limits in 2030 and the trend is expected to continue to grow for the Sixth Generation (6G).

In addition to this, the emergence of Small Cell Networks (SCN) also comes into play in 5G. This technology is based on the idea of deploying short-range, low-power, and low-cost base stations operating in conjunction with the main macrocellular network infrastructure [17]. We need many more small cells than macro cells, and small cells are deployed much more dynamically. In consequence, manual processes for configuration and optimization are no longer feasible, SCN have to be plug-and-play and self-configurable. In 5G, SON will continue playing an essential part, as Heterogeneous networks (HetNets) require even more automatic control. With the SON approach, the network can leverage knowledge of its current state to allocate resources efficiently, ensuring they are directed where and when they are needed most. As a result, users enjoy constant, seamless and virtually unlimited connectivity.

The SON use cases can be structured in different ways. one of the possible high-level classifications is the following (adopted in [14]):

- Self-Planning. Determining the settings for each new network node involves selecting the site location and specifying the hardware configuration, but does not include site acquisition and preparation.
- Self-Deployment. Preparation, installation, authentication and verification of every new network node. It includes all procedures to bring a new node into commercial operation.
- Self-Optimization. Utilization of measurements and performance indicators collected by the User Equipments and the base stations in order to auto-tune the network settings.
- Self-Healing. Execution of the routine actions that keep the network operational and/or prevent disruptive problems from arising. This includes the necessary Software (SW) and HW upgrades and/or replacements.

When there are several SON functionalities operating on the same network there may be conflict between their objectives so that the overall SON gain is reduced. In classical SON there is a latency, since the situation is first observed, diagnosed and finally compensatory action is taken. This reactive approach is not compatible with 5G quality of experience levels. Therefore, to be able to perform successfully in 5G network, the intended solution for SON paradigm should be proactive. In this way, the network can predict the potential problem beforehand instead of waiting to observe and spot the problem. Empowering SON with big data is the key to transforming SON

from being reactive to proactive [18].

In this sense, algorithms designed by humans to operate in SON networks are moving towards the use of Machine Learning (ML) and Artificial Intelligence (AI) to gain this predictive component and adapt to a proactive paradigm.

2.2.3 AI/ML applied to Self-Organizing Networks

The wide availability of configuration and performance data makes it possible to gather substantial information on the state of the network in real time, in this context AI becomes a key feature.

As discussed in the previous section, decisions that were previously made by experts manually or with optimization algorithms can now be made by ML algorithms. This not only improves the overall operational efficiency of the NR network infrastructure, but also has significant impact into the reduction of management and energy related costs. The use of Artificial Intelligence (AI) and Machine Learning (ML) as a key enabler for future networks has been recognized at European [19] and global level [20]. In the following subsections, two of the most popular AI/ML methods are reviewed in general terms: Neural Networks (NN) and Reinforcement Learning (RL) [8]. We focus on these two since the combination of both results in Deep Reinforcement Learning (DRL), the technique implemented in this project and whose theoretical basis is presented in section 2.3.

Neural Networks

Neural Networks (NN) emerged as an attempt to emulate the behaviour of the human brain in a computer. The equivalent component of a neuron in a NN is called a node. The nodes are connected to each other through links of varying weight that simulate the neural connections of the human brain. In addition, these nodes perform nonlinear operations using their activation functions [21].

The nodes are distributed in layers, there are 3 types of layers in a NN:

- **Input layer.** The data is received in this layer.
- **Hidden layer.** Layers through which the signal travels, undergoing changes depending on the weights.
- **Output layer.** Layer that returns the result of network processing.

Although all networks must have an input and an output layer, the number of hidden layers or the number of nodes is not fixed. A simple NN design

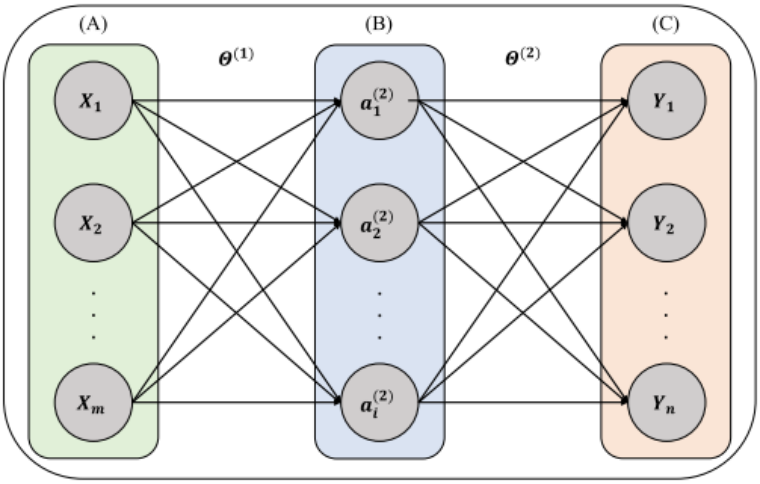


Fig. 2.12: Most basic design of a neural network, consisting of 3 layers [7]

of three layers is shown on Figure 2.12. A denotes the input layer, B the hidden layer and C the output layer. The variable link weights are depicted as $\theta^{(j)}$, which correspond to the matrix of weights controlling the function mapping between layer j to layer $j+1$ and the activation function of each neuron as ${a_i}^j$, where i is the neuron number and j is the layer number. As can be seen in the connections between layers, they all go forward and no loops are formed. This type of network is known as Feed-forward neural networks (FFNN).

The objective of an FFNN is to approximate an unknown function $y = f^*(x)$. During training, the network tries to adjust parameters θ. So the neural network defines a function $f(x, \theta)$, which depends on the inputs x and the parameters θ, with the objective of adjusting the values of θ in such a way that $f(x, \theta) = f^*(x)$.

The process of training Feedforward Neural Networks (FFNNs) follows a similar pattern to other machine learning algorithms. This involves computing a loss function, $J(\theta)$, based on the model's parameters (weights), and then using gradient descent to adjust these parameters in order to minimize the loss function. There are several types of loss functions, but the most utilized are the mean square error, the Huber Loss, Cross-entropy, the Kullback-Liebler Divergence, etc [22]. This loss function is used to update the parameters. This update starts from the last layer and propagates backwards, hence it is known as backpropagation.

FFNNs can perform automatic feature extraction across layers of different depths. In mobile networks the amount of data is massive, FFNNs

benefit from large amounts of data as this allows them to improve generalization.

In addition to FFNN, there are multiple other types of NNs. When the number of hidden layers is more than one, it is called Deep Neural Networks (DNN) [23]. Figure 2.13 shows a simple example of a DNN with 2 hidden layers. The use of DNN in mobile network optimization has grown in

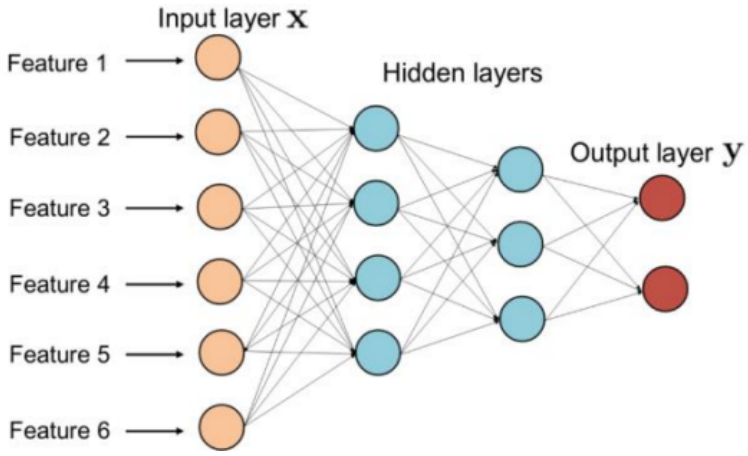


Fig. 2.13: Scheme of a deep neural network with 2 hidden layers [8]

recent years. It covers areas such as automatic computation offloading and edge caching [24], or also applications at the physical layer [25] for novel tasks such as spectrum analysis, radio virtualization and optimization, blockage prediction or beam alignment. DNNs are capable of adapting to multiple problems as long as a large amount of data and metrics can be collected, as is the case in the vast majority of mobile network optimization cases.

The most common architecture for DNNs is feed-forward, in which the information flows from the input layer to the output layer without internal loops. There are other types of architectures, such as Recurrent Neural Networks (RNN), where information is propagated both forward and backward. This change allows the output to depend on both the current input and its history through the network. This makes RNN capable of capturing temporal correlations such as those found, for example, in user mobility prediction.

Another type of architecture within DNN are Convolutional Neural Networks (CNN) which specialize in inferring local patterns in the feature space of a matrix input. Two-dimensional convolutional neural networks (2D-CNN) have been widely used in image processing. When the data is of a spatio-temporal nature, the architecture is extended to 3D-CNN, which adds the temporal dimension to the problem. Mobile network traffic exhibits patterns in both space and time, e.g. the geographical position of a base

station and traffic trends throughout the day. In this case, 3D-CNN can be used to infer patterns from a time series matrix of traffic with a spatial component.

Reinforcement Learning

Reinforcement Learning (RL) is a learning method in which an agent takes actions in an environment and receives feedback from it, so that the learning technique is based on trial-and-error. Unlike many other forms of machine learning, the agent is not told what actions to take, but discovers the best actions by trying them out. The agent's goal is to maximize his reward, in other words, given an initial situation it must find a set of actions for which the environment provides him with the maximum reward. This type of solution has drawn the attention to mobile network researches due to its proven efficacy to address complex multi-domain problems, yielding close to optimal results [8]. Furthermore, the use of RL avoids the need for expert knowledge to mathematically model the environments to be optimized.

In each interaction with the environment, the agent sees a state $s(t)$ and selects an action $a(t)$. The action $a(t)$ in state $s(t)$ obtains a certain reward $r(t)$. Most reinforcement learning problems can be formulated as Markov Decision Process (MDP), a complete mathematical formalism of RL is developed in section 2.3. Most mobile communications solutions maximize the expected reward by learning the state-value or action-value functions [26]. Among the value-base methods, the most well-known are Q-learning and SARSA [27].

This method is very powerful for complex environments such as resource allocation and orchestration problems. However, the algorithm needs to explore the environment fully and this means that in a complex system such as a 5G network, the learning phase can be inefficient as it requires a large amount of time to reach an optimal solution. Deep Reinforcement Learning (DRL) addresses this issue by utilizing Deep Neural Networks (DNN) as function approximators, thereby reducing the complexity associated with traditional RL methods.

The best known DRL technique is Deep Q-learning Network (DQN). DNN is used as a function approximator for action selection in a discrete space, based on Q-learning. The use of DNN has led to the exploration of new problems that so far could not have been tackled by RL due to its high dimensionality. DRL has obtained good results in energy saving strategies for switching off cells [28] , optimal routing [29], mobile edge computing [30] or network slicing [31], to name a few examples.

The biggest drawback of these solutions is that a very large number of interactions with the environment is needed to explore it well enough to reach an optimal solution. This is a problem when you want to deploy it in a real production environment. What is done is to use simulators that emulate as closely as possible the behaviour of the network to train them in this environment beforehand.

2.3 Theorical foundations of Deep Reinforcement Learning (DRL)

2.3.1 Fundamentals of reinforcement learning

Reinforcement Learning (RL) is a subfield of Machine Learning (ML) that addresses the problem of the automatic learning of optimal decisions over time. It is based on optimal control theory and Markov Decision Process (MDP). Richard Bellman first explored it in the 1950s, within the framework of dynamic programming and quasilinear equations [32].

The RL method bases its learning on trial-and-error, a concept based on natural human learning. A simple example of an RL problem would be a child learning to ride a bicycle. In his first attempts, he will not last long on the bike and may even fall off. Through these negative experiences, he learns what actions he should not take in order not to repeat these mistakes. In the course of time he will have managed to learn to ride a bicycle as a result of a long training where he has learned from both his successes and his mistakes.

RL problems are expressed as a system composed of an agent and an environment. The interaction between the agent and the environment can be reduced to the following steps:

1. The environment generates information describing the system, known as state.
2. The agent observes the state provided by the environment and based on it selects an action.
3. The environment receives the action selected by the agent. From this, it moves to a new state and generates a reward.
4. The agent receives this information and learns from what has happened. It started from a state s_t, with an action a_t moved to state s_{t+1} and obtained a reward r_t.

The steps described are illustrated by the block diagram in Figure 2.14. When the cycle of (state $\rightarrow$ action $\rightarrow$ reward) completes, we say that one

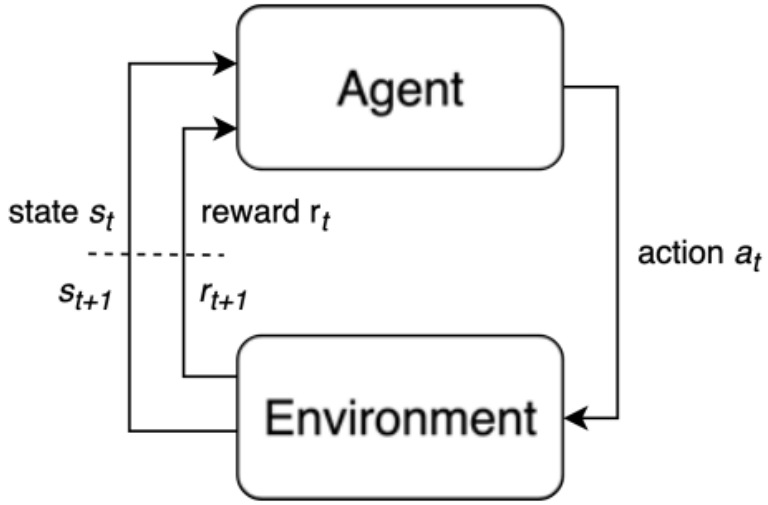


Fig. 2.14: Reinforcement learning operating diagram [9]

time step has passed. The cycle can repeat forever or terminate by reaching either a terminal state or a maximum time step $t = T$. The time horizon from $t = 0$ to when the environment terminates is called an episode. The (s_t, a_t, r_t) tuple is called an experience. A trajectory is a sequence of experiences over an episode, $\tau = (s_0, a_0, r_0), (s_1, a_1, r_1), ...$ The following is a formal description of the state, action and reward [9].

$$s_t \in \mathcal{S} \textit{ is the state, } \mathcal{S} \textit{ is the state space} \tag{2.2}$$

$$a_t \in \mathcal{A} \textit{ is the action, } \mathcal{A} \textit{ is the action space} \tag{2.3}$$

$$r_t \in \mathcal{R}(s_t, a_t, s_{t+1}) \textit{ is the reward, } \mathcal{R} \textit{ is the reward function} \tag{2.4}$$

The state space $\mathcal{S}$ is the set of all possible states in an environment. For example, if our environment is an operator's network cell, the state could be defined by the number of users it serves and the total throughput. So the state space would have dimension 2 and would be composed of an integer value and a float value. The total number of states in the state space will be given by all possible combinations of the variables that make up the state space.

Similarly, the action space $\mathcal{A}$ is the set of all possible actions defined to be taken in an environment. Continuing with the previous example, the action could be to increase or decrease the bandwidth of the cell. In this case, the action space would commonly be coded as 0 and 1, one value for each action.

Finally, reward can be positive, negative, large or small, but it's just a number whose purpose is to inform the agent how well it has behaved. It evaluates the goodness of the action taken.

Markov Decision Process

In a system of states, the transition from one state to another can be modelled mathematically with what is known as a transition function. In RL the transition function is formulated as an Markov Decision Process (MDP).

For a system to be an MDP, it must satisfy the markov property, which means that the future of the system from any state must depend only on the current state. The Markov property requires that the states of the system be distinct and identifiable. Consequently, only a single state is needed to predict the future behavior of the system, rather than relying on the entire history or the previous N states [11]. The transition function is expressed as follows:

$$s_{t+1}\ P\left(\ s_{t+1} | \, s_t, a_t\right) \tag{2.5}$$

The Markov property implies that the current state and action at time step t contain sufficient information to fully determine the transition probability for the next state at $t+1$. In RL an MDP is defined by a tuple of 4 variables: $\mathcal{S}$, $\mathcal{A}$, $\mathcal{R}(.)$ and $\mathcal{P}(.)$ [9]. The first three correspond to the definitions in equations 2.2, 2.3, 2.4 and $\mathcal{P}(.)$ is the state transition function of the environment defined in 2.5.

To formalize the agent's objective to be maximized, the return, we use the trajectory of an episode, $\tau = (s_0, a_0, r_0), ..., (s_T, a_T, r_T)$:

$$R\left(\tau\right) = r_0 + \gamma\ r_1 + \gamma^2 r_2 + \ldots + \gamma^T r_T = \sum_{t=0}^{T} \gamma^t \cdot r_t \tag{2.6}$$

The objective $J(\tau)$ will be the expected value of the return over many trajectories.

$$J\left(\tau\right) = E\left[\sum_{t=0}^{T} \gamma^t \cdot r_t\right] \tag{2.7}$$

The return $R(\tau)$ is the sum of the discounted rewards $\gamma^t \cdot r^t$ over all time steps, where $\gamma \in [0, 1]$ is the discount factor, a very important variable that modifies how the agent understands the reward. This parameter determines how much future rewards are valued compared to immediate rewards.

In the extreme case with $\gamma = 0$, the objective only considers the initial reward r_0. Conversely, when $\gamma = 1$ all rewards are considered equally regardless of when they are received. When γ is large, more weight is given to future rewards, allowing the agent to take a more "long-term" view.

Learnable functions and DRL algorithms

There are three primary functions to learn in reinforcement learning [9]:

- A policy π , which maps state to action: $a \sim \pi(s)$
- A value function, $V^{\pi}(s)$ or $Q^{\pi}(s, a)$, to estimate the expected return.
- The environment model, $P(s_{t+1})|s_t, a_t$.

A policy π is the strategy followed by the actions taken by the agent to maximize the objective. A policy can be stochastic, meaning that it can give different actions with certain probabilities for the same state. This is denoted as $\pi(a|s)$, which represents the probability of selecting the action a given the state s.

Value functions provide information about the quality of states and actions in terms of expected future rewards. They help the agent understand how beneficial it is to be in a particular state or take a particular action, with the goal of maximizing the cumulative long-term reward. The value function $V^{\pi}(s)$ evaluates how good or bad a state is, measuring the expected return of being in state s, assuming that the agent continues to act according to its current policy π. The Q-value function Q^{π} evaluates how good or bad a state-action pair is, measuring the expected return from taking action a in state s assuming that the agent continues to act according to its current policy π.

The transition function $P(s_{t+1})|s_t, a_t$ contains the probabilities that the environment goes to a state s_{t+1} from a state s_t because of an action a_t. With the transition function, an agent can predict the next state without the need to execute the action in the actual environment. This is useful for evaluating the potential consequences of different actions in advance.

From these 3 functions that the agent can learn from the environment, the 3 main families of Deep Reinforcement Learning (DRL) algorithms are derived: policy-based, value-based and model base methods. An overview of each family and how they are related is given in Figure 2.15. Model-based algorithms have had limited use in addressing mobile communication issues because network deployments are often too complex to model accurately. Consequently, determining transition probabilities and planning for future scenarios currently appears impractical. Most of the solutions that implement DRL in communication networks belong to the policy-based and value-based categories [8]. In this thesis, the DQN algorithm is discussed in greater depth (2.3.3), since it was the solution implemented in the practical part of the project.

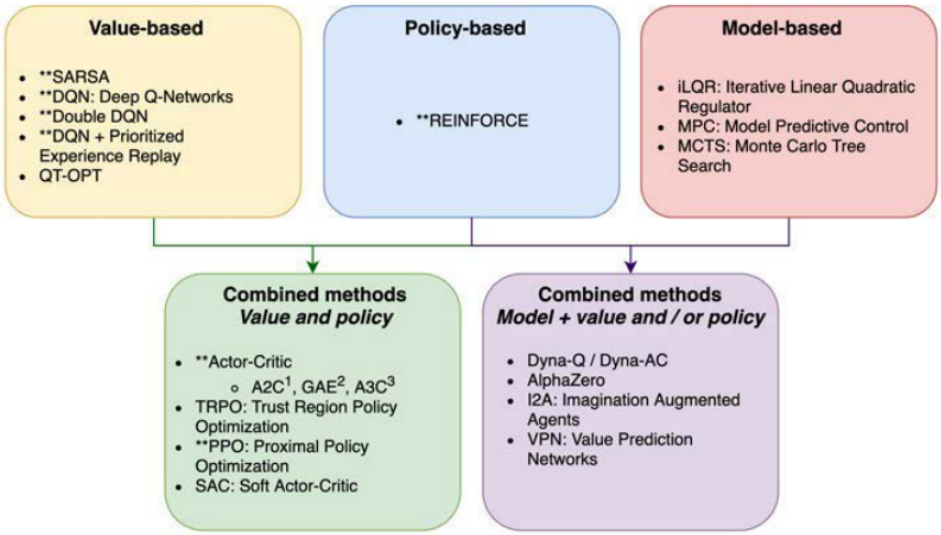


Fig. 2.15: Deep reinforcement learning algorithm families [9]

2.3.2 Deep Learning for Reinforcement Learning

Neural networks learn functions that modify an input value by providing an output, the layers of the neural network modify the value of the input. The weights of the layers, commonly known as parameters θ, will define the function learned by the neural network.

To learn a function, the neural network needs a sufficiently representative input data set and a way to evaluate the outputs produced by the network. One way is to have a set of input data with an associated target output. The goal is to train the network to predict the correct output given any input. To achieve this, a loss function is defined, which quantifies how far the network's predictions are from the actual targets. This option is implemented in supervised learning, for RL another strategy is needed. The most straightforward option in RL is to provide a scalar value indicating how good or bad the output is. That is, what has been defined as reward or return.

With a loss function, the network can change the parameters θ to minimize loss. This is known as gradient descent because we change the parameters in the direction of steepest descent on the loss surface in search of a global minimum [11]. Assume a neural network that learns a function $f(x;\theta)$ and $L(f(x;\theta), y)$ be the predefined loss function. A training step can be summarized as follows [9]:

1. Obtain an input value with its associated output (x, y) from the train-

ing dataset.

2. Obtain the output provided by the neural network in response to the input $\hat{y} = f(x, \theta)$.

3. The loss $L(\hat{y}, y)$ is calculated using the (known) target output and the neural network prediction.

4. Calculate the gradient (partial derivative) of the loss $\nabla_\theta L$ with respect to the parameters of the network.

5. Use an optimizer to update the network parameters using the gradient. For example, a stochastic gradient descent optimizer makes the following update: $\theta \leftarrow \theta - \alpha \nabla_\theta L$, where α is a scalar learning rate.

However, the training process described is not directly applicable to RL, neither the inputs x nor the target outputs y are given in advance. These values are obtained from the states and rewards that the agent receives from the environment after interacting with it. This represents a particular challenge for training neural networks in reinforcement learning. Furthermore, the current state and rewards of an environment are not independent of the states and rewards in previous instants of time. This violates an assumption of gradient descent, that data is identically and independently distributed (i.i.d.). The speed at which a network converges and the quality of training can be affected, so the various DRL techniques focus on being able to minimize these effects.

2.3.3 Deep Q-learning Network (DQN)

This section introduces the Deep Q-learning Network (DQN) algorithm proposed by Mnih et al. [33] in 2013.

The Q- and V-Functions

In section 2.3.1 the two value functions $V^\pi(s)$ and $Q^\pi(s, a)$ were presented conceptually. The mathematical formalism of these expressions is defined below in order to develop the theoretical foundation of DQN.

The Q-function measures the value of state-action pairs (s, a) under a particular policy π [9], as defined in Equation 2.8.

$$Q^\pi(s, a) = E_{a,\tau\sim\pi}\left[\sum_{t=0}^{T} \gamma^t \cdot r_t\right] \tag{2.8}$$

The value of $Q^\pi(s, a)$ is given by the expected value of the return, defined in equation 2.7, taking an action a in a state s and acting according to the

policy π. Value functions are always defined relative to a specific policy π, which is why they are indicated with a π superscript. Various policies can produce different future action sequences from a given state-action pair (s, a), potentially leading to different rewards.

$V^{\pi}(s)$ measures the value of state s under a particular policy π [9], as defined in Equation 2.9.

$$V^{\pi}(s) = E_{s,\tau\sim\pi}\left[\sum_{t=0}^{T}\gamma^{t}\cdot r_{t}\right] \tag{2.9}$$

The value of $V^{\pi}(s)$ is given by the expected value of the return from that state s onwards under a specific policy π. The two functions are related, $V^{\pi}(s)$ is the expectation over the Q-values for all the actions a available in a particular state s under the policy π.

$$V^{\pi}(s) = E_{a\sim\pi(s)}\left[Q^{\pi}(s,a)\right] \tag{2.10}$$

To facilitate the understanding of these two functions, let's use a chess game as an example, from the perspective of one of the players. The player will be represented by the policy π, a given configuration of the pieces on the board will be a state s. From a game situation (state) chess experts are able to intuit whether victory is near, this intuition is provided by the value of $V^{\pi}(s)$. Suppose this function takes values between 0 and 1. If the game situation is very close to being resolved in favour of a win, $V^{\pi}(s)$ will be very close to 1. In this particular case, it would be giving the probability of victory from a state in the form of reward.

However, in chess, given a good board configuration, there are different decisions that can end the game with more or less moves. This information would be provided by $Q^{\pi}(s, a)$ which gives a value for each possible movement. This value can be used to decide on the best move (action) to make in a particular position (state).

The disadvantage of learning $Q^{\pi}(s, a)$ is that the function approximation is more computationally complex and requires more data to learn from compared to $V^{\pi}(s)$. To learn $V^{\pi}(s)$ well, it is necessary that the data represent the state space reasonably well. Conversely, to learn $Q^{\pi}(s, a)$ well, it is necessary that the data represent all (s, a) pairs, not just the states [34].

Although $V^{\pi}(s)$ has an easier approximation function, it has an important disadvantage when the system is stochastic. The agent can act optimally by selecting the action a that provides a better expected return from state s. If the system is stochastic, taking action a from state s provides

different reward values. Therefore, the agent needs to try many times the action a from state s in order to obtain a reliable estimate of the expected reward. This is computationally very expensive and in some use cases unfeasible. $Q^{\pi}(s,a)$ avoids this problem because it directly learns the value of (s,a). As a result, RL algorithms which select actions using a learned value function tend to approximate $Q^{\pi}(s,a)$ [27].

Learning the Q-function in DQN

DQN learns the Q-function using Temporal Difference (TD) learning. The objective is to use a neural network that for a pair (s,a) produces an estimate of the Q-value. The key insight in TD learning is that Q-values for the current time step can be defined in terms of Q-values of the next time step [9].

From now on, the following change in the nomenclature will be made in order to facilitate the writing of the mathematical formalism: $(s_t, a_t, r_t) \rightarrow (s,a,r)$ and $(s_{t+1}, a_{t+1}, r_{t+1}) \rightarrow (s', a', r')$.

$Q^{\pi}(s,a)$ is defined recursively, as shown in Equation 2.11. This expression is known as the Bellman equation.

$$Q^{\pi}{}_{tar}\,(s,a) = r + \gamma \max_{a'} Q^{\pi}\left(s', a'\right) \tag{2.11}$$

The subindex *tar* is used to indicate that this is the target Q-value that the neural network is intended to generate. To estimate $Q^{\pi}{}_{tar}\,(s,a)$, DQN uses the maximum Q-value of all possible actions from that state. $Q^{\pi}{}_{tar}\,(s,a)$ can be calculated using the right-hand side equation 2.11 for each tuple (s,a,r,s',a'). To calculate the Q-values only the information of the next step is needed, not the whole trajectory of the episode. This allows the Q-function to be updated at each step, which is known as TD learning, instead of waiting until the end of the whole episode.

Off-policy algorithm

An important feature of DQN is that it is an off-policy algorithm. This means that the function that is learned is independent of the policy that is followed to select actions and generate experiences. An experience is the tuple composed of the reward obtained after taking an action from a state: $(s,a,r,s\prime)$. As seen in the equation 2.11, the value of $Q^{\pi}{}_{tar}\,(s,a)$ is updated with the maximum Q-value of the next state within all possible actions. Therefore, it does not depend on the action a' taken by the current policy in state s'. This occurs in on-policy algorithms such as SARSA, whose Bellman equation is shown in equation 2.12:

$$Q^{\pi}{}_{tar:SARSA}\,(s,a) = r + \gamma\; Q^{\pi}\left(s', a'\right) \tag{2.12}$$

In this case, ${Q^{\pi}}_{tar}(s,a)$ is updated with the Q-value of the next state s' and the next action a' according to the policy. The fact of being off-policy allows DQN's objective is to achieve the optimal Q-function that is defined in equation 2.13:

$$Q^*(s,a) = \max_{\pi} Q^{\pi}(s,a) = Q^{\pi^*}(s,a) \tag{2.13}$$

The optimal Q-function is the maximum expected reward that can be obtained by taking action a in state s and then following policy π^*. The optimal policy π^* is the best possible action selection strategy, i.e. the one that maximizes the expected reward in all states.

If we now think of the Bellman equation of DQN (equation 2.11) according to equation 2.13, if the estimate of Q^{π} is correct, then maximizing $Q^{\pi}(s',a')$ would be optimal. This implies that the policy that ${Q^{\pi}}_{tar}(s,a)$ corresponds to is the optimal policy π^* [9].

Action selection: ϵ-greedy

As discussed, TD learning is a method for learning to evaluate actions. However, what has not yet been mentioned is a method for selecting actions, what we call policy. If we assume that the optimal Q-function has already been learned, then each pair (s,a) has associated with it the best expected value of taking that action a in that state s. If the agent always selects the maximum Q-value, then it will be acting optimally. In order for the agent to reach the point of having learned a good policy, and as a consequence to select actions optimally, it is necessary that ${Q^{\pi}}_{tar}(s,a)$ has been updated during training to reach optimal values. Two important concepts come into play here: :

- **Exploration.** Consists of the agent selecting various options, even those that do not provide the best immediate reward, with the objective of acquiring information about the performance of each available option. That means, during the exploration phase, the agent will perform random actions in order to learn more about the environment which it operates.
- **Exploitation.** It occurs when the agent decides to focus on a particular option that has proven to be most profitable so far, with the objective of starting to generate consistent profits from that choice. In DQN when the agent selects the maximum Q-value.

If the agent always exploits, it will only know a part of all possible (s,a) pairs, therefore it will learn suboptimal or bad actions in those (s,a) pairs that it has not learned. If the agent always explores he will know a wide range of pairs (s,a) but he will never learn an optimal policy since he does

not select the actions that offer him the best reward. A trade-off between exploration and exploitation is important for good training.

It is common to use as an action selection strategy a ϵ-greedy policy. Under this policy, an agent selects the greedy action (exploitation) with probability $1 - \epsilon$ and acts randomly (exploration) with probability ϵ, which is known as the exploration probability. Exploring has the risk of getting bad results for some time, but it gives the opportunity to discover better states and ways to act. If the agent had access to the optimal-Q function it should act greedily, but while it is learning the Q-function, acting greedily may prevent it from improving.

A usual strategy to handle this tradeoff is to start training with ϵ values close to 1. At the beginning, the agent has not yet learned anything, so there is nothing to exploit. Over time, ϵ is gradually decayed, As the agent learns better Q-functions, and so better policies, there is less benefit to exploring and the agent should act more greedily [9]. This improves the efficiency of training as the agent focuses on better actions. The amount of exploration that an agent needs to perform is highly dependent on the environment in which it operates and the task that it must complete. Some environments might require a large amount of exploration, while others might be able to learn with little exploration.

Experience replay

In 1992 Long-Ji Lin observed that TD learning could be slow, since at each time step information has to be propagated backwards in the neural network [35]. To reduce this time, he proposed a Q-learning enhancement called experience replay.

An experience replay memory stores the k most recent experiences an agent has gathered. Each time an agent trains, one or more batches of data are sampled random-uniformly from the experience replay memory. Each of these batches is used in turn to update the parameters of the Q-function network [9]. Each batch usually contains experiences from different episodes and different policies, which reduces the variance of the parameter updates and stabilizes the training. If memory is full, the oldest experience is discarded, older experiences become less useful because an agent is less likely to visit the older states.

DQN Algorithm

The pseudocode for DQN with a ϵ-greedy policy is given in Algorithm 1. The Q-function estimate $\hat{Q}^{\pi}(s, a)$ is parametrized by a neural network with

parameters θ, denoted Q^{π_0}. First, experiences are generated according to

Algorithm 1 DQN training algorithm

1: Initialize learning rate α
2: Initialize exploration probability ϵ
3: Initialize number of batches per training step, B
4: Initialize number of updates per batch, U
5: Initialize batch size N
6: Initialize experience replay memory with max size K
7: Randomly initialize the network parameters θ
8: **for** $m = 1$ to MAX_STEPS **do**
9: Gather and store h experiences (s_i, a_i, r_i, s'_i) using ϵ-greedy policy
10: **for** $b = 1$ to B **do**
11: Sample a batch b from the experience replay memory
12: **for** $u = 1$ to U **do**
13: **for** $i = 1$ to N **do**
14: # Calculate target Q-values for each example
15: $y_i = r_i + \gamma \max_{a'} Q^{\pi}(s', a')$
16: **end for**
17: # Calculate the loss, for example using MSE
18: $L(\theta) = \frac{1}{N} \sum_i (y_i - Q^{\pi_\theta}(s_i, a_i))^2$
19: # Update the network's parameters
20: $\theta = \theta - \alpha \nabla_\theta L(\theta)$
21: **end for**
22: **end for**
23: Decay ϵ
24: **end for**

the ϵ-greedy action selection policy (line 9). To train the agent, B batches of experiences of size N are selected from the from experience replay memory. For each batch, U parameters of the neural network are updated in the following way:

- First, we calculate the target Q-values for each element in the batch (line 15).
- Then, we calculate the loss (line 18).
- Finally, calculate the gradient of the loss and update the network parameters θ (line 20).

After the training of a time step has been completed, the value of epsilon is decreased to follow the trade-off strategy between exploration and exploitation discussed above.

Chapter 3

System model and problem definition

This chapter describes the system model adopted for this project, describing the network design. It concludes by defining the problem statement addressed within this system.

3.1 System model

The system consists of a New Generation Radio Access Network (NG-RAN) formed by a set B of 5G NR Small cells (SCs) that are controlled by a radio resource allocation agent. This infrastructure can be used by an OTT provider or vertical industry players to establish specific service requirements in a 5G slice. A set U of User Equipment (UE) exist in the system, these users should be provided with enough resources to satisfy a guaranteed bit rate or service demand. User traffic is of type Enhanced Mobile Broadband (eMBB).

3.1.1 Network design model

The 5G network modelled supports scalable numerologies with subcarrier spacing of $2^{\mu} \cdot 15KHz$ ($\mu = 0, 1..., 4$). As shown in Figure 3.1, the gNB bandwidth is split into a set of Physical resource block (PRBs), each consisting of 12 consecutive subcarriers in the frequency domain. The number of PRBs is mapped to a specific bandwidth based on the numerology. The maximum permissible bandwidth is determined by the spectrum band in which NR operates. Specifically, the limit is $100MHz$ for the sub-6 GHz band and $400MHz$ for the millimeter wave band.

A number R_b of PRBs from the total available PRBs will be assigned to the SC $b, b \in B$. We denote the number of PRBs by the variable R, where

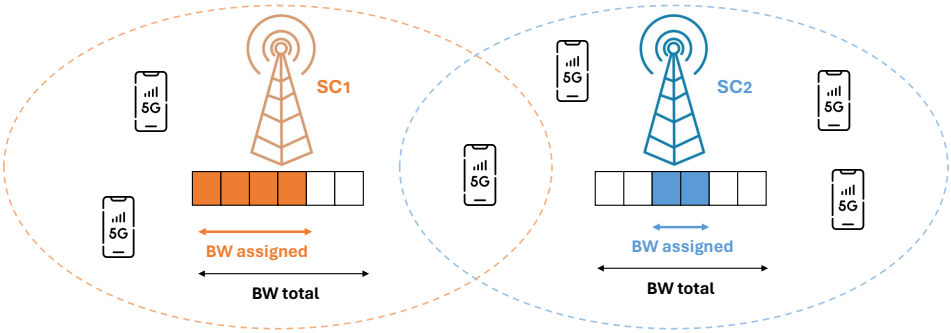


Fig. 3.1: NG-RAN model and example of bandwidth allocation.

the subscript b refers to the specific cell. The R_b PRBs assigned provides the SC bandwidth BW_b, which in turn determines the transmit power, ${P_b}^{TX}$ of the SC b.The transmission power is proportional to the allocated bandwidth. A power P_R is defined per PRB, so the transmission power of the SC is given by:

$$ {P_b}^{TX} = P_R \cdot R_b \tag{3.1} $$

The received power ${P_b}^{RX}(d)$ at a certain distance d when served by the SC b is given by:

$$ {P_b}^{RX}(d) = {P_b}^{TX} - L(d) \tag{3.2} $$

where $L(d)$ is the path loss at the distance d. The propagation model that defines the path loss of the UEs is based on linear regression, assuming NLOS situation:

$$ L(d) = L_0 + n \cdot \log_{10}(d) \quad (dB) \tag{3.3} $$

Where L_0 is a constant that represents the path loss at a reference distance $d_0 = 1m$, and n is the exponent of the variation with distance, which depends on the environment and the characteristics of the transmission medium.

The SINR experienced by a UE u in a bandwidth r is given by:

$$ \text{SINR}\,(u, r) = \frac{{P_{u,b}}^{RX}\,(d_{b,u})}{\left(\sum_{j \in B \setminus \{b\}} L_j \cdot \pi_j\,(r) \cdot {P_{u,j}}^{RX}\,(d_{j,u})\right) + P_N} \tag{3.4} $$

where L_j is the load factor of the cell j, The function $\pi(j)$ denotes a binary indicator that equals 1 if the spectrum portion r is assigned to cell j, and 0 otherwise. In the numerator of equation 3.4, the term represents the power received by user u from their serving cell b. The interference from neighbouring cells and the noise power P_N appear in the denominator.

The interference that cell j causes to user u is determined by the product of the cell j load factor and the power received by user u from cell j. This

interference occurs only if cell j is assigned the same PRBs as the serving cell of user u.

The cell load factor is determined by the relationship between service demand and cell capacity:

$$\hat{L}_j = \frac{\sum_{u|j=\Gamma(u)} D_u}{\sum_{u|j=\Gamma(u)} BW_u \cdot SE_u} \tag{3.5}$$

where D_u represents the service demand of the UE, SE_u denotes the average spectral efficiency of the UE and BW_u is the spectrum allocated to UE. $\Gamma(u)$ is a function that returns the cell j serving UE u. The adjusted load factor is defined as:

$$L_j = \min(\hat{L}_j,\ 1) \tag{3.6}$$

This ensures that the adjusted load factor does not exceed 1, indicating that the cell cannot handle more service demand than its capacity. The spectral efficiency of UE u at bandwidth r is derived from $SINR(u, r)$ according to the following SINR mapping [36]:

$$SE = \begin{cases} 0, & \text{SINR} < \text{SINR}_{\text{min}} \\ \alpha \cdot \log_2(1 + \text{SINR}), & \text{SINR}_{\text{min}} \leq \text{SINR} \leq \text{SINR}_{\text{max}} \\ \text{SE}_{\text{max}}, & \text{SINR} \geq \text{SINR}_{\text{max}} \end{cases} \tag{3.7}$$

where SE_{max} is the maximum achievable spectral efficiency with link adaptation, $SINR_{min}$ and $SINR_{max}$ are the minimum and maximum SINR values, respectively, and α stands for the attenuation factor, which represents implementation losses. Finally, the throughput of UE u is given by:

$$T\,(u) = \frac{BW_u}{|R_b|} \cdot \sum_{r \in R_b} SE\,(u, r) \tag{3.8}$$

The bandwidth assigned to UE BW_u depends on the resource scheduling scheme, in the project Round-Robin is used:

$$BW_u = \frac{BW_{PRB} \cdot R_b}{U_b} \tag{3.9}$$

where BW_{PRB} is the bandwidth per PRB and U_b is the number of UEs served by cell b that serves UE u. So the throughput expression reduces to:

$$T\,(u) = \frac{BW_{PRB}}{U_b} \cdot \sum_{r \in R_b} SE\,(u, r) \tag{3.10}$$

The throughput of UE u is calculated as the sum of the spectral efficiency over the allocated bandwidth, multiplied by the bandwidth of a PRB, and then divided by the number of users served by cell b. Further information about this network model can be found in [37].

3.2 Problem definition

This section defines the problem that this project aims to solve. All the assumptions and constraints considered in this problem are defined along the section.

In this project, we propose an energy saving agent that ensures minimum throughput for all users of a 5G network slice. The goal is to minimize bandwidth allocation to reduce transmission power, thereby enhancing the network's energy efficiency. However, this reduction in resources must ensure that minimum throughput is maintained as the primary requirement.

We denote T_u as the throughput of the UE at the edge served by SC b, that is, the user with the worst SINR. The primary requirement to be ensure is given by:

$$T_u \geq T_{min} \tag{3.11}$$

where T_{min} represents the minimum throughput threshold that the operator or service provider aims to ensure within the slice. We put focus on the throughput of the worst UE because if it guarantees the minimum throughput, the rest of the users served by the SC will also comply with this minimum throughput value. Once the throughput criterion is satisfied, the optimization objective shifts to minimizing transmission power, resulting in the following criterion:

$$Minimize\ {P_b}^{TX} \quad s.t \quad T_u \geq\ T_{min} \tag{3.12}$$

According to Equation 3.1, minimizing the transmission power requires the agent to decrease the number R_b of PRBs allocated to the SC b. Bandwidth directly affects throughput; therefore, a reduction in the number of allocated PRBs will lead to a corresponding decrease in throughput. For this reason, the minimization is subject to $T_u \geq\ T_{min}$, the primary requirement of the agent.

The agent's objective is to optimize this condition within each individual cell of the network, ultimately leading to a reduction in energy consumption across the entire network.

Chapter 4

Solution Design

This chapter describes the solution developed to solve the problem posed in section 3.2. First, the solution is outlined and introduced at a high level, followed by a detailed explanation of each of its components (action space, state space, reward algorithm).

Lastly, the implementation of the solution using the Gym reinforcement learning framework and the interaction with the DQN agent are described.

The code developed in the project is available in the GitHub repository referenced in [10]. The repository structure and brief descriptions of each script are detailed in Appendix D.

4.1 High Level Design

The problem defined in section 3.2 is addressed in this project using Deep Reinforcement Learning (DRL), in particular Deep Q-learning Network (DQN). Regardless of the method chosen, applying a Reinforcement Learning (RL) technique to a problem requires framing it as an interaction between an agent and an environment. The agent performs actions that modify the environment and receives an observation influenced by the action it has taken. The concepts of agent and environment were introduced in a general context in section 2.3.1. Below, they are specifically defined for our particular use case:

- Agent. A piece of code that takes actions that modify the bandwidth allocation in the gNBs, it is a radio resource allocation agent. The actions will involve adjusting the number of PRBs allocated within the cell, either increasing or decreasing them, and modifying the placement of the allocated bandwidth within the available spectrum (i.e., it decides the amount and what specific PRBs to allocate within the available spectrum).

- Environment. It is the New Generation Radio Access Network (NG-RAN) capable of receiving the agent's actions, applying them, and providing an observation that describes its current state, along with a reward that evaluates the effectiveness of the agent's action. As this is a proof of concept, the solution is not designed for a production environment. Instead, it uses a radio access simulator that implements the network model defined in section 3.1.1, emulating the behaviour of a real network

Figure 4.1 shows a diagram of the agent's operation over the environment (NG-RAN).

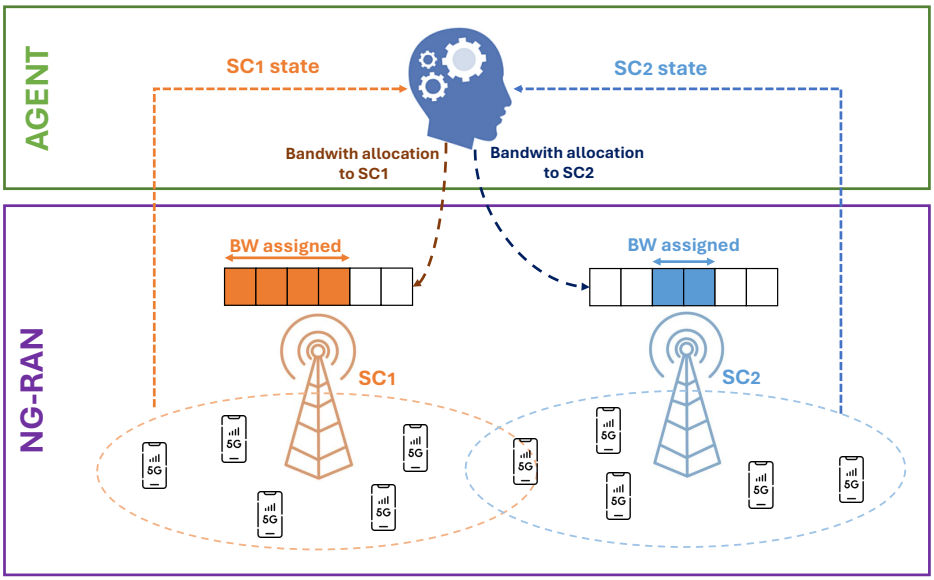


Fig. 4.1: Illustration of agent operation over the NG-RAN.

The agent must be an entity with expert knowledge regarding the optimization criterion followed. In the case of a traditional solution, a group of human experts define an algorithm that implements the desired optimization. In a DRL solution, the agent is a Neural Networks (NN) that has learned to optimize through reinforcement learning training. The agent takes actions on the environment, observes the current state, and evaluates the reward, which indicates the effectiveness of its actions. Through this process, it learns what actions to take and acquires expert knowledge for which it was designed.

Metaphorically, once the agent is trained and has acquired expert knowledge, the SCs will consult the agent for advice on improving its situation according to the established optimization criteria. Based on the information

provided by the SCs, the agent will recommend the necessary bandwidth modifications.

To ensure the agent acquires the desired knowledge, it is crucial to design a solution that guides its learning process. The three key concepts in designing an RL agent are the action space, state space, and reward algorithm. The following sections detail the design of each of these aspects.

4.2 Action Space

The action space defines the set of actions that the agent is capable of doing in the environment. This problem involves a radio resource allocation agent that modifies the bandwidth of the gNBs. As mentioned in network model design, depending on numerology, a certain bandwidth will be available. This bandwidth is split into a number of PRBs, the agent will decide how many and what specific PRBs within the available gNB spectrum to be allocated.

Set a numerology μ, there is a bandwidth BW in the frequency range $[f_1, f_2]$ that is split into a number R of PRBs where each PRB comprises a bandwidth BW_{PRB}. For the agent, the bandwidth allocation will be represented as an array of R positions, where each position corresponds to a PRB represented by an index, as illustrated in Figure 4.2.

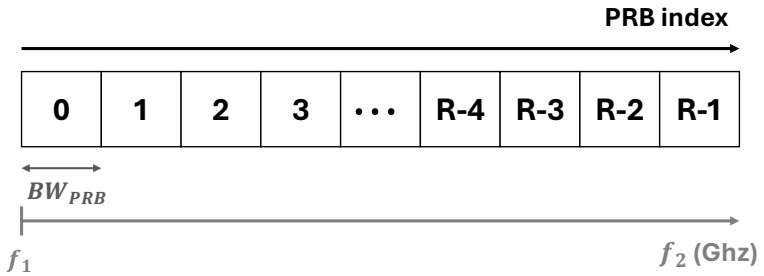


Fig. 4.2: Bandwidth allocation represented as an array of R PRBs.

The agent can modify the allocated bandwidth in two ways. First, it can increase or decrease it, adding or removing PRBs respectively. On the other hand, it can modify the position of the allocated bandwidth in the spectrum by shifting the assigned PRBs. An important aspect of the design is that the assignment of PRBs is contiguous, meaning the PRBs assigned in a SC comprise a single frequency range $[f_1, f_2]$ completely. From these two bandwidth modification options, 7 actions that the agent can perform are derived. These actions are detailed in Table 4.1. The number p of PRBs that the agent adds, deletes, or shifts is a design parameter that depends

ID	Description
0	Delete p PRBs from the left side
1	Delete p PRBs from the right side
2	Add p PRBs to the left side
3	Add p PRBs to the right side.
4	Shift PRBs p positions to the right.
5	Shift PRBs p positions to the left.
6	Not modify the PRBs.

Table 4.1: Action space

on the scenario and the specific optimization goals. Chapter 5 will present experiments and results where all the design values used are specified. Figure 4.3 shows a graphic example of the different actions of the agent on the bandwidth allocation array. In this example, the SC has 3 PRBs assigned, the number of PRBs that the agent adds, deletes and shifts is $p = 1$.

The defined actions have a direct impact on the optimization objective of the problem. The agent may increase and reduce the throughput of the UEs by respectively increasing and reducing the number of PRBs assigned to the SC. However, the placement of the allocated bandwidth within the available spectrum is also a critical factor affecting throughput. According to the defined network model (section 3.1.1), interference from neighbouring SCs appears in the denominator of Equation 3.4; As interference increases, the SINR experienced by UEs is reduced. Therefore, allocating bandwidth in a way that minimizes interference will be advantageous, enabling transmission power to be minimized effectively.

The agent can utilize all available actions to reduce interference while satisfying throughput requirements and minimizing allocated bandwidth. It can add PRBs to either end of the assignment, allowing it to select the side with the least interference. It can also delete PRBs from either end, enabling it to reduce allocated bandwidth on the side experiencing the most interference. The actions of shifting assigned PRBs are specifically designed to enable the agent to allocate bandwidth in the spectrum frequencies with minimal interference.

Finally, it is necessary to define an action that does not modify the bandwidth allocation. In this problem, the optimization objective aims to minimize allocated bandwidth while ensuring that the user's throughput at the edge meets a predefined minimum. The agent does not know beforehand what the final solution should be or when it should stop optimizing. As a consequence, it is necessary for the agent to have the option of taking no

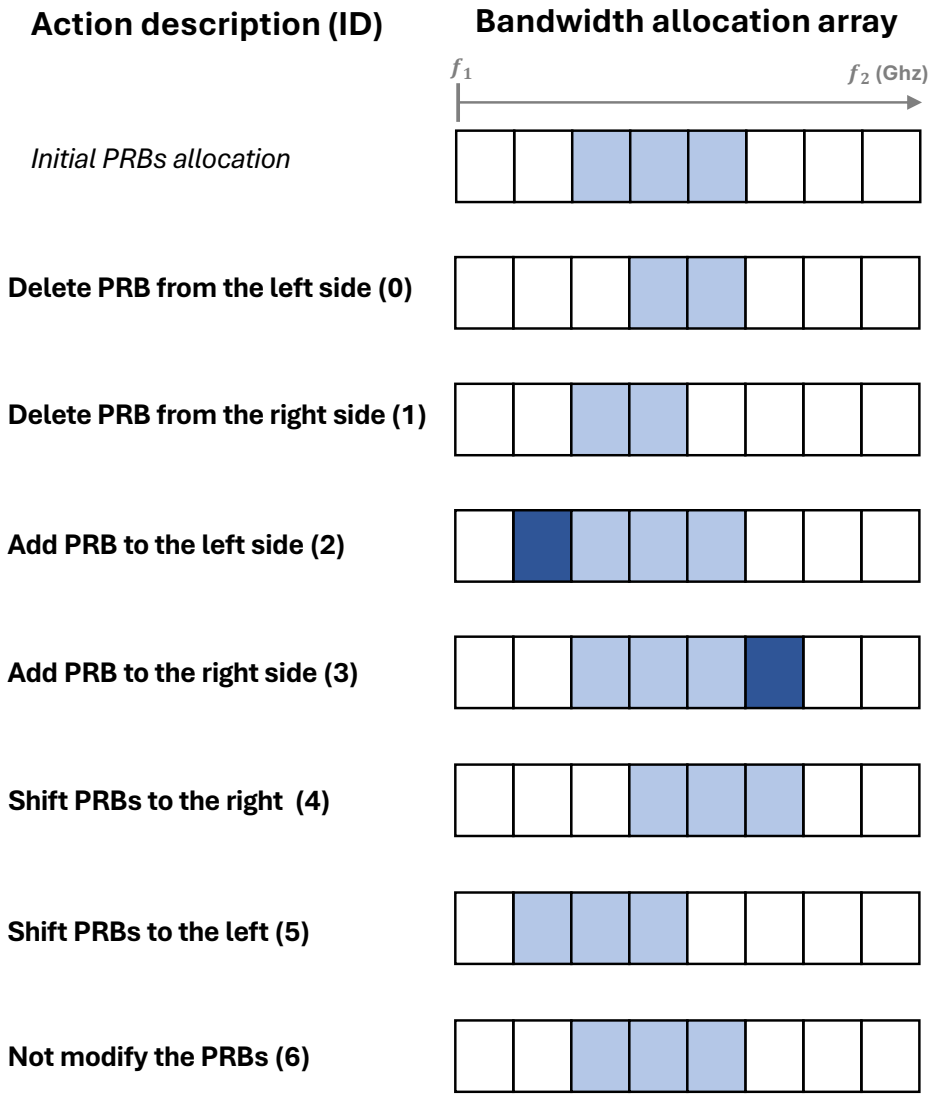


Fig. 4.3: Graphic example of the action space.

action (doing nothing) to discover if this is the best action. If the optimal decision for the agent is to refrain from modifying the bandwidth allocation, it indicates that it has identified an optimal solution where any alteration of the assigned PRBs would be detrimental. The way in which the agent is guided in learning the optimization criterion is developed in section 4.4.

4.3 State Space

The state space is the set of variables that collectively describe the current state of the environment. This set of variables is also known as observation, since the state that describes the environment is what the agent observes about it. The agent interacts with the SCs by adjusting their bandwidth. In response, the environment provides an observation that describes the state of the SC the agent is interacting with. In the solution designed in this project, the state of a SC is described by variables listed in Table 4.2.

Variable	Abbreviation
UE throughput at the edge	T_u
Number of PRBs assigned to the SC	R
PRB position flag	f_p
Number of interfered PRBs	R_i
Interference PRB position flag	f_i

Table 4.2: Abbreviation of state space variables

With N being the number of neighbouring cells of a SC, the observation that defines the state of a SC is formalized as:

$$obs = \left[T_u, R, f_p, R_{i0}, \cdots, R_{iN-1}, f_{i0}, \cdots, f_{iN-1}\right] \tag{4.1}$$

The number of interfered PRBs and the interference PRB position flag are variables defined for each neighbouring cell. Consequently, they can be reformulated as follows:

$$\vec{R_i} = [R_{i0}, \cdots, R_{iN-1}] \tag{4.2}$$

$$\vec{f_i} = \left[f_{i0}, \cdots, f_{iN-1}\right] \tag{4.3}$$

The observation can be expressed in a simplified form as:

$$obs = \left[T_u, R, f_p, \vec{R_i}, \vec{f_i}\right] \tag{4.4}$$

The size of the observation is $3 + 2N$, thus it is dependent on the specified number of neighbouring cells. Throughout the document, various alternatives are discussed to maintain a reasonable observation size in scenarios where there may be an excessively large number of potential neighbouring cells.

The following subsections provide a complete description of each of the variables that make up the observation.

4.3.1 UE throughput at the edge

Each SC serves a set of UEs that experience a specific throughput. As mentioned in section 3.2, the agent works on the throughput of the worst UE because if it guarantees the minimum throughput, the rest of the users served by the SC will also comply with the minimum throughput value. As a consequence, in the observation that describes the state of the SC, the throughput of the UE with the worst SINR is provided. That is, the UE at the edge.

The throughput is a positive value, measured in bits per second, that can vary within a range determined by the specific scenario considered. The parameters that constitute the network model, along with the number of cells, users, and their positions in the scenario, influence the throughput values. Regardless of the range of values, throughput is a continuous variable. In the solution designed for this project, this variable is discretized to reduce its variance and facilitate the agent's learning process.

In RL learning, the agent must thoroughly explore the action-state space to discover the best solutions. If the state space is too large, it will be more difficult for the agent to converge to an optimal solution. Discretizing a continuous variable significantly reduces the range of values it can take, thereby decreasing the number of distinct states within the state space. Appendix C shows the improvement in the agent's learning as a consequence of reducing the dimensionality of the state space by discretizing the throughput variable. To achieve this, an experiment was conducted to compare the learning results of the agent using two different state spaces. The first space considered the raw throughput variable as a continuous value, while the second space utilized the discretized throughput option employed in the project.

In order to determine the appropriate number of bins for discretizing the range of throughput values, it is necessary to conduct a preliminary study of the probability density function associated with the variable. In chapter 5, where the experiments and results obtained in specific scenarios are presented, the established number of bins is detailed.

The presence of UE throughput at the edge within the state space is inherent, as it constitutes one of the fundamental variables in the defined optimization criterion.

4.3.2 Number of PRBs assigned to the SC

In the optimization criterion defined in equation 3.12, the minimization of the transmission power is defined. The transmission power is proportional

to the allocated bandwidth, which is established by the number of PRBs assigned to the SC. Therefore, it is essential that the agent knows the number of PRBs assigned to the SC since in this way it is considering the amount of transmission power.

This variable is discrete and can take values in the range $[1, R_T]$ with R_T being the total number of PRBs available in the gNB.

4.3.3 PRB position flag

The purpose of this variable is to provide the agent with information regarding the allocated bandwidth's position within the spectrum. The objective is for the agent to recognize when the assigned PRBs are located at the boundaries of the available spectrum. So that the agent does not take actions to add or shift PRBs outside the allowed frequency limits.

This variable depends on the number p of PRBs that the agent adds, deletes or shifts with his actions. The flag can take the values shown in Table 4.3.

Flag value	Description
0	The PRBs are more than p positions from the edges.
1	The PRBs are less than p positions from the left edge.
2	The PRBs are less than p positions from the right edge.
3	The PRBs are less than p positions from both edges.

Table 4.3: PRB position flag values

Figure 4.4 shows an illustrative example of the operation of this flag for a value of $p = 1$. That is, the number of PRBs that the agent adds, deletes or moves in this example is 1. For example, when $flag = 1$, the allocated PRBs are positioned at the left edge of the available spectrum. Consequently, the agent should refrain from taking actions such as adding one PRB to the left or shifting the allocation towards the left side. The agent will learn this through the reward algorithm, but this variable needs to be in the state space so it can associate the reward with this flag.

Another alternative to provide this type of information to the agent is to use the PRB index in the array (see Figure 4.2). However, in this way, the variable could take R_T possible values. With the flag designed in this solution, the variable can take 4 possible values, which considerably reduces the dimensionality of the state space.

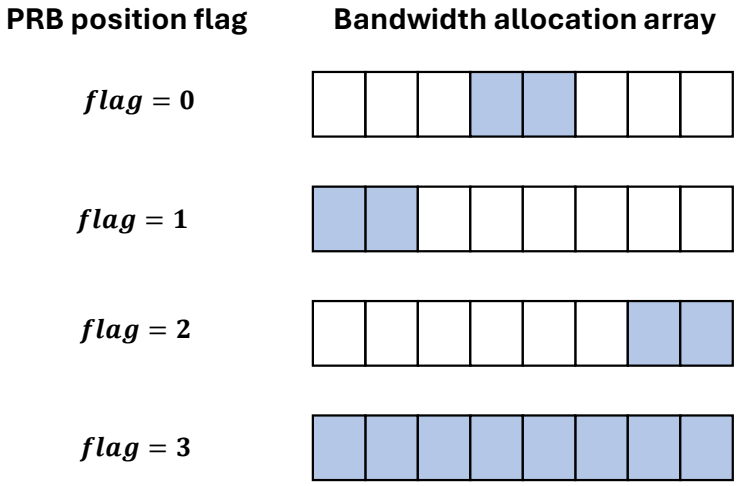
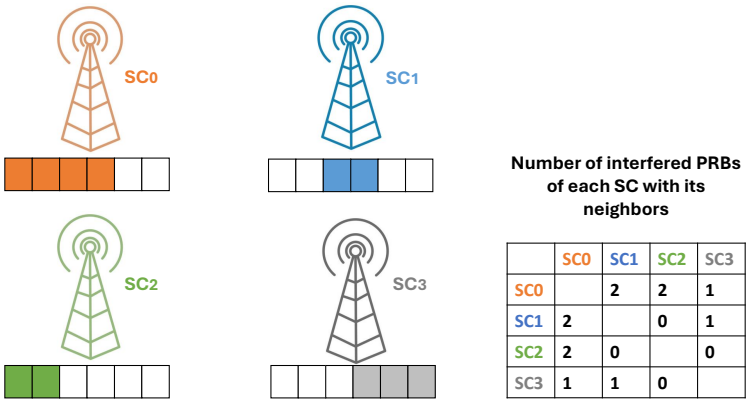


Fig. 4.4: Graphic example of the PRB position flag.

4.3.4 Number of interfered PRBs from each neighbour.

The bandwidth assigned to a gNB may overlap with that of another gNB within the same spectrum. If the adjacent gNB is in proximity, this overlap can lead to interference. The state space includes the number of PRBs of the SC that have interference with each of the SCs defined as neighbours. Let N be the number of neighbours of an SC, N variables will appear in the state space that will indicate the number of PRBs interfered with each neighbour. Figure 4.5 shows an example for a scenario of 4 SCs with a certain allocation of PRBs. All SCs are defined as neighbours. The table in the figure shows the number of interfered PRBs of each SC with its neighbours.



	SC0	SC1	SC2	SC3
SC0		2	2	1
SC1	2		0	1
SC2	2	0		0
SC3	1	1	0	

Fig. 4.5: Graphic example of interference in PRBs.

4.3.5 Interference PRB position flag

With the previous information, the agent only knows the number of PRBs of the SC with interference, but does not know the location of those PRBs in the spectrum. So that the agent has complete information about the interference, each number of interfered PRBs is accompanied by a flag called: "Interference PRB position flag".

The purpose of this flag is to indicate on which side of the spectrum the interference is occurring so that the agent can associate this flag with actions that help reduce the interference. The flag can take the 3 values indicated in table 4.4.

Flag value	Description
0	There is no interference
1	The interference is on the right side. The best action is to shift left or delete PRBs from right side.
2	The interference is on the left side. The best action is to shift right or delete PRBs from left side.

Table 4.4: Interference PRB position flag values

Figure 4.6 illustrates the four potential scenarios of interference between an SC and its neighbouring cell, along with the corresponding values of the flag indicator for each case. In situation A of Figure 4.6, the interfered PRBs are on the left side for SC_0 and on the right side for SC_1. The best

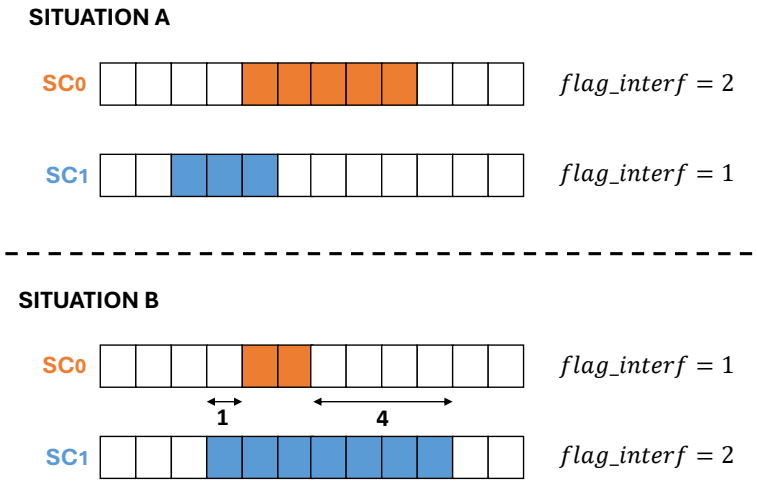


Fig. 4.6: Four potential scenarios of interference and the interference PRB position flag value for each case.

action the agent can take for SC_0 is to shift the assignment to the right or delete PRBs from the left side. Therefore, the value of the flag in SC_0 is 2. Conversely, the flag in SC_1 has a value of 1, as the best action for the agent is either to shift the assignment to the left or to delete PRBs from the right side.

Situation B in Figure 4.6 represents a different interference scenario; however, the Interference PRB position flag behaves in the same manner. The PRBs allocated in SC_0 are entirely interfered with by the allocation in SC_1, as the bandwidth assigned to SC_0 falls within the bandwidth assigned to SC_1. The best action for SC_0 to avoid interference is to shift the allocation to the left, there is less distance on this side towards an area of the spectrum without interference. Therefore, the value of the flag in SC_0 is 1. The opposite happens to SC1, to avoid interference it is easier for it to shift its assignment to the right or delete PRBs from the left side, so the flag in this case has a value of 2.

The interference PRB position flag together with the number of interfered PRBs provides the agent with complete information to be able to be aware of the interference and act accordingly.

4.4 Reward algorithm

So far, the actions that the agent can take and the set of variables that describe the state of the environment have been defined. We understand how the agent can modify the environment and which variables it receives to determine the effects of its actions on the environment. The action space and the state space are two key concepts in any RL problem, since they represent the communication between the agent and the environment. However, for the agent to learn the desired optimization objective and become an entity with expert knowledge, the key piece of the system is the reward algorithm.

The way in which the agent is rewarded defines the behaviour it learns, or what is referred to as policy in RL. As presented in section 2.3.1 in general terms, the agent takes an action in the environment and receives a reward along with the new state of the environment (observation). In our problem, the agent will modify the bandwidth of a SC. The environment will provide the agent with an observation, as described in equation 4.1, and indicate the effectiveness of its action through a reward.

The reward will simply be a number, during the learning phase the agent will try to maximize the accumulated reward which will indicate good performance. Consequently, it is important to design the reward algorithm to

align with the optimization objective defined in the problem. If the reward is directly related to the objective, it will guide the agent towards the solution during the learning process.

The designed reward algorithm is directly related to the criteria defined in equations 3.11 and 3.12. It utilizes the observation variables from equation 4.1 to determine the reward value. So the agent will be able to directly relate the tuple (a, s, s') to a reward value. Where a is the action that the agent has taken, s is the state from which it starts and s' is the new state after the action.

Figure 4.7 shows the block diagram of the reward algorithm designed in the project. It is divided into 3 stages that are explained in the following sections. Three state space variables participate in the reward algorithm: throughput of the UE at the edge, number of PRBs assigned to SC and total number of PRBs with interference. In the block diagram, throughput is denoted by "thr", the number of PRBs assigned to the SC as "$PRBs$", and the total number of PRBs with interference as "$interf\_PRBs$".

The index t indicates the current time step, with $t-1$ being the previous time step.

It should be noted that the total number of PRBs with interference is not strictly the sum of the $R_{i0}, \cdots, R_{iN-1}$ variables of equation 4.1. These variables indicate the number of PRBs, with interference with each neighbouring cell. The total number of PRBs that a SC has with interference is not simply the sum of these variables. For instance, a SC may have the same 5 PRBs interfered with by all of its N neighbours. In this scenario, the total number of interfered PRBs would be 5, not $5N$.

4.4.1 First stage: Guarantee minimum throughput

As specified in section 3.2, where the problem is defined, the primary requirement that must be ensured is that the throughput of the UE at the edge exceeds the specified minimum threshold. Therefore, the initial step of the reward algorithm is to verify that this requirement is fulfilled. This stage of the algorithm is inside the red box in Figure 4.7.

First, the reward algorithm checks if the throughput exceeds the minimum. If it does not exceed it, the error with respect to the minimum value is calculated and rewarded according to a linear relationship:

$$Reward = 5 \cdot E - 5 \quad if \quad thr < thr_{min} \tag{4.5}$$

where E is the relative error between the edge UE throughput and the

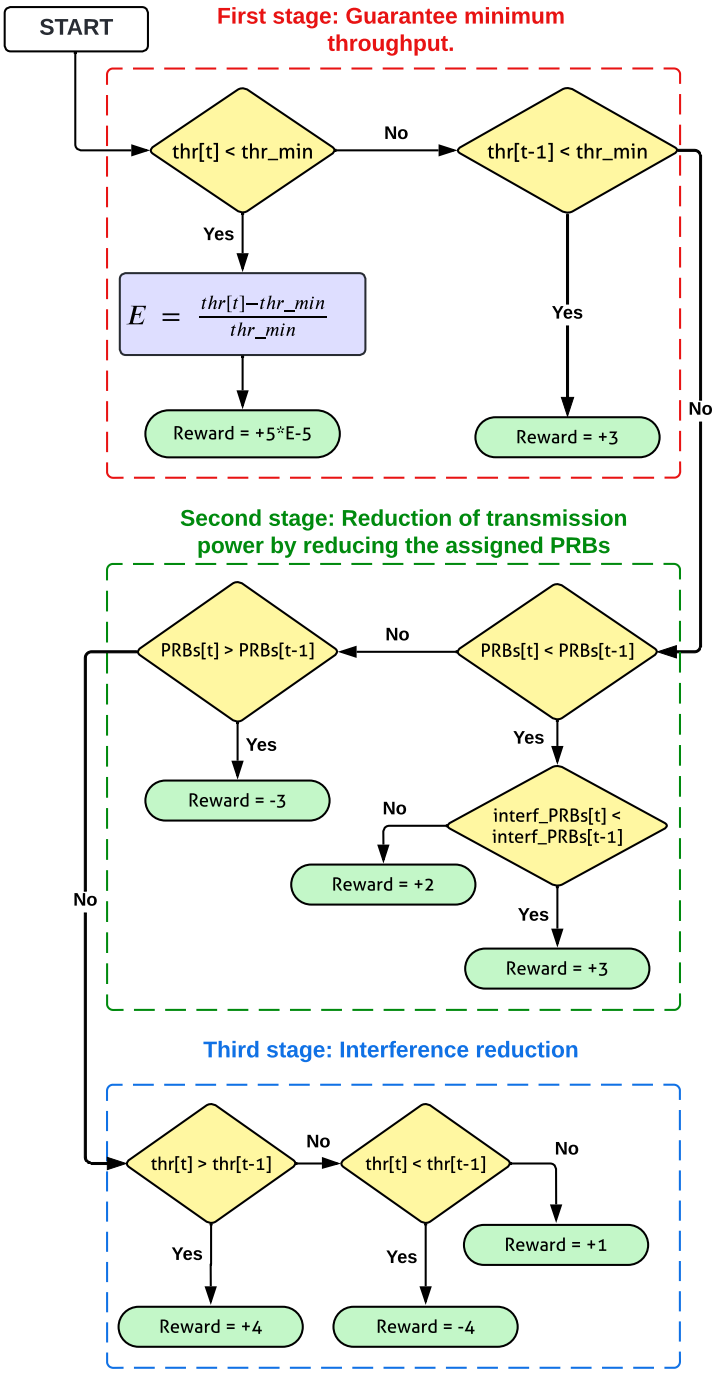


Fig. 4.7: Reward algorithm.

minimum throughput:

$$E = \frac{thr - thr_{min}}{thr_{min}} \tag{4.6}$$

Since throughput is a positive variable, its minimum value is zero, so the relative error takes values in the range $[-1, 0)$. Figure 4.8 depicts the linear correlation between reward and relative error. The reward diminishes proportionally with the deviation of throughput from its minimum value. When $thr < thr_{min}$ the reward moves in the range $[-10, -5)$ so that the

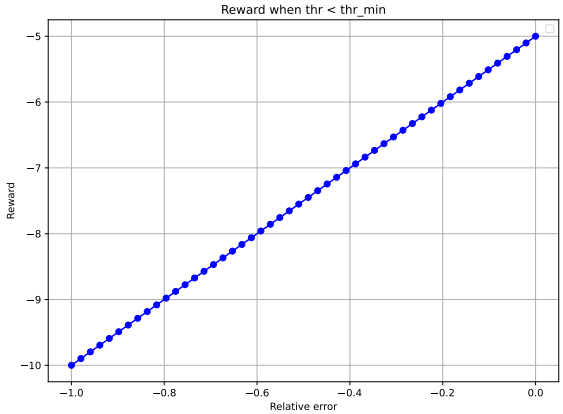


Fig. 4.8: Relationship between reward and relative error when $thr < thr_{min}$.

agent can know how far it is from the minimum throughput.

If the throughput value is greater than the minimum, it is checked to see what the value was in the immediately previous time step. If the throughput in the previous time step was below the minimum threshold, it indicates that the agent's action was effective in surpassing the threshold. Consequently, a reward of +3 is issued. On the contrary, if the minimum threshold was already reached in the previous time step, we proceed to the second stage of the reward algorithm.

4.4.2 Second stage: Transmission power minimization

Once UE throughput at the edge surpasses the predefined minimum (equation 3.11), the goal shifts to minimizing transmission power (equation 3.12).

According to the definition of transmission power (equation 3.1) in the proposed network model, the power directly depends on the assigned bandwidth. Therefore, the only method to decrease the transmission power is by reducing the bandwidth allocation. The agent can achieve this by executing the "delete assigned PRBs" actions, as long as the throughput does not drop

below the allowed level. However, if the second stage of the reward algorithm has been reached, it means that the throughput criterion is already being met.

The second part of the reward algorithm appears inside the green box in Figure 4.7. We check whether the number of PRBs allocated in the current time step is greater or less than those allocated in the previous time step. If the assigned PRBs have been increased, it is negatively rewarded with a value of -3. On the other hand, if the PRBs have decreased their reward is positive, with a value of $+3$ if the interference has decreased, or $+2$ if the interference remains unchanged.

Reducing the number of allocated PRBs is rewarded positively since it reduces transmission power. However, this reward is enhanced if a reduction in interference is also achieved. If we recall the actions available to the agent, there are two options for deleting PRBs: from the right side or from the left side. If the interference is located on one side, it is beneficial to delete the PRBs on that side. This enables the agent to correlate the action of deleting PRBs from a specific side with the value of the "Interference PRB position flag", which indicates the side where interference is located.

When interference is avoided, throughput increases, enabling further reduction in allocated bandwidth and consequently in transmission power.

If the number of assigned PRBs remains unchanged between consecutive time steps, it proceeds to the third stage of the reward algorithm.

4.4.3 Third stage: Interference reduction

If the number of assigned PRBs remains constant between consecutive time steps, the agent has only executed 3 out of the 7 possible actions:

- Shift PRBs p positions to the right.
- Shift PRBs p positions to the left.
- Not modify the PRBs.

According to the network model defined in this project, a variation in interference always has an impact on the throughput (equations 3.4, 3.7, 3.10). In the third stage, the reward algorithm checks the throughput value between consecutive time steps. If the throughput has increased, it means that the agent has shifted the allocation and reduced the interference. In the opposite case, the displacement increases the interference and therefore the throughput is reduced.

This stage of the algorithm is located inside the blue box in Figure 4.7. Shifting assignments to reduce interference is rewarded with a +4 value, while increasing interference is penalized with a −4 reward.

The reward is higher for reducing interference (+4) compared to reducing the number of PRBs (+3 or +2), despite the primary objective being to minimize PRBs and thereby transmission power. The reason is that to maximize the reduction in allocated bandwidth while ensuring the throughput criterion, minimizing interference is essential. Maximum throughput is achieved by minimizing interference as much as possible, according to equation 3.4. Once maximum throughput is attained, bandwidth can be gradually reduced until ideally reaching the threshold $thr = thr_{min}$. However, if there is interference in the bandwidth allocation (which can be eliminated), the optimization of the criterion defined in equation 3.12 will never be optimal.

The end of the third stage and the reward algorithm is reached if the throughput did not change between two consecutive time steps ($thr[t-1] = thr[t]$). This can only be achieved if the agent takes the action that does not modify the PRBs, the action with $ID = 6$ in Table 4.1, and the throughput exceeds the minimum (otherwise it would not be possible to pass the first stage of the reward algorithm). In this case, a reward of value +1 is provided.

Rewarding the action that "does nothing" is key to the agent's correct learning. In this optimization problem, the agent does not know in advance when it should stop optimizing; it only knows that it should minimize equation 3.12 as much as possible. The only way for the agent to recognize that it has reached an optimal solution is when any action that modifies the bandwidth becomes detrimental. At this point, not modifying the PRBs should be the best action the agent can take.

To enable the agent to learn this behaviour, it is essential to appropriately calibrate the values within the reward algorithm. The reward given for doing nothing (+1) is smaller than any good action, such as reducing assigned PRBs (+3 or +2) or reducing interference (+4). But it must also be greater than any bad action, such as increasing PRBs (−3) or increasing interference (−4). Thus, the agent will identify the optimal action of doing nothing only when there is no interference left to delete and when further reducing PRBs would cause the throughput to drop below the established minimum.

Since the agent aims to maximize the reward accumulated during the episode, it is crucial to design the reward values in a way that prevents the agent from entering a loop of repeatedly performing and undoing the same

action. For example, if the agent adds p PRBs on the right side (action $ID = 3$) and in the next step removes p PRBs from the right side (action $ID = 1$), it's not really doing anything. Therefore, the goal of the reward algorithm is to prioritize action where the PRBs are not modified (action $ID = 6$) over actions that result in a do-undo cycle. This is achieved by properly designing the reward values. if the agent takes action $ID = 3$, which adds PRBs to the right side, and then follows with action $ID = 1$, which removes them, it will receive a reward of $+3$ for the first step and -3 for the second step. This results in a cumulative reward of 0. Conversely, if action $ID = 6$ is performed in two consecutive time steps, which does not modify the PRBs, an accumulated reward of $+2$ is obtained. As a result, the agent will favour this latter action, aligning with the desired objective.

It should be noted that this behaviour only occurs when the minimum throughput criterion is being guaranteed (equation 3.11). If the throughput does not meet the minimum, the reward algorithm remains in the first stage and the reward is governed by the linear relationship in Figure 4.8.

Lastly, it is important to note the presence of a specific penalty assigned to prohibited actions. Prohibited actions include attempts by the agent to add PRBs beyond the available spectrum or to shift assignments outside the available spectrum. When this occurs, the environment provides a reward of -12 (the lowest reward value of the algorithm) and returns the same state that the agent started with. The following sections describe the main components of the solution developed in Gym.

4.5 Development of the solution with Gym framework

The framework used to define the agent-environment communication is the Python Gym library [38], developed and maintained by OpenAI. Gym offers a diverse range of environments for reinforcement learning problems and enables users to define new environments, as is our case. The significance of Gym lies in its use of a unified interface for defining any environment, which will be adhered to in this project.

4.5.1 Gym class components

The environment is represented in Gym by the Env class and is made up of 4 main pieces [11]:

- **action_space.** This field of the class defines the set of actions that the agent is capable of doing in the environment. Actions can be dis-

crete, continuous, or a combination of both. In our proposed solution, we utilize a discrete action space consisting of seven distinct actions. These actions are represented by the values $[0, 1, 2, 3, 4, 5, 6]$, with each value serving as an identifier corresponding to the actions detailed in Table 4.1.

- **observation_space.** The space of observations is the equivalent of the space of states defined in the theoretical foundation of RL (section 2.3.1). In Gym, the state provided by the environment is known as observation because it is what the agent observes about the environment. This observation is composed of an array of variables that collectively describe the current state of the environment. Similar to actions, these variables can take discrete values, continuous values, strings, or a combination of them. In our problem, the observation consists of discrete variables as outlined in Section 4.3. The observation is represented as an array with a size of $3 + 2N$, as specified in Equation 4.1.Where N is the number of neighbours that the SC has defined.

- **reset().** This method resets the environment to an initial state, returning the initial observation.

- **step().** This method enables the agent to take an action and provides feedback on the outcome, including the next observation and the reward obtained.

4.5.2 Reset method

As introduced in the theoretical foundation of RL, communication between the agent and the environment occurs through what we call steps. A sequence of steps, starting from when the agent begins taking actions until it concludes, is called an episode. Therefore, at the start of an episode, it is necessary to reset the environment to an initial state with reset method. From here on, the step method is used for the agent to take actions until the episode concludes, where the environment would be restarted for the next episode. This behaviour is how any RL agent training works. The reset() and step() methods must include all the necessary components that make it possible to train the agent in our particular problem.

The scenario implementing the environment in which the agent operates consists of a number B of Small Cells, a number U of UEs and their positions within the dimensions of the scenario. Additionally, each SC will have a bandwidth allocation that will be determined by the number R of PRBs assigned and its position in the available spectrum. During the episode, the agent modifies the bandwidth of only one SC, referred to as the agent SC.

The PRB assignment for the rest of the SCs in the scenario remains static throughout the episode.

The reset method will be responsible for restoring the environment to its initial state, allowing the agent to start introducing actions within the environment. Figure 4.9 shows the flow chart that describes the operation of the reset method. Initially, the environment is configured with a specific number U of UEs, a number B of SCs, and their positions within the scenario dimensions $[\vec{B_x}, \vec{B_y}]$. In addition, the number p of PRBs that the agent modifies with its actions (adds, deletes or shifts) is also configured.

The remaining variables required for the environment to operate are randomized. These include the positions of the UEs, the bandwidth allocation to each SC, and the specific SC that the agent will control. For the agent to effectively learn the optimization objective in the specific scenario being trained, it must be exposed to a wide range of possible situations that may occur. For this reason, in each episode the initial state is randomized.

The randomization of the allocated bandwidth in the SCs involves randomly determining both the number of PRBs allocated and their positions

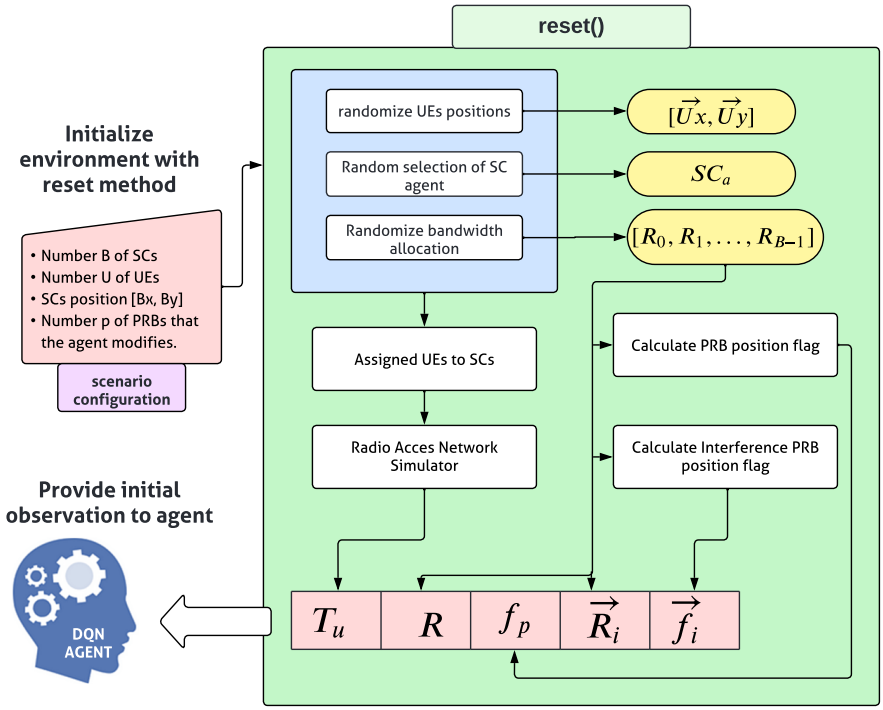


Fig. 4.9: Reset method flowchart.

within the spectrum. With R representing the number of PRBs and B the number of SCs, the array $[R_0, R_1, ..., R_{B-1}]$ specifies the number of PRBs allocated to each SC in the scenario.

Once the positions of the UEs $[\vec{U_x}, \vec{U_y}]$ have been defined, the process of assigning UEs to their serving SCs is performed. This process is based on the path loss according to the propagation model defined in the proposed network model (section 3.1.1). The details of this procedure are described in Appendix A. This assignment occurs only once at the beginning of each episode because it is based on path loss, and the positions of UEs remain static throughout the episode.

Once each UE has designated its serving SC, the radio access network simulator is executed. The simulator implements the proposed network model and calculates the throughput of all UEs. It provides the throughput of the UE at the edge served by SC_a (the cell controlled by the agent), which is located at the first position of the observation array. Implementation details of the radio access network simulator are described in Appendix B.

On the other hand, the values of the "PRB position flag" (section 4.3.3) and "interference PRB position flag" (section 4.3.5) are calculated. These flags, along with the throughput, the number of PRBs assigned to SC_a, and the number of PRBs interfered with each neighbouring cell $\vec{R_i}$, constitute the observation that describe the state of the environment.

4.5.3 Step method

The agent receives the initial observation and is ready to introduce actions in the environment using the step() method. The flow chart of this method is shown in Figure 4.10. The agent's actions within the environment will modify the bandwidth allocation for the specified SC_a, while the allocations for the remaining SCs will remain unchanged. Once the change is applied, the flags are calculated and the throughput of the worst UE is obtained by running the radio access network simulator. The observation is formed in the same way as in the reset() method. The novelty of the step() method is the calculation of the reward using the reward algorithm described in section 4.4. The environment provides the agent with the observation and the reward that the agent will associate with the state it started from and the action it took.

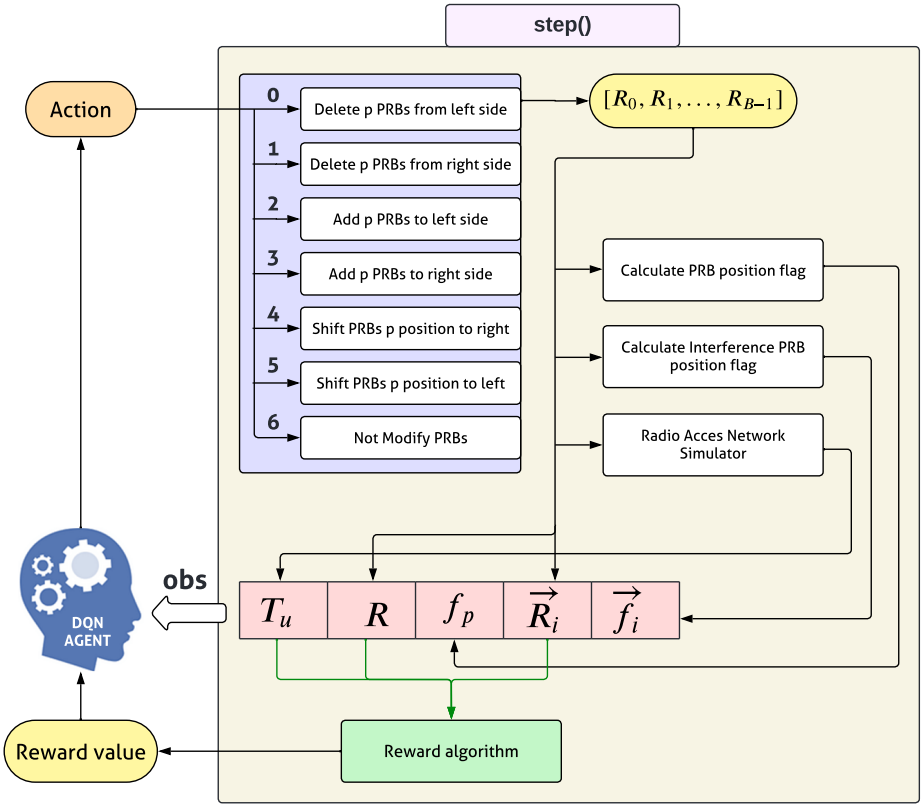


Fig. 4.10: Step method flowchart.

4.5.4 Agent training workflow

Once the reset() and step() methods are defined, we can build the complete process that makes up the training of the agent. In certain reinforcement learning problems, the agent has a defined stopping condition, which could be achieving the desired solution or reaching a forbidden state. In these cases, the duration of an episode is determined by these ending conditions. As mentioned in previous sections, in the problem addressed in this project, there is no final solution that the agent can know to end the episode. A specific number of steps per episode is established, and the episode ends when this number is reached. Figure 4.11 shows the flow chart of the agent training process. This diagram shows the communication between agent and environment through the reset() and step() methods. The number of episodes and the number of steps per episode are training configuration parameters. In this project, a DQN agent implemented using the stable_baselines3 [39] Python library has been utilized. This library provides various DRL algorithms designed to interact with the Gym environment interface. Thus, the

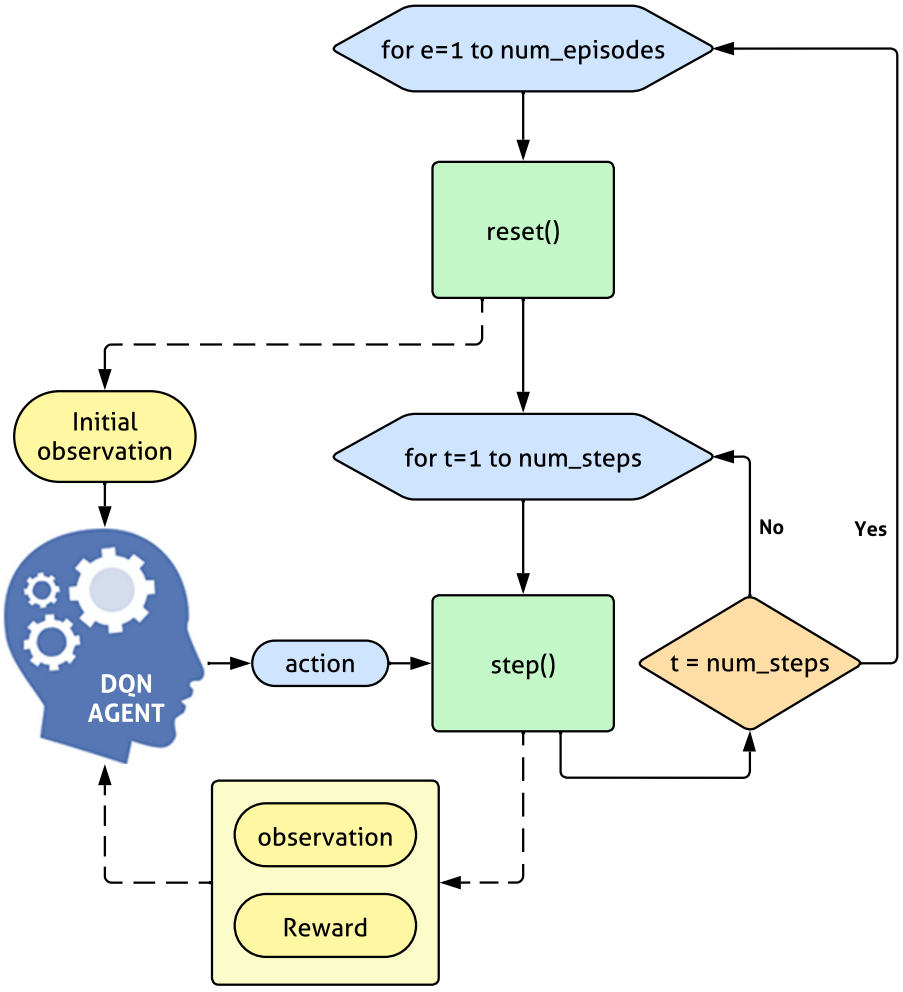


Fig. 4.11: Flowchart of agent training with reset() and step() methods.

library is responsible for internally implementing the algorithm described in Section 2.3.3.

Chapter 5

Experimental evaluation and Results

This chapter analyses the results obtained for the DQN agent developed in the project.

First, the experimental setup where the agent has been tested is presented. This involves the configuration of the network model and the parameters necessary for the operation of the agent. The tests have been carried out in two scenarios with a different number of Small Cell (SC) and UEs.

To find the optimal DQN agent, an experiment has been carried out in which the impact of different configuration parameters of the agent training is studied. Finally, we provide a step-by-step demonstration of the agent's operation across the two scenarios under various conditions. The actions are detailed alongside their effects on throughput and bandwidth allocation in each SC of the scenario.

5.1 Experimental setup

This section describes the experimental setup used to test the performance of the DQN agent developed in the project. The configuration is divided into parameters used in the RAN Simulator, which define the network model, and parameters used in the agent's configuration.

5.1.1 Network model configuration

The scenario occupies a geographical area of 100m x 100m, where a 5G NG-RAN as described in section 3.1.1 is deployed. The SCs operate in the

$3.4GHz$ band with a bandwidth of $100MHz$, corresponding to the bandwidth of commercial gNBs at $3.4GHz$. Numerology $\mu = 0$ is used, a subcarrier spacing of $15KHz$ and a bandwidth of $180KHz$ per PRB. Therefore, the available bandwidth is splitted into the following number of PRBs:

$$Number\ of\ PRBs = \frac{Total\ Bandwith}{Bandwith\ per\ PRB} = \frac{100MHz}{0.18MHz} \approx 555.5 \quad (5.1)$$

We work with 555 PRBs, resulting in a total bandwidth of 99.9MHz. The bandwidth allocation will be treated as an array of 555 positions. The spectrum is mapped to the PRB array, with each position in the array is represented by an index. Each index corresponds to one PRB and a bandwidth of 180kHz, as illustrated in Figure 5.1. For instance, if a cell has 200 PRBs assigned, starting at position 0 of the array, it will have the frequency range [3.4, 3.436] GHz, representing a bandwidth of 36 MHz.

The parameter values used in the RAN simulator were obtained in accordance with 3GPP recommendations as detailed in [36]. Table 5.1 shows the value of each parameter and its description, its influence on the network model is described in section 3.1.1. As shown in Table 5.1, the transmission power per PRB is specified as $P_R = 1.8$ mW = 2.55dBm. Figure 5.2 illustrates the total transmission power as a function of the number of PRBs allocated to the gNB, the value is presented in dBm. The maximum transmission power for the full bandwidth of 99.9 MHz is 30 dBm.

In the experimental tests carried out in this chapter, two different scenarios have been used:

- **Scenario A.** Composed of 2 SCs and 25 UEs as illustrated in Figure 5.3a.
- **Scenario B.** Composed of 4 SCs and 50 UEs as illustrated in Figure 5.3b.

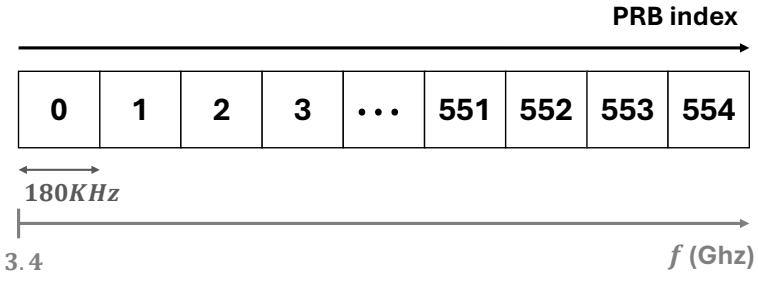


Fig. 5.1: Bandwidth as an array of 555 PRBs.

Parameter	Description	Value	Unit
L_0	Path loss value at a reference distance $d_0 = 1m$	22.38	dbm
n	Exponent of the variation with distance	43.3	-
b_z	Antenna height	4	meters
u_z	UEs height	1.5	meters
P_N	Noise power	-112.44	dBm
D_u	Service demand of users	5	Mbps
SE	Average spectral efficiency of UEs	2	bps/Hz
α	Attenuation factor	0.6	-
$SINR_{min}$	Minimum allowed value of SINR	0.1	-
$SINR_{max}$	Maximum allowed value of SINR	999.95	-
SE_{max}	Maximum achievable spectral efficiency with link adaptation	5.98	-
P_R	Transmission power per PRB	1.8	mW

Table 5.1: Values of the parameters used in the RAN simulator calculations.

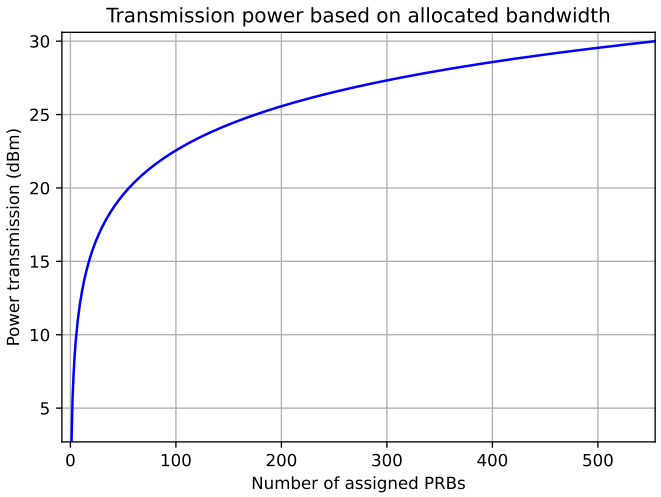
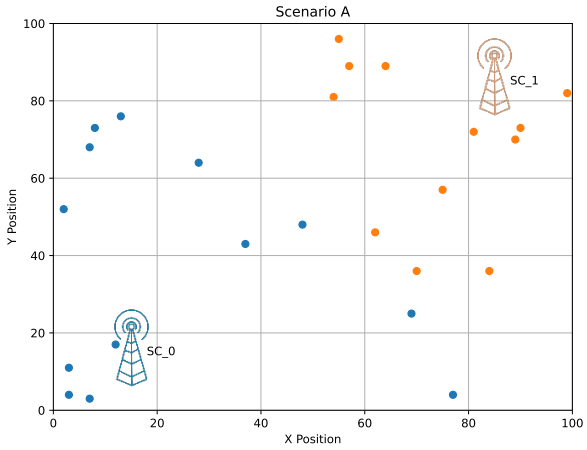


Fig. 5.2: Transmission power as a function of the number of PRBs allocated to the gNB.

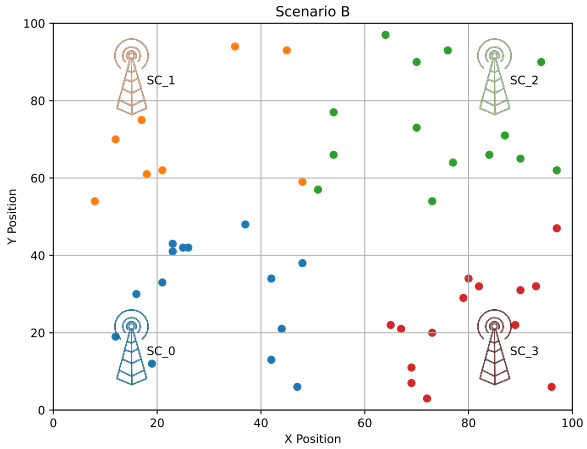
In both scenarios, all SCs are defined as neighbours. The UE colour indicates which is its serving SC, according to the colour of the SC.

5.1.2 Agent configuration

This section defines the configuration parameters that affect the DQN agent.



(a)



(b)

Fig. 5.3: Scenarios used in experimental tests

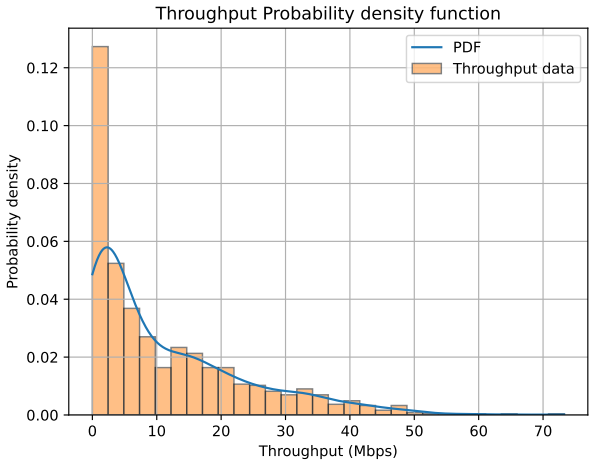
Optimization criterion

All SCs have a 5G slice deployed that meets the service requirements outlined in Section 3.2, as specified below:

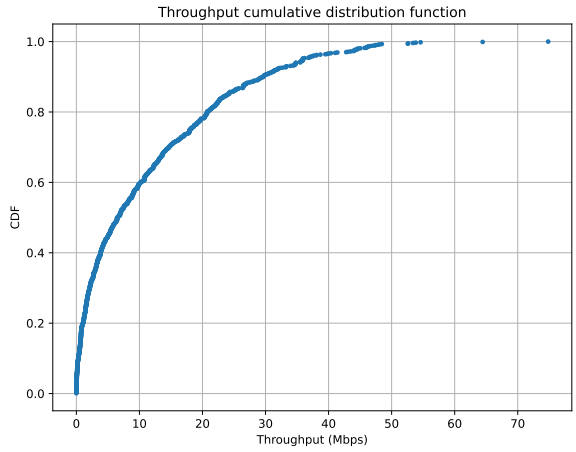
$$Minimize\ P_b^{TX} \quad s.t \quad T_u \geq T_{min} \tag{5.2}$$

where T_u is the throughput of the UE served by the SC with the worst SINR, and T_{min} is the minimum throughput allowed in the slice. To fulfil this service, the DQN agent developed is employed. It is responsible for controlling the bandwidth allocation for each SC within the slice.

Minimum throughput is a design requirement established by the operator or service provider to serve a particular application. In the experimental setup, T_{min} = 5Mbps is used, which is a typical value for eMBB type services. A study was conducted to analyse the throughput values experienced by the UEs across 500 runs of the Radio Access Network simulator. In each run, both the positions of the UEs and the bandwidth allocation for each SC were randomized. Probability Density Function (PDF) and Cumulative Distribution Function (CDF) of the throughput are presented in Figures 5.4a and 5.4b, respectively.



(a)



(b)

Fig. 5.4: Throughput PDF and CDF calculated over 500 executions of RAN simulator.

The values are concentrated in the range of 0 to 80 Mbps, with the 60% falling below 10 Mbps. Therefore, the minimum throughput value determined is consistent with the characteristics of the RAN simulator.

Action space

In Section 4.2, the actions available to the agent in the environment have been presented. Each action modifies a certain number p of PRBs. As presented, each gNB has a total available bandwidth of 99.9 MHz split into 555 PRBs. In the experimental tests carried out in this chapter, a number $p = 10$ PRBs is defined that the agent will modify with its actions. This allocation represents a bandwidth of 1.8 MHz, which aligns with PRB allocations used in real production environments. Table 5.2 shows the 7 actions that the agent can perform in this experimental setup.

ID	Description
0	Delete 10 PRBs from the left side
1	Delete 10 PRBs from the right side
2	Add 10 PRBs to the left side
3	Add 10 PRBs to the right side.
4	Shift PRBs 10 positions to the right.
5	Shift PRBs 10 positions to the left.
6	Not modify the PRBs.

Table 5.2: Action space in experimental setup

State space: Observation

As described in section 4.3, the observation that defines the state of the environment depends on the scenario configuration. First, the throughput value included in the observation is discretized. The benefits of discretization are shown in Appendix C. The throughput values fall within the range of $[0, 80]$ Mbps. However, certain areas within this range are more probable than others. Therefore, the discretization of this variable is not performed uniformly; instead, the number of bins used to divide the range varies accordingly. Let thr represent the throughput value to be discretized, and i denote the index of the discretized bin. The discretization function operates as follows:

$$i = \begin{cases} \left\lfloor \frac{\text{thr}}{0.5} \right\rfloor & \text{if thr} \leq 20 \text{ Mbps} \\ 40 + (\text{thr} - 20) & \text{if } 20 \text{ Mbps} < \text{thr} \leq 30 \text{ Mbps} \\ 50 + \frac{\text{thr}-30}{2} & \text{if } 30 \text{ Mbps} < \text{thr} \leq 40 \text{ Mbps} \\ 55 + \frac{\text{thr}-40}{5} & \text{if thr} > 40 \text{ Mbps} \end{cases} \tag{5.3}$$

where [.] is a function that approximates to the nearest integer. This discretization function results in 62 bins: 40 bins within the range $[0, 20]$ Mbps, 10 bins within the range $(20, 30]$ Mbps, 5 bins within the range $(30, 40]$ Mbps, and 7 bins within the range $(40, 80]$ Mbps. In this way, the throughput variable will have greater representativeness at lower values, according to its PDF and CDF, as shown in Figure 5.4.

The observation variables related to the neighbouring cells depend on the number of neighbours defined in the scenario. In Scenario A, there are 2 SCs, resulting in the number of neighbours being $N = 1$. In Scenario B, there are 4 SCs, all defined as neighbours, resulting in the number of neighbours being $N = 3$.

In the scenario A, the observation will be an array composed of the following 5 variables:

$$obs = [T_u, R, f_p, R_i, f_i] \tag{5.4}$$

On the contrary, the observation in scenario B will be composed of an array of 9 variables:

$$obs = [T_u, R, f_p, R_{i0}, R_{i1}, R_{i2}, f_{i0}, f_{i1}, f_{i2}] \tag{5.5}$$

All variables are discrete; Table 5.3 shows the number of possible values they can take. The state space grows, with the number of neighbours defined in the scenario. In the context of this project, neighbouring cells are those that, due to their location, can cause considerable interference. It is important to consider these cells as neighbours, so the agent can effectively reduce interference with them. The higher the dimensionality of the state space, the more exploration the agent will need during its training. This is because the agent must know a greater number of states to effectively learn all the options the environment offers. In the experiments carried out we will observe how the agent behaves in the face of the growth of the state space, as occurs in scenario B with respect to scenario A.

Variable	Abbreviation	range
UE throughput at the edge	thr	62
Number of PRBs assigned to the SC	r	556
PRB position flag	f_p	4
Number of interfered PRBs	r_i	556
Interference PRB position flag	f_i	3

Table 5.3: Range of values of the variables of the observation array

5.2 Agent performance study

In complex reinforcement learning problems, such as those addressed in this project, predicting the optimal values for each training parameter can be challenging. Therefore, it is essential to conduct multiple training runs and analyse the resulting behaviour to determine the most effective parameter settings.

5.2.1 DQN hyperparameters

As outlined in Section 4.5, the implementation of the DQN algorithm is carried out using the Python library stable_baselines3. This library internally implements the algorithm described in *Algorithm* 1 and allows agents to be trained in environments defined using the Gym interface, as is the case in our project.

To train the agent, the training hyperparameters must be configured. The specified hyperparameters are detailed below:

- **Discount Factor** γ. This parameter determines how much future rewards are valued compared to immediate rewards, as presented in equation 2.7 of the state of the art.
- **Learning Rate** α. This parameter determines the step size the algorithm takes to update the neural network weights θ during the minimization of the loss function.
- **Learning start step**. Number of steps before the agent starts learning.
- **Exploration fraction**. Fraction of training time during which the agent will explore.
- **Exploration initial** ϵ. Initial value of ϵ for the ϵ-greedy policy.
- **Exploration final** ϵ. Final value of ϵ for the ϵ-greedy policy.
- **Experience replay buffer size**. This parameter specifies the size of the replay buffer, which will store the k most recent experiences collected by the agent.
- **Batch size**. Number of experiences extracted from the replay buffer in each update of the neural network.

In addition to the training hyperparameters, there are two other fundamental factors that significantly impact the agent's performance. On one hand, the duration of the training is determined by the number of episodes. On

the other hand, the number of steps per episode is also crucial. This latter parameter is specific to the agent developed in this project, where the stopping condition within each episode is reaching the maximum allowable number of steps.

An ϵ-greedy action selection strategy is employed, the theoretical foundation of which is outlined in Section 2.3.3. The ϵ value will decrease throughout training according to the hyperparameter that defines the exploration time fraction. To obtain optimal performance and ensure that the agent learns the optimization objective, it is essential to achieve a balance between exploration and exploitation during training. The agent must sufficiently explore the state space of the environment to understand the various possibilities it offers, while also exploiting this knowledge to identify the optimal solutions in each situation.

This trade-off is primarily influenced by the training duration, the ϵ-greedy policy, and the size of the experience replay buffer. In each step of the training, the agent collects an experience that is stored in the experience replay buffer. Let us recall that we define experience as the tuple (a, s, s', r), where a represents the action, s is the initial state, s' is the resulting state, and r is the associated reward. During the exploration phase, the experiences collected by the agent are not necessarily those that provide the best reward. The objective of this phase is to gather information about all possible options, rather than focusing solely on those that offer the best immediate reward. Conversely, during the exploitation phase, actions are selected based on the maximum values of the Q-function, ensuring that the best action is chosen given the initial state s. Therefore, the experience replay buffer, from which experiences are extracted to update the weights of the Neural Networks (NN), will contain a combination of experiences from the exploration and exploitation phases. If the buffer predominantly contains experiences from the exploration phase, the agent will be familiar with a wide range of state-action pairs (s, a) but will fail to learn an optimal policy. Conversely, if the buffer contains significantly more exploitation experiences than exploration, the agent will learn the optimal solution for a limited range of state-action pairs, leaving it uncertain about what to do in other situations. For effective training, it is necessary to explore enough steps for the agent to become familiar with the entire state-action space. But it must also exploit enough steps to learn the optimal solution in most possible situations.

5.2.2 Training configurations

There is no standard configuration that works optimally for all DQN problems, so a study has been carried out with the aim of obtaining the best possible training configuration. To achieve this, the hyperparameters listed in Table 5.4 have been set and variations have been made to the number of steps per episode, the number of training episodes, and the exploration fraction.

Training hyperparameter	Value
Discount Factor γ	0.95
Learning Rate α	0.001
Learning start step	500
Exploration initial ϵ	1
Exploration final ϵ	0.05
Experience replay buffer size	1000000
Batch size	32

Table 5.4: Training hyperparameters value

A high discount factor γ is employed to prioritize long-term rewards, which is crucial given the complexity of our problem, as it ensures the agent considers future outcomes. Various γ values have been tested, it has been observed that high values lead the agent to better performance. The values chosen for the learning rate, experience replay buffer size, and batch size are the default values in DQN stable_baselines3. The ϵ-greedy policy begins to be applied from step 500 with an initial value $\epsilon = 1$ that is gradually reduced to $\epsilon = 0.05$ which is maintained until the end of training.

The agent has been trained over 1000, 2000, 3000, and 4000 episodes, with the number of steps per episode varied at 500, 700, and 1000. Additionally, exploration fractions of 30% and 70% have been employed in the training process. This combination results in 24 training configurations that are listed in Table 5.5. The training has been carried out in scenario A, with two SCs and 25 UEs as illustrated in Figure 5.3a.

5.2.3 Analysis of results

Once the 24 agents have been trained, with their respective configurations, as detailed in Table 5.5, we proceed to the inference phase to evaluate their performance. To achieve this, the 24 agents are subjected to 500 episodes, all starting from the same initial state. Each initial state is defined by the positions of the UEs, the allocation of bandwidth across the two SCs, and

Steps	Episodes	Exploration fraction (%)	Exploration steps	Total steps
500	1000	30	150000	500000
500	1000	70	350000	500000
500	2000	30	300000	1000000
500	2000	70	700000	1000000
500	3000	30	450000	1500000
500	3000	70	1050000	1500000
500	4000	30	600000	2000000
500	4000	70	1400000	2000000
700	1000	30	210000	700000
700	1000	70	490000	700000
700	2000	30	420000	1400000
700	2000	70	980000	1400000
700	3000	30	630000	2100000
700	3000	70	1470000	2100000
700	4000	30	840000	2800000
700	4000	70	1960000	2800000
1000	1000	30	300000	1000000
1000	1000	70	700000	1000000
1000	2000	30	600000	2000000
1000	2000	70	1400000	2000000
1000	3000	30	900000	3000000
1000	3000	70	2100000	3000000
1000	4000	30	1200000	4000000
1000	4000	70	2800000	4000000

Table 5.5: Value of the parameters in the 24 agent training configurations.

the specific SC controlled by the agent. The following metrics are used to evaluate agent performance:

- **Success rate.** We define the success rate as the percentage of episodes in which the agent achieves a throughput for the UE at the edge that exceeds the minimum threshold of $t_{min} = 5$Mbps. An episode is considered successful if the main constraint of the optimization objective, as established in Equation 3.11 of Chapter 3, is met.
- **Average value of the achieved throughput (Precision).** The precision of the agent is defined as the average value of the throughput achieved in the 500 episodes. According to the optimization criterion presented in Equation 3.12, the most optimal solution that the agent can achieve involves minimizing the assigned PRBs as much as possible, which directly translates into a reduction in throughput.

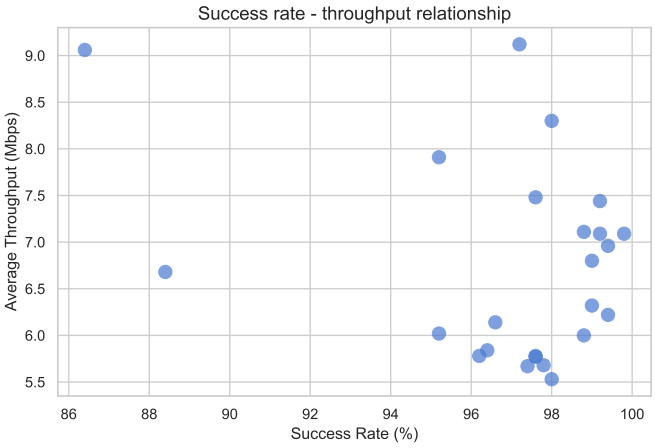
Therefore, the closer the achieved throughput is to the minimum allowed throughput, while still remaining above it, the better the agent's performance will be. Therefore, a throughput value very close to the minimum allowed implies high precision in the solution found, as it guarantees the minimum required throughput in the slice while also minimizing the transmission power as much as possible.

- **Interference-free episode rate**. This rate represents the percentage of episodes in which the bandwidth allocation of SC assigned by the agent is free from interference. As discussed in previous chapters, to minimize the allocated bandwidth and thereby reduce transmission power, it is essential to delete all possible interference. If this condition is met, the agent is able to find the optimal solution as per the optimization criterion specified in Equation 3.12. Therefore, this metric is a good indicator of the agent's performance.

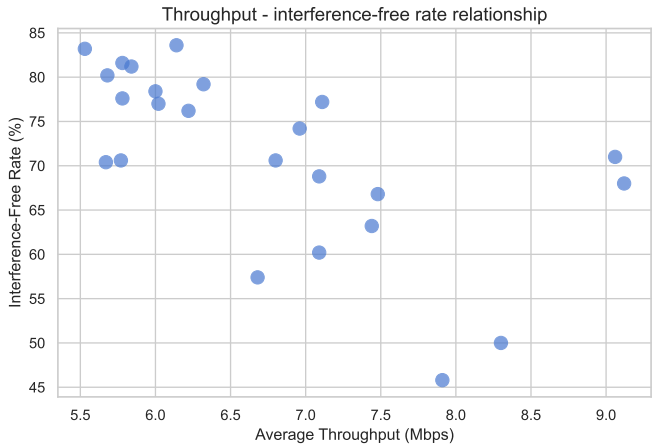
The best agent should have a high success rate, the lowest possible value of achieved throughput and a high rate of interference-free episodes. Table 5.6 presents the results obtained from the 500 episodes. The table displays the values of the three evaluated metrics alongside the agent's training configuration, which includes the number of steps per episode, the total number of episodes, and the exploration fraction. Agents with 500 steps per episode are highlighted in blue, those with 700 steps per episode are indicated in red, and agents with 1000 steps per episode are depicted in green.

Figure 5.5 shows scatter graphs between the 3 metrics analyzed to understand the relationship between them. Figure 5.5a shows the relationship between the success rate and the average throughput achieved. Throughput values closest to $T_{min} = 5$Mbps occur at success rates around 98%. As the success rate increases and approaches 100%, the throughput also increases, indicating a decrease in the agent's precision. There is a trade-off between these two metrics: an agent with a success rate nearing perfection will, on average, be less accurate. While it will always guarantee the minimum throughput in the slice, it will not minimize the transmission power to the greatest extent possible. Conversely, an agent that maximizes precision will occasionally fail, thereby reducing its success rate. That is, in attempting to reduce transmission power by decreasing the allocated bandwidth, the throughput may fall below the minimum allowed.

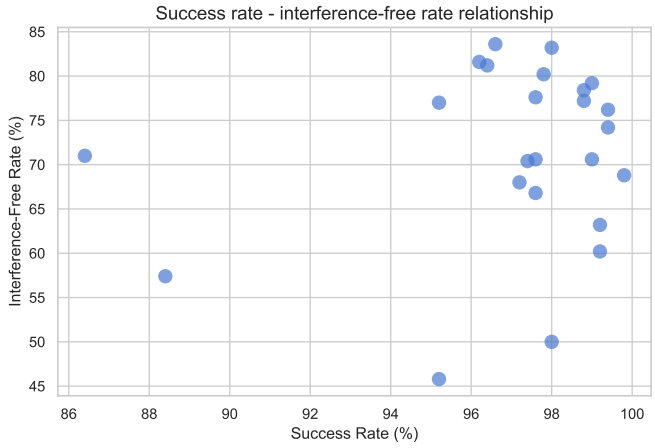
Figure 5.5b shows the relationship between the average throughput achieved and the interference-free rate. In this case, a certain correlation is observed between both metrics. Low values of throughput imply high values of the interference-free rate. This occurs because when the bandwidth allocation by the agent is free from interference with neighbouring cells, the throughput depends solely on the assigned PRBs. Consequently, it is easier for the agent



(a)



(b)



(c)

Fig. 5.5: Scatter plots between the 3 performance metrics.

Steps	Episodes	Exploration fraction (%)	Success rate (%)	Average throughput (Mbps)	Interference-free rate (%)
500	1000	30	95.2	7.91	45.8
500	1000	70	98.8	6	78.4
500	2000	30	95.2	6.02	77
500	2000	70	97.6	5.77	70.6
500	3000	30	99.2	7.09	60.2
500	3000	70	99.4	6.96	74.2
500	4000	30	99.2	7.44	63.2
500	4000	70	96.2	5.78	81.6
700	1000	30	97.2	9.12	68
700	1000	70	96.6	6.14	83.6
700	2000	30	99.4	6.22	76.2
700	2000	70	98.8	7.11	77.2
700	3000	30	99	6.32	79.2
700	3000	70	86.4	9.06	71
700	4000	30	97.8	5.68	80.2
700	4000	70	98	8.3	50
1000	1000	30	88.4	6.68	57.4
1000	1000	70	99	6.8	70.6
1000	2000	30	99.8	7.09	68.8
1000	2000	70	97.4	5.67	70.4
1000	3000	30	98	5.53	83.2
1000	3000	70	96.4	5.84	81.2
1000	4000	30	97.6	5.78	77.6
1000	4000	70	97.6	7.48	66.8

Table 5.6: Value of the metrics in the performance evaluation of the 24 training configurations.

to reduce the PRBs to find the optimal allocation that provides throughput as close as possible to the minimum. On the other hand, in the presence of interference, a reduction in allocated bandwidth can further decrease performance. Therefore, the agent tends to find solutions with higher throughput values, as reducing the assigned PRBs further at that point could result in throughput falling below the minimum allowed for the slice.

Figure 5.5c shows the relationship between the success rate and the interference-free rate. In this graph, we can observe the same behaviour that occurred between success rate and throughput, due to the correlation between throughput and interference. At success rates close to 100%, the percentage of episodes without interference is slightly lower compared to

that achieved with success rates around 98%.

Figures 5.6, 5.7, and 5.8 illustrate the success rate and average throughput for agents with 500, 700, and 1000 steps per episode, respectively. This bar graph represents the success rate value in blue and the average throughput value in green. Below each pair of bars, the number of episodes and the exploration fraction are shown. The left side of the y-axis displays the scale for the success rate percentage, while the right side of the y-axis shows the scale for throughput in Mbps. The minimum throughput is marked with a dashed horizontal red line.

The best agent will be the one that has achieved the best trade-off between success rate and average throughput, that is, the compromise between success and precision. In general, most agents achieve a high success rate. However, there is considerable variation in their average throughput. The throughput values closest to the minimum allowed are observed when using 1000 steps per episode. This is expected, as a higher number of steps per episode allows for greater exploration within the same episode. This enables the agent to gain a deeper understanding of the action-state space, thereby enhancing its ability to identify optimal solutions across a broader range of states. The number of episodes will influence how much knowledge the agent acquires of the state-action space, since each episode begins from a randomly selected initial state. Training an agent with an increasing number of episodes does not necessarily guarantee improved performance. It is crucial to find the optimal balance between the duration of training and

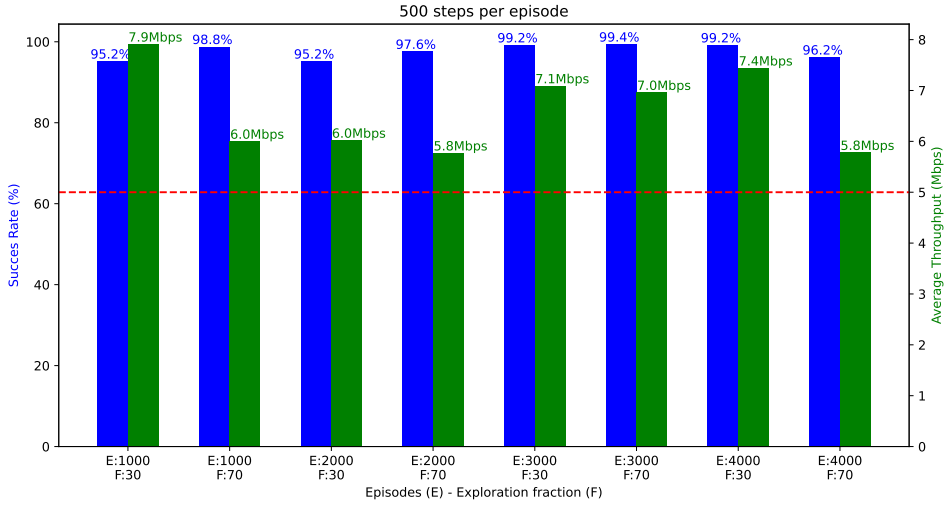


Fig. 5.6: Success Rate and average throughput results for 500 steps per episode.

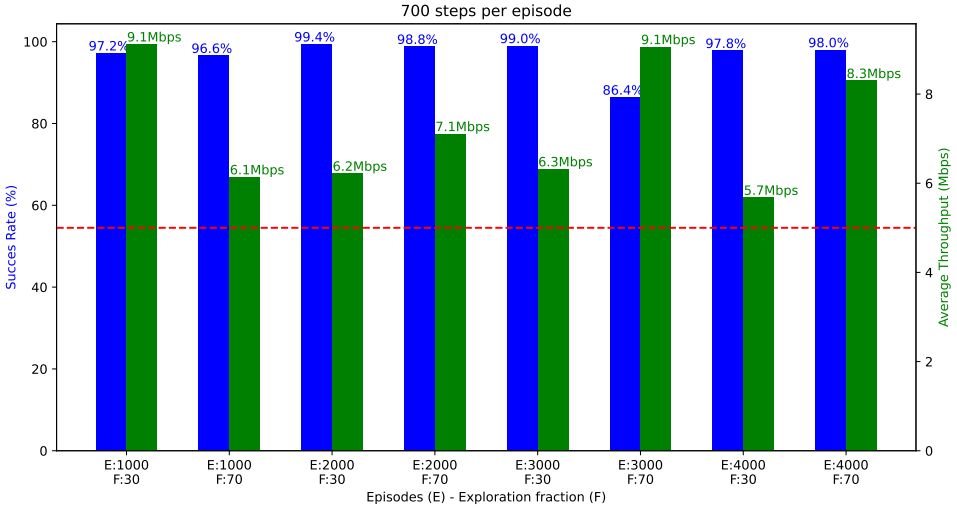


Fig. 5.7: Success Rate and average throughput results for 700 steps per episode.

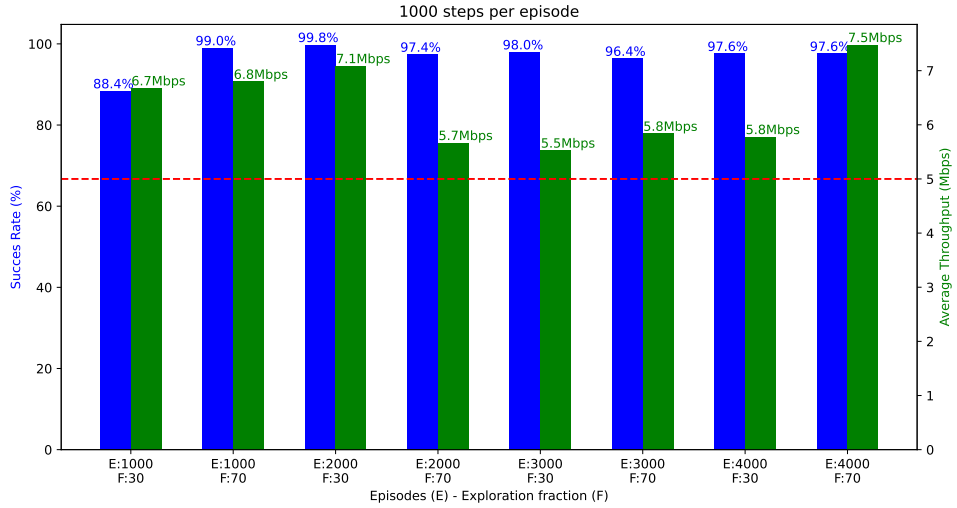


Fig. 5.8: Success Rate and average throughput results for 1000 steps per episode.

the trade-off between exploration and exploitation. The parameter that will indicate this trade-off, in conjunction with the number of episodes, is the exploration fraction. Values of 30% and 70% have been utilized in the experiment. This parameter determines the percentage of training steps during which the ϵ exploration probability is reduced from 1 to 0.05, as specified in Table 5.4. A higher percentage indicates increased agent exploration relative to exploitation.

In the three values of steps per episode, the best performance is observed at 2000 or 3000 episodes, suggesting that this is an effective training length. Among these configurations, the best-performing agent was achieved with 1000 steps per episode, 3000 episodes, and 30% of exploration fraction. This agent has achieved a 98% success rate, an average throughput of 5.5Mbps and an interference-free rate of 83% (these values can be found in table 5.6).

It should be noted that most training configurations yield quite acceptable results, with slight differences that allow us to select the optimal solution. This good performance, regardless of the hyperparameter settings, indicates that the agent design is well aligned with the optimization objective. The action space, state space, and reward algorithm are responding appropriately. In the following sections, particular examples will be presented showing the operation of the agent step by step.

From this experiment, it can be concluded that the number of steps per episode is crucial for enhancing the agent's accuracy across a broader range of states. However, this increase in steps must be paired with an appropriate training duration and a balanced trade-off between exploration and exploitation.

5.3 Agent Operation

This section shows the agent's operation step by step, that is, the actions it takes in each situation and their effect on the environment.

To achieve this, various specific situations are analysed within the two scenarios presented in Figure 5.3. During training, the agent operates on a single SC while the bandwidth allocation of the remaining SCs remains fixed. However, in these tests, the agent acts on all SCs in the scenario at each step. The agent is agnostic of the SC's position in the scenario. It only needs the observation describing the state of an SC to take action on its bandwidth allocation.

In each example, we will begin with an initial situation defined by the position of the UEs in the scenario and the bandwidth assigned to the SCs. Each SC will be described by its observation array, which will be provided to the agent. Based on this observation, the agent will recommend the optimal action for each SC to achieve the optimization objective.

5.3.1 Scenario A

Scenario A is composed of two small cells and 25 UEs as shown in Figure 5.3a. Below are 3 examples that show the agent's operation in this scenario

for 3 different initial situations. These demonstrations employ the agent that delivered the best performance in the study from the previous section, which was configured with 1,000 steps per episode, 3,000 episodes, and a 30% of exploration fraction.

Example 1A

In this example, SC_0 is assigned a bandwidth of 7.2 MHz (40 PRBs) and SC_1 is assigned a bandwidth of 43.2 MHz (240 PRBs). There is no interference between the PRBs assigned to each cell. The location of the UEs and their serving cell are shown in Figure 5.9. The throughput of the UEs with

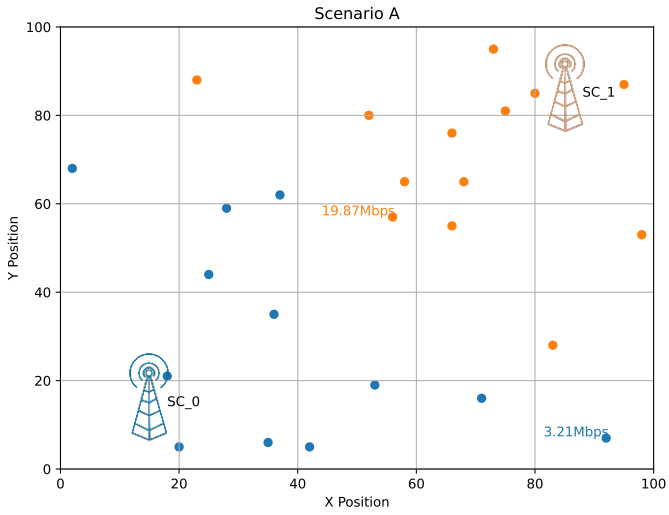
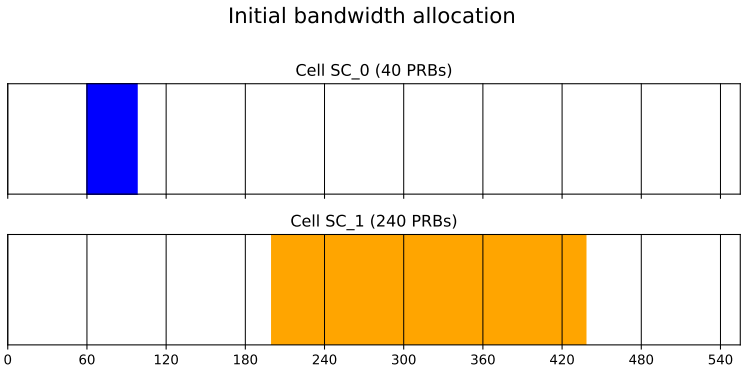


Fig. 5.9: UEs position and throughput of the worst users in each SC in Example 1A.

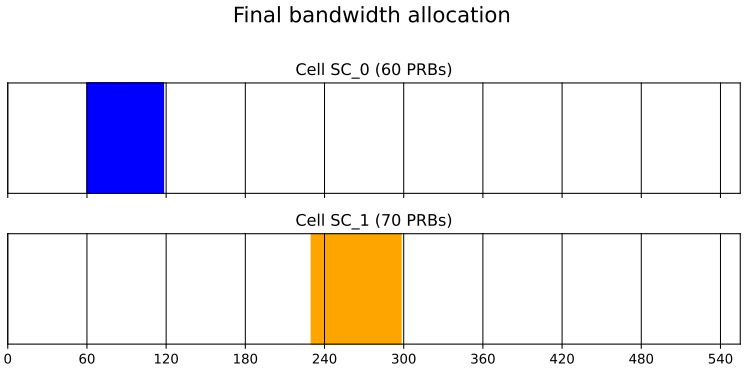
the worst SINR in each SC is also displayed. The UE at the edge of SC_0 has a throughput of 3.21Mbps while the user at the edge of SC_1 has a value of 19.87Mbps.

Figure 5.10 shows the initial bandwidth allocation, and the final bandwidth allocation after the agent operation. The agent increased the bandwidth of SC_0 by 3.6 MHz (20 PRBs) and decreased the bandwidth of SC_1 by 30.6 MHz (170 PRBs).

To gain a deeper understanding of the agent's decisions, Figure 5.11 shows the evolution of throughput, the number of assigned PRBs, and the number of interfered PRBs. The grid in these graphs is colour-coded to represent the actions taken by the agent at each step, with the corresponding



(a)



(b)

Fig. 5.10: Initial and final bandwidth allocation in Example 1A.

colour legend displayed at the bottom of Figure 5.11. The two graphs at the top of Figure 5.11 illustrate the evolution of throughput for each SC. The minimum allowed throughput for the slice, $T_{min} = 5$Mbps, is indicated by a red dotted line. The bottom left figure shows the number of PRBs assigned to each SC, while the bottom right figure shows the number of interfered PRBs in each SC. In all the representations in Figure 5.11, the x-axis values displayed in the lower graph are shared between the two SCs. These representations will be used throughout all the examples in this section, aiding in our understanding of how the agent behaves to achieve the optimization goal.

The UE at the edge of the cell SC_0 has a throughput lower than the minimum allowed in the slice. The agent decides to increase the PRBs on the right side of the assignment. In two steps, taking this action increases the number of PRBs by 20, achieving a throughput of 5.13 Mbps as shown in Figure 5.11. The situation for cell SC_1 is different. This cell has a higher allocated bandwidth, resulting in the user with the lowest SINR achieving

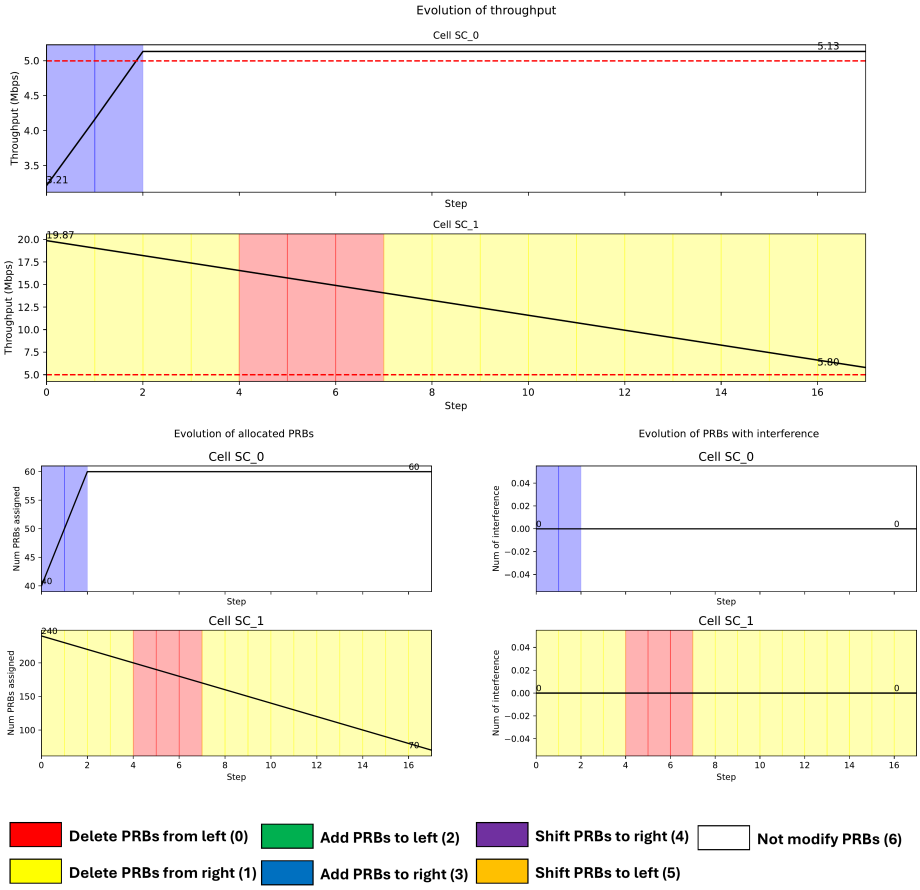


Fig. 5.11: Example 1A: Throughput, number of assigned PRBs and number of interfered PRBs along with the action taken at each step.

a throughput of 18.29 Mbps. In line with the optimization objective, SC_1 has significant capacity to reduce its bandwidth, which would allow for a reduction in transmission power. The agent decides to remove PRBs from the left and right side of the allocation until the throughput reaches 5.89 Mbps. When the agent finds what it considers an optimal solution, it decides to take the action that does not modify the PRBs. In Figure 5.11, this action is represented by the white color code.

The agent required 2 steps to optimize SC_0 and 16 steps to optimize SC_1. In this situation, where there is no interference between cells, the situation is the simplest for the agent. It only needs to assess whether the throughput is above or below the minimum threshold to determine the appropriate action. In this example, the throughput is optimized to be as close to the minimum as possible, ensuring that the allocated bandwidth—and

consequently the transmission power—cannot be reduced further.

Summarizing the agent's operation in this example:

- SC_0. It increases its bandwidth by 3.6 MHz (20 PRBs), resulting in a corresponding rise in transmission power of 15.56 dBm. This adjustment improves the UE throughput at the edge from 3.21 Mbps to 5.13 Mbps, meeting the minimum allowed throughput for the slice.

- SC_1. It reduces its bandwidth by 30.6 MHz (170 PRBs), leading to a decrease in transmission power of 24.85 dBm. As a result, the UE throughput at the edge decreases from 19.87 Mbps to 5.80 Mbps, while still meeting the minimum allowed throughput and optimizing transmission power to the greatest extent.

Example 2A

This example is a bit more complex than the previous one. SC_0 is assigned a bandwidth of 21.6 MHz (120 PRBs) and SC_1 is assigned a bandwidth of 25.2 MHz (140 PRBs). There are 12.6 MHz (70 PRBs) of interference between both cells. The location of the UEs and their serving cell are shown in Figure 5.12. The UE at the edge of SC_0 has a throughput of 6.82Mbps while the user at the edge of SC_1 has a value of 5.38Mbps.

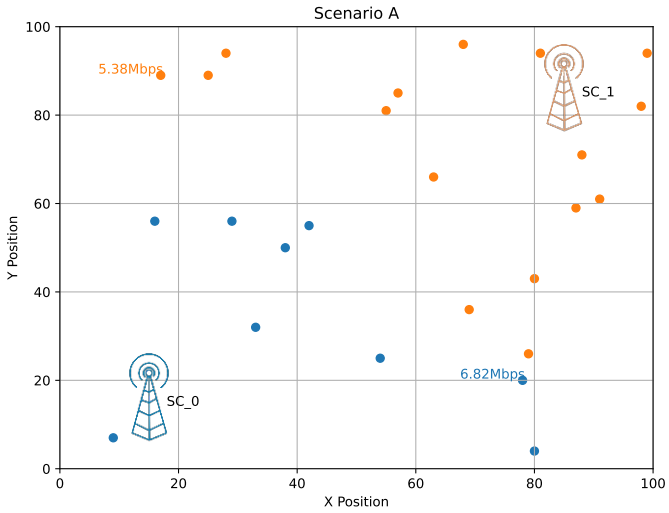
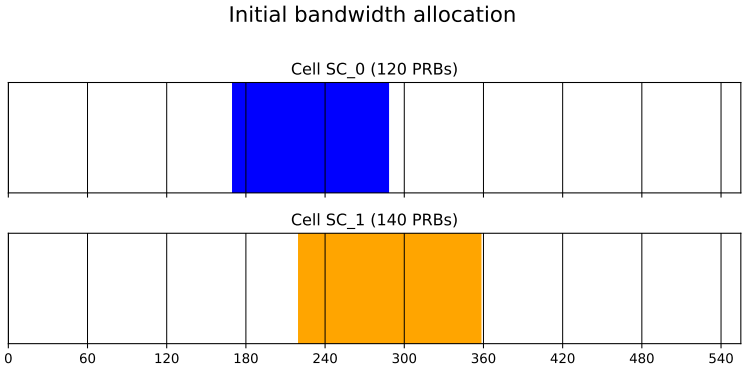
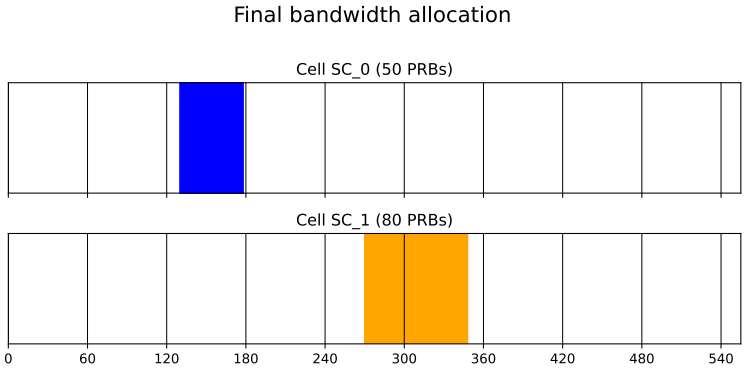


Fig. 5.12: UEs position and throughput of the worst users in each SC in Example 2A.

Figure 5.13 shows the initial bandwidth allocation, and the final bandwidth allocation after the agent operation. The agent decreased the bandwidth of SC_0 by 12.6 MHz (70 PRBs) and decreased the bandwidth of SC_1 by 10.8 MHz (60 PRBs). Additionally, the agent successfully eliminates all interfered bandwidth between the two cells.



(a)



(b)

Fig. 5.13: Initial and final bandwidth allocation in Example 2A.

Figure 5.14 shows the evolution of throughput, the number of assigned PRBs, and the number of interfered PRBs in example 2A. In this example, the throughput for both cells initially exceeds the minimum allowed in the slice. Consequently, the agent's objective is to reduce the bandwidth to lower the transmission power. The agent takes actions to shift the allocated bandwidth in order to eliminate interference. Specifically, it adjusts the allocation of SC_0 to the left and SC_1 to the right, shifting 10 PRBs in each step. As shown in Figure 5.13a, the most efficient way to eliminate interference is to shift the bandwidth allocated to each SC in the specified directions.

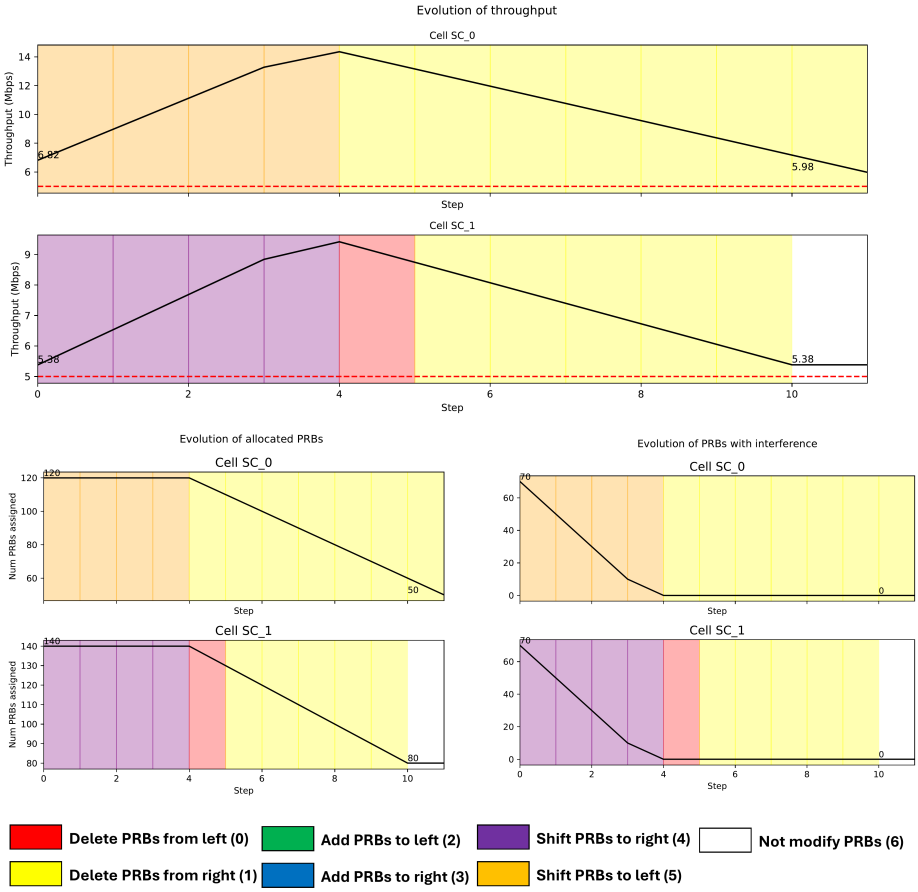


Fig. 5.14: Example 2A: Throughput, number of assigned PRBs and number of interfered PRBs along with the action taken at each step.

The number of PRBs with interference at each step is displayed in the graph on the right side of Figure 5.14. The agent requires 4 steps to eliminate the 70 interfered PRBs, as it removes 20 PRBs per step (10 from each cell). As interference is reduced, throughput increases. This concept has been discussed in several chapters and is the basis of the reward algorithm designed.

To maximize the reduction in allocated bandwidth while ensuring the throughput criterion, minimizing interference is essential. Maximum throughput is achieved by minimizing interference as much as possible. Once maximum throughput is attained, bandwidth can be gradually reduced until ideally reaching the threshold $thr = thr_{min}$. This behaviour is exactly how the agent is acting in this example. First, the agent completely deletes interference. Once that is accomplished, it then reduces the number of assigned

PRBs to achieve the minimum bandwidth necessary to ensure throughput as close to the minimum allowed as possible.

Summarizing the agent's operation in this example:

- SC_0. It reduces its bandwidth by 12.6 MHz (70 PRBs), resulting in a corresponding decrease in transmission power of 21.19 dBm. This adjustment reduces the UE throughput at the edge from 6.82 Mbps to 5.98 Mbps. The agent has assigned the PRBs such that the interference has been completely reduced.
- SC_1. It reduces its bandwidth by 10.8 MHz (60 PRBs), resulting in a corresponding decrease in transmission power of 20.52 dBm. In this case, the same throughput value is achieved as initially, but with reduced bandwidth and transmission power, highlighting the benefits of completely eliminating interference.

Example 3A

In the final example of scenario A, we analyze a situation where the bandwidth of one SC is entirely interfered with by the other SC. SC_0 is assigned a bandwidth of 41.4 MHz (230 PRBs) and SC_1 is assigned a bandwidth of 73.8 MHz (410 PRBs). All PRBs assigned to SC_0 are interfered by the assignment in SC_1, which utilizes the majority of the available spectrum. The location of the UEs and their serving cell are shown in Figure 5.15. The UE at the edge of SC_0 has a throughput of 3.95Mbps while the user at the edge of SC_1 has a value of 21.19Mbps.

Figure 5.16 shows the initial bandwidth allocation, and the final bandwidth allocation after the agent operation. The agent decreased the bandwidth of SC_0 by 28.8 MHz (160 PRBs) and decreased the bandwidth of SC_1 by 63 MHz (250 PRBs). Additionally, the agent successfully eliminates all interfered bandwidth between the two cells.

Figure 5.17 shows the evolution of throughput, the number of assigned PRBs, and the number of interfered PRBs in example 3A. In this example, the situations for each cell are different. On the one hand, SC_0 does not meet the minimum throughput allowed in the slice. On the other hand, SC_1 exceeds this threshold by a substantial margin and has been allocated more bandwidth than necessary based on the slice requirements.

Given the initial bandwidth allocation depicted in Figure 5.16a, the most effective strategy for the agent to enhance the throughput for users served by SC_0 is to increase the bandwidth. As a consequence, it takes the action of adding PRBs to the right side of the assignment. In the case of SC_1, as

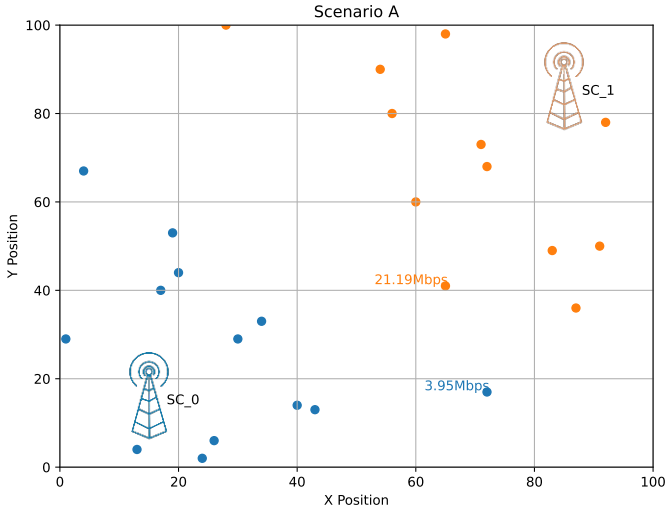


Fig. 5.15: UEs position and throughput of the worst users in each SC in Example 3A.

shown in Figure 5.16a, shifting the allocated bandwidth will not mitigate the interference. Of the 555 available PRBs, SC_1 has 410 assigned. Regardless of its position in the spectrum, SC_1 will consistently experience interference from the 230 PRBs assigned to SC_0. The agent has learned during its training that, in such cases, the optimal strategy is to reduce bandwidth by deleting PRBs. This approach is applied to SC_1, as illustrated in Figure 5.17. The agent decides to strategically remove the PRBs from the right side. As can be seen in Figure 5.16a, the agent needs fewer steps to start removing interfering PRBs if it removes the PRBs on the right side. This behavior illustrates how the "PRB Interference Position Flag" (section 4.3.5), included in the observation, allows the agent to more effectively position itself in the bandwidth and identify which side of the allocation is most advantageous for intervention.

When the throughput in SC_0 reaches the minimum, the situation changes for the agent. At this point, the objective shifts to minimizing transmission power by reducing the bandwidth. The agent chooses to shift the PRBs assigned to SC_0 to the right side. As observed in Figure 5.17, this action leads to a reduction in interfered PRBs and an increase in throughput. These actions on SC_0 affect the observation describing the state of SC_1. Thus, the agent realizes that if SC_0 is shifting its bandwidth to the right, then shifting SC_1's bandwidth to the left will help reduce the number of interfered PRBs for both cells. The agent shifts the bandwidth of SC_1 to the left while also

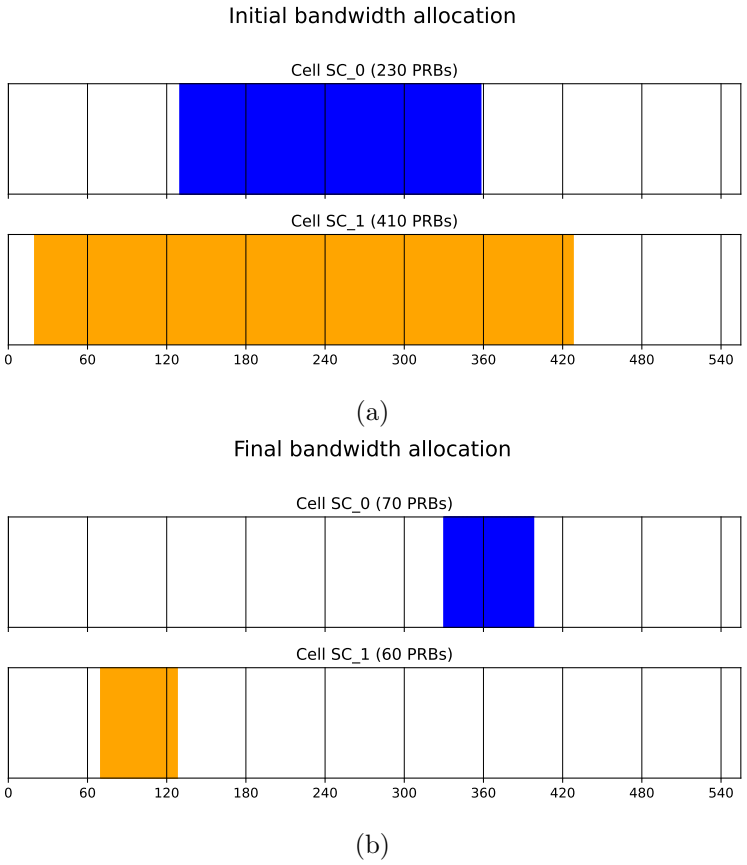


Fig. 5.16: Initial and final bandwidth allocation in Example 3A.

reducing PRBs on that side, which lowers the throughput. Simultaneously, the agent shifts the bandwidth of SC_0 to the right until the interference is completely eliminated.

Once there are no interfering PRBs, as shown in the lower right graph of Figure 5.17, the agent takes the actions of deleting PRBs on both sides in both cells until it achieves the optimal throughput, very close to 5 Mbps. This example demonstrates how the agent is continuously aware of the state of the SC and is capable of taking long-term actions to ensure finding the optimal solution in accordance with the slice requirements.

Summarizing the agent's operation in this example:

- SC_0. It reduces its bandwidth by 28.8 MHz (160 PRBs), resulting in a corresponding decrease in transmission power of 24.59 dBm. The agent has completely reduced interference, allowing it to increase throughput from 3.95 to 5.80Mbps with 160 fewer PRBs.

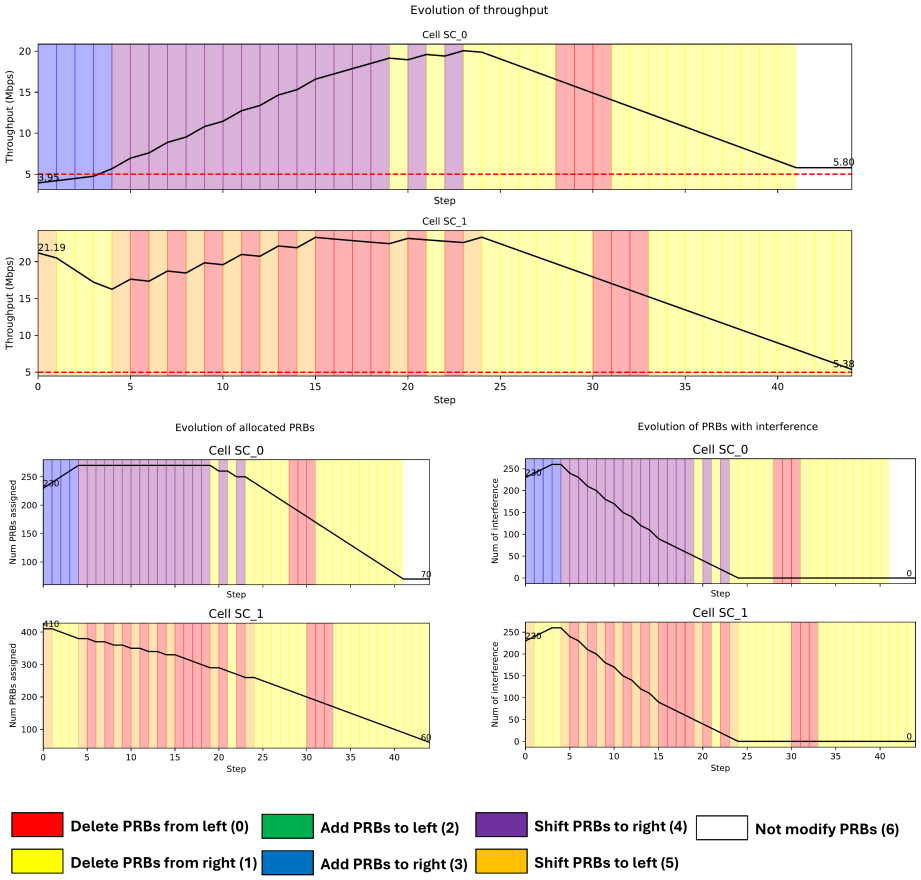


Fig. 5.17: Example 3A: Throughput, number of assigned PRBs and number of interfered PRBs along with the action taken at each step.

- SC_1. It reduces its bandwidth by 63 MHz (250 PRBs), resulting in a corresponding decrease in transmission power of 26.53 dBm. The UE throughput at the edge of SC_1 has been reduced from 21.19 Mbps to 5.38 Mbps.

5.3.2 Scenario B

Scenario B is composed of four small cells and 50 UEs as shown in Figure 5.3b. Below is an example showing how the agent works in this scenario. This example use an agent, trained in scenario B, with the agent configuration that offered the best performance: 1000 steps per episode, 3000 episodes, and a 30% exploration fraction.

An important factor to consider in scenario B, as compared to scenario A, is that while the bandwidth remains the same, the number of SCs

and UEs is doubled from the previous scenario. As a result, the available throughput tends to decrease due to the increased number of users (Equation 3.10), and the likelihood of interference rises because of the additional cells. Consequently, it becomes more challenging to find solutions close to $T_{min} = 5$Mbps, unlike in scenario A.

Example 1B

In this example, SC_0 is allocated a bandwidth of 19.8 MHz (corresponding to 100 PRBs), SC_1 is allocated 27 MHz (150 PRBs), SC_2 is allocated 21.6 MHz (120 PRBs), and SC_3 is allocated 43.2 MHz (240 PRBs). There are multiple interferences among the cells of the scenario, as illustrated in Figure 5.19a. The location of the UEs and their serving cell are shown in Figure 5.18. The throughput of the UEs with the worst SINR in each SC is also displayed. SC_0 and SC_2 have throughput below the minimum required for the slice, with SC_0 significantly underperforming and SC_2 nearly meeting the minimum threshold. Conversely, UEs at the edge of SC_1 and SC_3 experience significantly higher throughput. Therefore, the agent will follow a different strategy in each cell to ensure that they meet the slice requirements.

Figure 5.19a shows the initial bandwidth allocation, and Figure 5.19b shows the resulting allocation after the agent operation. As observed, the agent has reduced the bandwidth allocated to all SCs and completely deleted interference. To understand how this outcome was achieved, the agent's

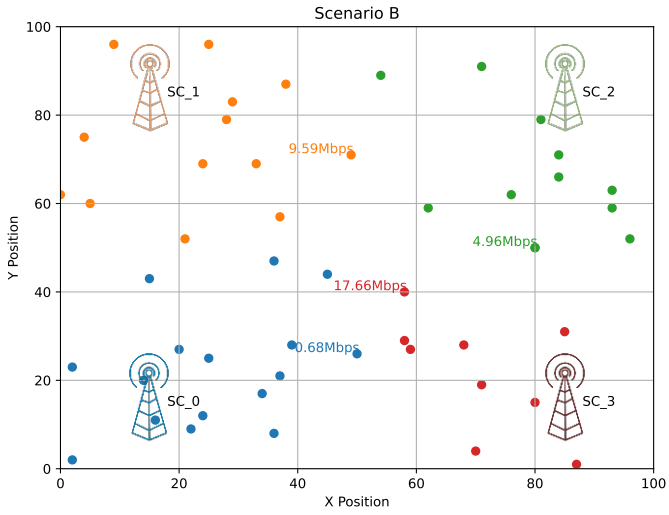


Fig. 5.18: UEs position and throughput of the worst users in each SC in Example 1B.

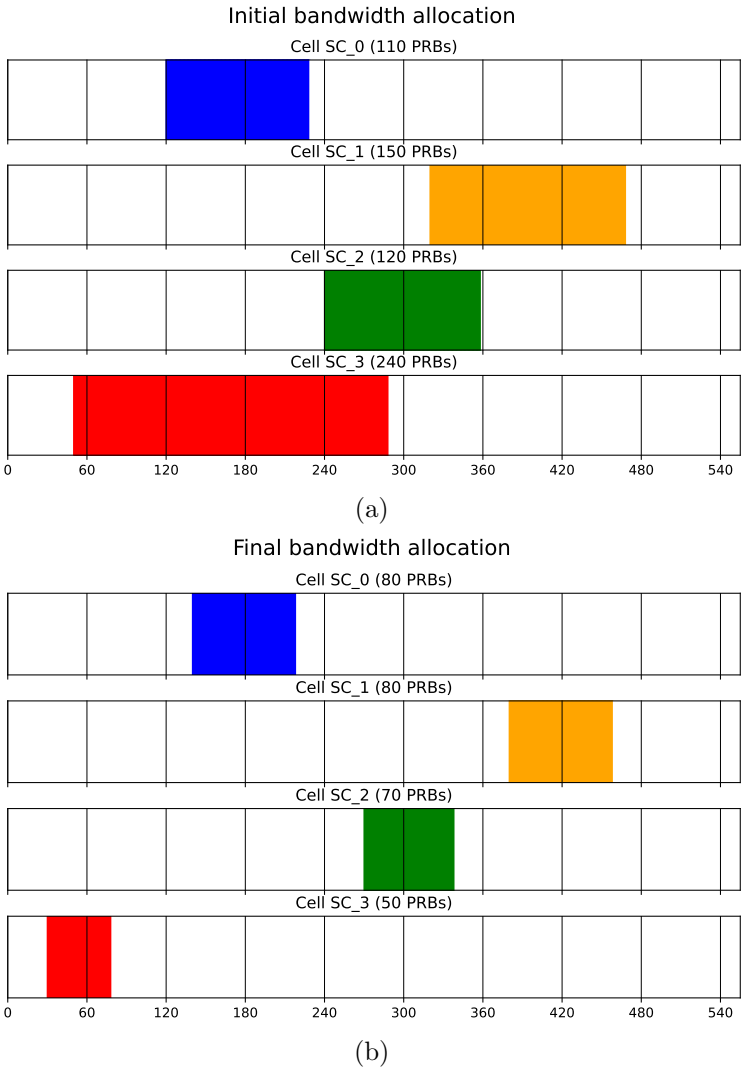


Fig. 5.19: Initial and final bandwidth allocation in Example 1B.

actions are analysed step by step. Figures 5.20, 5.21, and 5.22 show the evolution of throughput, the number of assigned PRBs, and the number of PRBs with interference, respectively. Similar to the graphs presented in the previous scenario, the background of the grid is colour-coded to represent the actions taken by the agent at each step.

At each step, the agent adjusts the bandwidth of all cells, except when it opts not to make any modifications. Modifying the bandwidth of one SC can impact the throughput experienced by UEs in other SCs, as it may increase or decrease the interference it causes in neighbouring cells. Therefore, to accurately analyse agent behaviour based on the representations in Figures

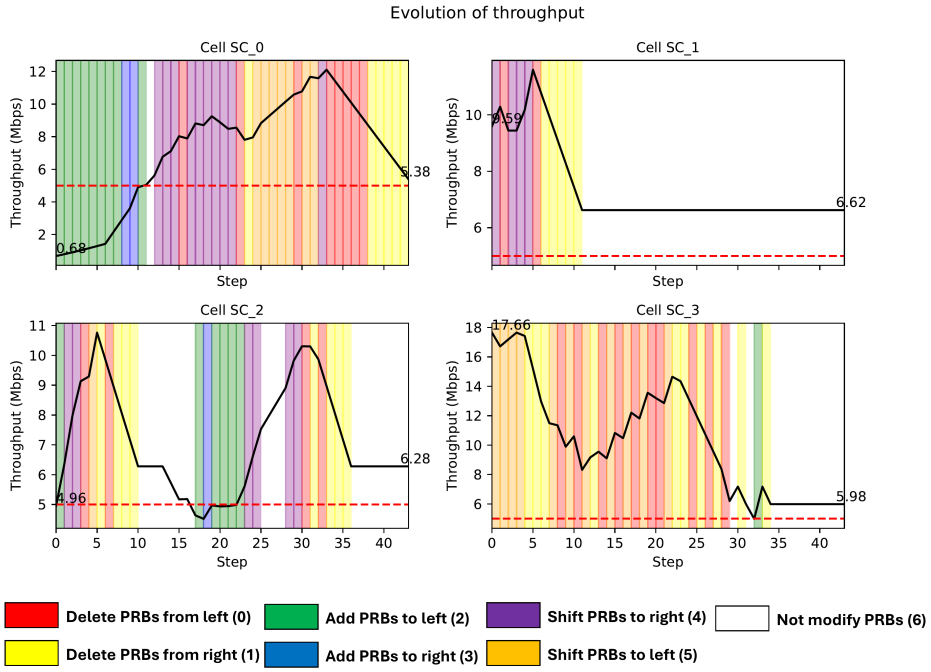


Fig. 5.20: Example 1B: Evolution of throughput along with the action taken by the agent at each step.

5.20, 5.21, and 5.22, it is essential to consider the actions taken in all SCs and not analyse a single SC individually.

The UE at the edge of SC_0 has an initial throughput of 0.68Mbps, so the agent decides to add PRBs to increase this value. In SC_1, the throughput is 9.59 Mbps, which is above the minimum allowed in the slice. This cell experiences interference in the left area of the spectrum caused by SC_2, as shown in Figure 5.19a. To eliminate this interference, the agent shifts the allocated bandwidth to the right and removes PRBs from the interfered zone on the left side. Once the interference is resolved, the agent reduces PRBs on the right side until the throughput reaches 6.62 Mbps, at which point the agent determines that further bandwidth modification is unnecessary.

The situation in SC_2 is somewhat more complex. Initially, the UE throughput at the edge is 4.96 Mbps, which is just below the minimum threshold of $T_{min} = 5$Mbps. The agent achieves this threshold with a simple increase of 10 PRBs. SC_2 has interference both on the right side of the spectrum with SC_1 and on the left side with SC_3 as seen in Figure 5.19a. The agent chooses the quickest method to reduce interference by shifting the bandwidth allocation to the right. This adjustment is advantageous because

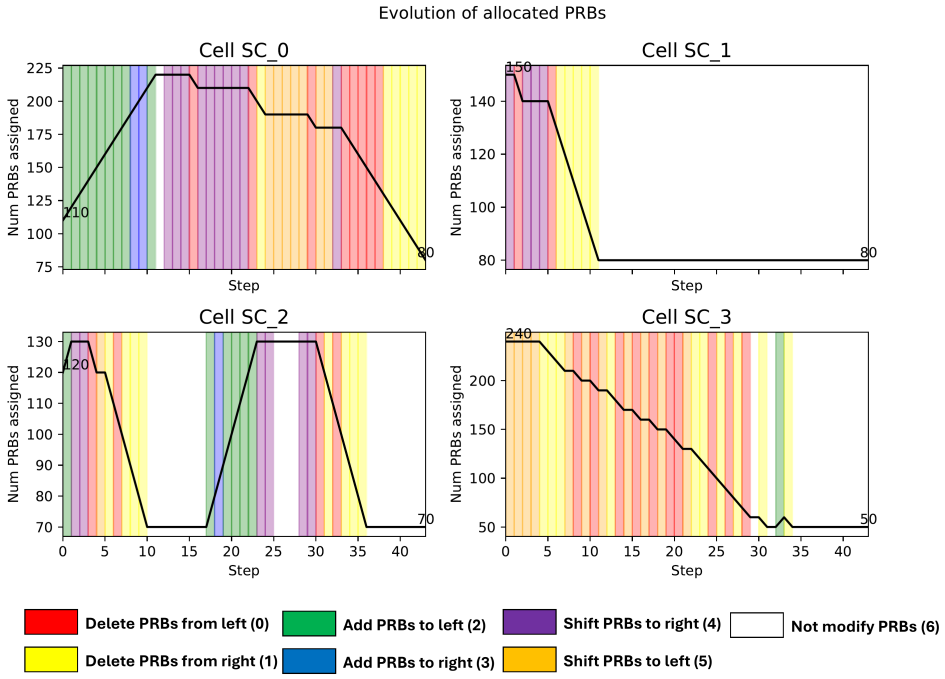


Fig. 5.21: Example 1B: Evolution of the number of PRBs assigned along with the action taken by the agent at each step.

SC_1 also moves in the same direction and removes its PRBs on that side, allowing both SCs to collectively eliminate their mutual interference.

The UE at the edge of SC_3 experiences the highest throughput due to the cell having the largest allocated bandwidth among the four. To manage this, the agent shifts the bandwidth allocation to the left and removes PRBs from the right side. This adjustment reduces both interference and throughput, ultimately resulting in a throughput of 5.98 Mbps.

Once the UE throughput at the edge of SC_0 exceeds the minimum allowed in the slice, the agent begins to address the interference. To achieve this, the agent shifts the allocated bandwidth to the right. SC_0 reduces its interference with SC_3 but begins to experience interference with SC_2, leading to a decrease in the throughput of this cell. This interference is significant enough that the UE throughput at the edge of SC_2 falls below the minimum threshold of 5 Mbps. The agent, having previously achieved a stable situation in SC_2 with no further bandwidth adjustments, observes the drop in throughput and decides to increase the PRBs to ensure the minimum required throughput is maintained.

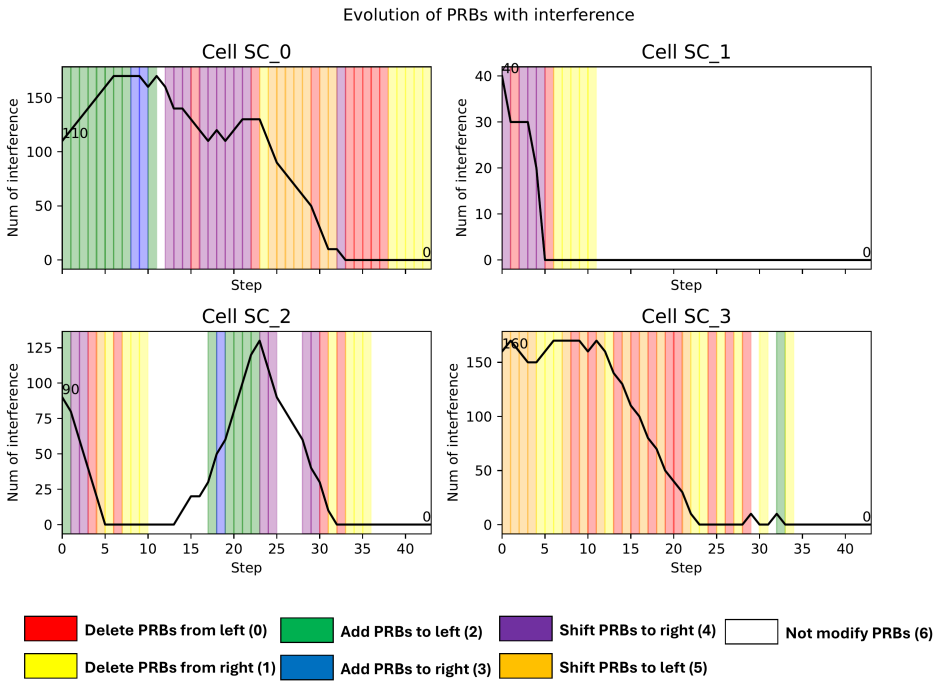


Fig. 5.22: Example 1B: Evolution of the number of PRBs with interference along with the action taken by the agent at each step.

In this example, we observe how the agent's actions on one SC impact the other SCs it either interferes with or stops interfering with. Consequently, an SC that the agent determined required no modifications may now need adjustments in bandwidth allocation to achieve an optimal solution once again. This behaviour demonstrates the agent's ability to continually ensure that all cells in the scenario meet the optimization requirements defined in the slice.

Finally, the agent achieves a stable situation for all SCs within 40 steps. The final throughput values are not as close to $T_{min} = 5$ Mbps as observed in scenario A, because this scenario has twice as many users. Since throughput is governed by Equation 3.10, a higher number of users per SC results in greater throughput degradation as the number of assigned PRBs decreases.

Summarizing the agent's operation in this example:

- SC_0. It reduces its bandwidth by 5.4 MHz (30 PRBs), resulting in a corresponding decrease in transmission power of 17.32 dBm. The agent has completely reduced interference, allowing it to increase throughput from 0.68 to 5.38 Mbps with 30 fewer PRBs.

- SC_1. It reduces its bandwidth by 12.6 MHz (70 PRBs), resulting in a corresponding decrease in transmission power of 21.03 dBm. The UE throughput at the edge of SC_1 has been reduced from 9.59 Mbps to 6.62 Mbps.
- SC_2. It reduces its bandwidth by 9 MHz (50 PRBs), resulting in a corresponding decrease in transmission power of 19.54 dBm. The UE throughput at the edge of SC_2 has been increased from 4.96 Mbps to 6.28 Mbps with 50 fewer PRBs.
- SC_3. It reduces its bandwidth by 34.2 MHz (190 PRBs), resulting in a corresponding decrease in transmission power of 25.34 dBm. The UE throughput at the edge of SC_3 has been decreased from 17.66 Mbps to 5.98 Mbps.

Chapter 6

Conclusions and future works

This final chapter presents the conclusions and key insights gained from the project's development. It also outlines potential future directions that could inspire subsequent projects.

6.1 Conclusions

The number of parameters to be optimized in 4G and 5G networks has grown so large that they are no longer easily manageable by traditional human-driven algorithms. Given that the cost of network infrastructure is already accounted for, operators are now primarily focused on optimizing their resources. Current solutions increasingly rely on replacing manual SON network algorithms with Machine Learning (ML) and AI techniques. This project aims to demonstrate that this trend is not merely a product of the AI "Boom", but a powerful tool for optimizing network infrastructure and reducing costs.

The simulator used to emulate the behaviour of a 5G NG-RAN underscores the complexity of the numerous interdependent parameters involved, highlighting the challenges of developing manual solutions based solely on expert knowledge. However, despite the use of AI, a deep understanding of the network and its behaviour remains a fundamental component of the developed solution. Reinforcement Learning (RL) was selected as the approach, wherein an agent learns autonomously based on a reward system. Specifically, Deep Q-learning Network (DQN) was utilized, as this Deep Reinforcement Learning (DRL) technique has demonstrated promising results in other use cases and serves as an effective initial approach.

To enable the DQN agent to learn the desired optimization objective, it is essential to guide it in finding optimal solutions. This is accomplished by carefully designing the action space, state space, and reward algorithm.

The solution developed in the project was based on the following premises:

- State space variables should describe everything necessary for the optimization goal and not add redundant information.
- State space variables must accurately reflect the changes resulting from the actions taken by the agent.
- The reward algorithm should be directly aligned with the optimization goal.
- The reward algorithm must allow the agent to link reward values with action-state pairs, thus achieving effective generalization in its learning process.

Chapter 4 details the implementation of these principles within the project's use case and explains the rationale behind the decisions made. Chapter 5 provides an evaluation of the solution, showcasing various results and demonstrations. From the obtained results, the following conclusions have been drawn:

- Dividing the reward algorithm into three stages has proven to be an effective strategy for ensuring a high success rate. The first stage focuses on ensuring that the minimum throughput is met in the slice, rewarding negatively until it is reached. This approach has enabled the agent, using the throughput variable from the state space, to accurately identify its current situation. If the throughput is below T_{min}, the agent selects actions to increase bandwidth and, consequently, throughput. Conversely, when throughput exceeds T_{min}, the agent's actions vary widely based on the specific context, influenced by the subsequent two stages of the reward algorithm. This reward algorithm leads the agent to focus on guaranteeing the minimum throughput before doing anything else. So the success rate is very high, as presented in chapter 5, with values very close to 100%.
- The second and third stages of the reward algorithm concentrate on minimizing the assigned PRBs and reducing interference, respectively. To minimize bandwidth efficiently, it is necessary to eliminate as much interference as possible. The reward algorithm has been designed with this principle in mind, assigning higher rewards for reducing interference compared to reducing PRBs. Both the results presented in Table 5.6 and the specific demonstrations in Section 5.3 illustrate how the agent has learned this behaviour. In the performance study of the agents, it is observed that whenever a throughput value very close to T_min is reached, the percentage of interference-free is also high compared to the rest of the cases. In the step-by-step demonstration

examples, it can be seen how in all cases the agent first eliminates all interference. Once this is accomplished, it then reduces bandwidth as much as possible until the throughput approaches the minimum allowed. As a result, this approach allows for the transmission power to be reduced as much as possible, which is the fundamental objective.

- The inclusion of the "PRB position flag" as a state space variable has enabled the agent to recognize when it should avoid adding PRBs outside the available spectrum range. In other words, it identifies when the assigned PRBs are at the left or right edge of the PRBs array. Consequently, it will use add or shift actions in the opposite direction to these edges.

- The presence of the "Interference PRB position flag" allows the agent to identify where most of the interference is located and take actions to avoid it more efficiently. As can be seen from the examples in section 5.3, the direction (right or left) in which the agent adds/removes or moves PRBs is not arbitrary. In all cases, it can be observed that the agent's actions related to avoiding interference depend on where the interference is located. As described in this section, the agent chooses actions that eliminate interference more quickly. For instance, if interference is predominantly on the left side of the spectrum, the agent will shift bandwidth allocation to the right or remove PRBs from the left side.

In addition to these conclusions, which validate the effective design of the action space, state space, and reward algorithm, the results achieved are highly satisfactory with respect to the optimization objectives of the problem. The first condition that sets a minimum throughput in the slice has been measured as the agent's success rate. The results presented in Table 5.6 indicate a very high success rate across most configurations. Additionally, the specific examples in Section 5.3 demonstrate that the minimum throughput is consistently guaranteed in all cases.

Once the first condition is guaranteed, the objective is to minimize the transmission power. To measure it, the distance to the minimum established throughput is measured, since this will indicate the maximum possible power saving. This value, referred to as the agent's accuracy, is highly dependent on the training hyperparameters. The best-performing agent achieved an average throughput of 5.5 Mbps, with $T_{min} = 5Mbps$. In the demonstration examples, it is evident that, regardless of the initial throughput, the final throughput consistently approaches T_{min}.

The solution has been tested in two scenarios: the first with 2 SCs and the second with 4 SCs. The size of the state space is determined by the

number of neighbouring cells defined for each SC. As the size of the state space increases, the agent requires more extensive exploration during training. If the number of neighbours is too high, the computational cost may become prohibitive, potentially hindering the agent's ability to find optimal solutions efficiently. However, not all cells in a network are neighbours to every other cell. Operators define neighbourhoods based on specific criteria. For the agent to operate effectively, it is essential to include only those neighbours that contribute significantly to harmful interference. Since there may be many cells that cause very small interference that is not worth considering in the solution.

Finally, with respect to DQN, it is clear that training hyperparameters play a crucial role. Extended training time does not necessarily ensure better performance. The effectiveness of the agent depends on the specific problem, requiring careful adjustment of factors such as exploration rate, the number of steps per episode or the size of the replay buffer. When dealing with a complex problem where intuition may fall short, it is essential to explore various training configurations and thoroughly evaluate the results to identify the optimal agent.

6.2 Future works

The agent developed in this project is constrained by the capabilities of the radio access network simulator used to emulate network behaviour. Although the simulator is relatively simple and includes only the most basic KPIs, it is adequate for producing behaviour that closely approximates real-world conditions. Increasing the complexity of the simulator presents a significant challenge in designing the action space, state definitions, and reward algorithm, due to the more intricate relationships between simulator parameters. However, the greater the complexity, the closer it will be to the behaviour of the real network. In this context, the primary direction for the project's continuation is to deploy it in a real production environment or in a laboratory that closely emulates the network's behaviour.

Another potential improvement related to increasing the simulator's complexity involves incorporating additional network KPIs. These new KPIs may prove to be more effective in the state space than those currently used in the existing solution. For example, if the operator uses a specific definition of interference, incorporating this KPI would better align with the actual interference conditions and assist the agent in learning how to reduce it more effectively. However, in this project, we have full knowledge of the equations implemented by the simulator, allowing us to precisely understand the behaviour of interference. As a result, the state space is specifically aligned

with interference reduction. In contrast, in a real network, there are no perfect equations that fully capture the relationships between different metrics. Therefore, utilizing KPIs that reflect the operator's specific understanding of interference in their network will guide the agent toward achieving satisfactory results.

Finally, another key area for future work is the exploration of alternative Deep Reinforcement Learning (DRL) techniques. The project has employed DQN and highlighted the inherent complexity of this technique. Achieving an optimal solution demands extensive trial and error, as well as a deep understanding of the underlying principles. However, there are numerous other techniques, each with its own complexity. It may be valuable to compare the performance of these techniques to identify potential improvements or alternative approaches. For example, Proximal Policy Optimization (PPO), Advantage Actor-Critic (A2C) could be studied.

Bibliography

[1] Mansoor Shafi, Andreas F. Molisch, Peter J. Smith, Thomas Haustein, Peiying Zhu, Prasan De Silva, Fredrik Tufvesson, Anass Benjebbour, and Gerhard Wunder. 5g: A tutorial overview of standards, trials, challenges, deployment, and practice. *IEEE Journal on Selected Areas in Communications*, 35(6):1201–1221, 2017.

[2] BR. Recommendation itu-r m.2083-0 imt vision-framework and overall objectives of the future development of imt for 2020 and beyond m series mobile, radiodetermination, amateur and related satellite services.

[3] C. Johnson. *5G New Radio in Bullets*. Independently published, 2019.

[4] H. Holma, A. Toskala, and T. Nakamura. *5G Technology: 3GPP New Radio*. Wiley, 2020.

[5] Erik Dahlman, Stefan Parkvall, and Johan Sköld. Chapter 7 - overall transmission structure. In Erik Dahlman, Stefan Parkvall, and Johan Sköld, editors, *5G NR: the Next Generation Wireless Access Technology*, pages 103–131. Academic Press, 2018.

[6] Jose Ordoñez Lucena, Pablo José Ameigeiras Gutiérrez, Diego Lopez, Juan José Ramos Muñoz, Javier Lorca, and Jesús Folgueira. Network slicing for 5g with sdn/nfv: Concepts, architectures and challenges, 2017.

[7] Paulo Valente Klaine, Muhammad Ali Imran, Oluwakayode Onireti, and Richard Demo Souza. A survey of machine learning techniques applied to self-organizing cellular networks. *IEEE Communications Surveys & Tutorials*, 19(4):2392–2431, 2017.

[8] Alexandros Kaloxylos, Anastasius Gavras, Daniel Camps Mur, Mir Ghoraishi, and Halid Hrasnica. Ai and ml–enablers for beyond 5g networks. 2021.

[9] Laura Graesser and Wah Loon Keng. *Foundations of deep reinforcement learning: theory and practice in Python*. Addison-Wesley Professional, 2019.

[10] `https://github.com/davidfernxndez/DQN-Resource-Allocation-Agent-in-5G`. Available in GitHub.

[11] Maxim Lapan. *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt Publishing Ltd, 2018.

[12] Shunliang Zhang. An overview of network slicing for 5g. *IEEE Wireless Communications*, 26(3):111–117, 2019.

[13] Prashant Subedi, Abeer Alsadoon, PWC Prasad, Sabih Rehman, Nabil Giweli, Muhammad Imran, and Samrah Arif. Network slicing: A next generation 5g perspective. *EURASIP Journal on Wireless Communications and Networking*, 2021(1):102, 2021.

[14] Juan Ramiro and Khalid Hamied. *Self-Organizing Networks (SON): Self-Planning, Self-Optimization and Self-Healing for GSM, UMTS and LTE*. Wiley Publishing, 1st edition, 2012.

[15] Osianoh Glenn Aliu, Ali Imran, Muhammad Ali Imran, and Barry Evans. A survey of self organisation in future cellular networks. *IEEE Communications Surveys Tutorials*, 15(1):336–361, 2013.

[16] S. Hämäläinen, H. Sanneck, and C. Sartori. *LTE Self-Organising Networks (SON): Network Management Automation for Operational Efficiency*. Wiley, 2011.

[17] Bom Rong, Xuesong Qiu, Michel Kadoch, Songlin Sun, and Wenjing Li. *5G heterogeneous networks: Self-organizing and optimization*. Springer, 2016.

[18] Hafiz Yasar Lateef, Ali Imran, and Adnan Abu-Dayya. A framework for classification of self-organising network conflicts and coordination algorithms. In *2013 IEEE 24th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*, pages 2898–2903. IEEE, 2013.

[19] European partnership under horizon europe smart networks and services, 2020.

[20] Nan Hu. 5g white paper 2 by ngmn alliance, 2020.

[21] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1998.

[22] Gareth M James. Variance and bias for general loss functions. *Machine learning*, 51:115–135, 2003.

[23] Chaoyun Zhang, Paul Patras, and Hamed Haddadi. Deep learning in mobile and wireless networking: A survey. *IEEE Communications surveys & tutorials*, 21(3):2224–2287, 2019.

[24] Wei Yang Bryan Lim, Nguyen Cong Luong, Dinh Thai Hoang, Yutao Jiao, Ying-Chang Liang, Qiang Yang, Dusit Niyato, and Chunyan Miao. Federated learning in mobile edge networks: A comprehensive survey. *IEEE Communications Surveys & Tutorials*, 22(3):2031–2063, 2020.

[25] Francesco Restuccia and Tommaso Melodia. Deep learning at the physical layer: System challenges and applications to 5g and beyond. *IEEE Communications Magazine*, 58(10):58–64, 2020.

[26] Zehui Xiong, Yang Zhang, Dusit Niyato, Ruilong Deng, Ping Wang, and Li-Chun Wang. Deep reinforcement learning for mobile 5g and beyond: Fundamentals, applications, and challenges. *IEEE Vehicular Technology Magazine*, 14(2):44–52, 2019.

[27] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[28] Rongpeng Li, Zhifeng Zhao, Xianfu Chen, Jacques Palicot, and Honggang Zhang. Tact: A transfer actor-critic learning framework for energy saving in cellular radio access networks. *IEEE transactions on wireless communications*, 13(4):2000–2011, 2014.

[29] Tran Anh Quang Pham, Yassine Hadjadj-Aoul, and Abdelkader Outtagarts. Deep reinforcement learning based qos-aware routing in knowledge-defined networking. In *Quality, Reliability, Security and Robustness in Heterogeneous Systems: 14th EAI International Conference, Qshine 2018, Ho Chi Minh City, Vietnam, December 3–4, 2018, Proceedings 14*, pages 14–26. Springer, 2019.

[30] Xianfu Chen, Honggang Zhang, Celimuge Wu, Shiwen Mao, Yusheng Ji, and Medhi Bennis. Optimized computation offloading performance in virtual edge computing systems via deep reinforcement learning. *IEEE Internet of Things Journal*, 6(3):4005–4018, 2018.

[31] Chen Qi, Yuxiu Hua, Rongpeng Li, Zhifeng Zhao, and Honggang Zhang. Deep reinforcement learning with discrete normalized advantage functions for resource management in network slicing. *IEEE Communications Letters*, 23(8):1337–1341, 2019.

[32] Richard Bellman. A markovian decision process. *Journal of mathematics and mechanics*, pages 679–684, 1957.

[33] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[34] David Silver. Lecture 6: Value function approximation. *UCL, Computer Sci. Dep. Reinf. Learn. Lect.*, pages 1–15, 2015.

[35] Long-Ji Lin. Self-improving reactive agents based on reinforcement learning, planning and teaching. *Machine learning*, 8:293–321, 1992.

[36] 3rd Generation Partnership Project. Technical Report TR 38.803. Technical Report V14.0.0, 3rd Generation Partnership Project; Technical Specification Group Radio Access Network, March 2017. RF and coexistence aspects.

[37] Jorge Navarro-Ortiz, Oriol Sallent, Jordi Pérez-Romero, et al. Radio access network slicing strategies at spectrum planning level in 5g and beyond. *IEEE access*, 8:79604–79618, 2020.

[38] Gym documentation. `https://www.gymlibrary.dev/index.html`.

[39] Dqn — stable baselines3 2.4.0a5 documentation. `https://stable-baselines3.readthedocs.io/en/master/modules/dqn.html`.

[40] Tensorboard — tensorflow. `https://www.tensorflow.org/tensorboard?hl=es-419`.

Appendices

Appendix A

UEs assignment to serving SCs

The propagation model that defines the path loss of the UEs is based on linear regression, assuming NLOS situation:

$$L(d) = L_0 + n \cdot \log_{10}(d) \quad (dB) \tag{A.1}$$

Where L_0 is a constant that represents the path loss at a reference distance $d_0 = 1m$, and n is the exponent of the variation with distance, which depends on the environment and the characteristics of the transmission medium. The distance between UE u and SC b is given by:

$$d = \sqrt{(b_x - u_x)^2 + (b_y - u_y)^2 + (b_z - u_z)^2} \tag{A.2}$$

In this project, the assignment of UEs to the SCs that serve them is strictly determined by path loss. The UEs will be assigned to the SC from which they receive the highest Received Signal Strength (RSS), as defined by the path loss:

$$RSS(d) = -L(d) \quad (dB) \tag{A.3}$$

The coordinates of SCs (b_x, b_y) and UEs (u_x, u_y) used in Equation A.2 for distance calculation are defined based on the dimensions of the scenario. In the project, we work with rectangular scenarios. The coordinates in the z dimension correspond to the vertical heights of the antenna and the user. The pseudocode of the process of assigning UEs to SCs is shown in algorithm 2.

Algorithm 2 UEs assignment to serving SCs

Set number of UEs, U
Set number of SCs, B
Set UEs position, $[U_x, U_y, U_z]$
Set SCs position, $[B_x, B_y, B_z]$
for $b = 0$ to B **do**
    for $u = 0$ to U **do**
        $d(b,u) = \sqrt{(b_x - u_x)^2 + (b_y - u_y)^2 + (b_z - u_z)^2}$
        $L(b,u) = L_0 + n \cdot \log_{10}(d(b,u))$
        $RSS(b,u) = -L(d,u)$
    end for
end for
$SC(u) = max(RSS(:,u))$

Appendix B

Radio Acces Network (RAN) simulator

The network model proposed in Section 3.1.1 has been implemented in Python as a radio access network simulator. This simulator operates according to the defined network design and emulates the behaviour of a real network, allowing interaction with the agent.

The pseudocode of the implemented RAN simulator is shown in algorithm 3. The simulator has knowledge of which SC serves each UE, obtained directly from the assignment of UEs to SCs performed in Algorithm 2.

First, the power received by each UE from all the SCs in the scenario is calculated (lines 20 to 25). Since these powers make up the expression of the SINR.

The load factor of the SCs is determined by aggregating the contributions of each UE served by the respective SC (lines 29 to 33). The user demand D_u and its spectral efficiency SE_u are the same for all UEs.

SINR and throughput are calculated for all the UEs in each assigned PRB (lines 36 to 43). Finally, the total throughput experienced by the UE will be the cumulative sum of the throughput across all allocated PRBs (line 44).

In the proposed network design, the SINR and throughput levels remain uniform across all PRBs. This uniformity arises because the received power at UEs is independent of frequency, and the noise power is assumed constant across the entire frequency band.

Algorithm 3 RAN simulator operation

Set number of UEs, U
Set number of SCs, B
Set UEs position, $[U_x, U_y, U_z]$
Set SCs position, $[B_x, B_y, B_z]$
Set number of PRBs in each SC, $[R_0, R_1, ..., R_B]$
Set power per PRB, P_R
Server SC Assignment $SC(u)$

1) Determine the power received by the UEs for all the SCs

for $b = 0$ to B **do**
    $P_{TX}(b) = P_R \cdot R_b$
    for $u = 0$ to U **do**
        $P_{RX}(b, u) = P_{TX}(b) - L(b, u)$
    end for
end for

2) Calculate cell load factor in each SC

for $u = 0$ to U **do**
    $b = SC(u)$
    $load(b) = load(b) + D_u/(R_b \cdot BW_{PRB} \cdot SE_u)$
end for
$load = min(load, 1)$

3) Calculate throughput for all UEs
for $u = 0$ to U **do**
    $b = SC(u)$
    for r in R_b **do**
        $interf(u, r) = \sum_{j \neq b} load(j) \cdot {P_{j,u}}^{RX}$
        $SINR(u, r) = {P_b}^{TX}/(interf + P_N)$
        Determine $SE(u, r)$ according to equation 3.7.
        $T(u, r) = (BW_{PRB} \cdot SE(u, r))/U_b$
    end for
    $T(u) = sum(T(u, :))$
end for

Appendix C

Advantages of discretizing throughput as a state variable

This appendix shows the improvement in the agent's learning as a consequence of reducing the dimensionality of the state space by discretizing the throughput variable.

In the observation that describes the state of the cells, the UE throughput at the edge of the cell appears. This variable is continuous in nature; for instance, in the experimental setup detailed in Chapter 5, it takes values within the range of (0.80] Mbps. In the developed solution, a discretization of this variable has been carried out to reduce the dimensionality of the state space and facilitate agent learning.

To demonstrate the benefits of discretization, an experiment was conducted using the same experimental setup described in Section 5.1. Two agents have been trained with the configuration listed in Table C.1. These

Steps per episode	Episodes	Exploration fraction (%)	Throughput
1000	3000	70	discrete
1000	3000	70	float

Table C.1: Agent training configuration to demonstrate benefits of discretization.

are two agents with identical training configurations; however, in one, the throughput variable is treated as a float, while in the other, it is treated as a discrete data type. The discretization is performed using 62 bins, as described in Section 5.1.2.

First, the convergence curves of both agents are compared, as shown in Figure C.1.

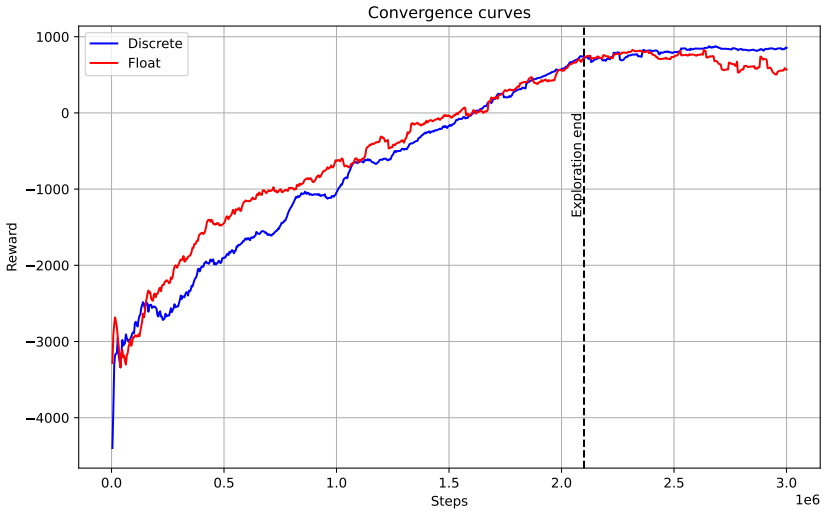


Fig. C.1: Convergence curves for agents with discrete throughput versus continuous throughput.

In Reinforcement Learning (RL), convergence curves illustrate the average reward per episode over the course of training. Based on the training configuration, exploration is performed according to ϵ-greedy policy up to step 2,100,000. After this point, the agent performs exploitation with a probability of 99.95%. At the beginning of the exploration phase, the agent with continuous throughput (represented by the red curve) achieves a higher reward than the agent with discrete throughput (represented by the blue curve). However, by the end of the exploration phase, both agents achieve practically identical rewards.

The most significant and noticeable difference between the two trainings occurs during the exploitation phase. The agent with discrete throughput (represented by the blue curve) clearly converges and maintains a stable reward level. In contrast, the agent with continuous throughput (represented by the red curve) experiences a decrease and does not converge stably, instead showing a noisy pattern.

In terms of convergence, we can observe a clear improvement in the agent with throughput as a discrete variable. To evaluate the behaviour

learned by each agent, both are subjected to the same 500 initial situations in Scenario A (see Figure 5.3a). This type of experiment is the same as that performed with 24 agents trained in section 5.2. Consequently, the evaluation will include the same metrics: success rate, average throughput value (precision), and rate of interference-free episodes. The results obtained are shown in Table C.2.

Throughput	Success rate (%)	Average throughput (Mbps)	Interference-free rate (%)
discrete	98.40	5.87	84.88
float	85.80	7.1	52.63

Table C.2: Discrete throughput versus continuous throughput metrics results.

The advantage observed in the convergence curves is evidenced by the result obtained in the agent performance metrics. In addition to achieving a higher success rate with throughput discretization, precision also improves significantly. The average throughput achieved is much closer to T_{min} = 5Mbps.

Appendix D

Code repository

This appendix details the structure of the code developed for this project, which is available in the GitHub repository referenced in [10].

First, the Python libraries are listed in Table D.1 along with the version compatible with the code. The Python version is 3.8.0.

Package	Version
gym	0.17.3
stable-baselines3	1.0
numpy	1.24.3
pandas	2.0.3
scipy	1.10.1
matplotlib	3.7.4
seaborn	0.12.2
tensorboard	2.13.0
tensorflow	2.13.0
keras	2.13.1

Table D.1: Versions of Python libraries compatible with the developed code in [10]

Figure D.1 shows the folder structure of the repository. The code is organized into two main sections.

D.1 DRL Framework

The first section, which includes the files *Environment.py*, *Training.py*, and *Inference.py*, is responsible for implementing the framework required to train DQN agents for the project's use case. The *environment.py* file defines a

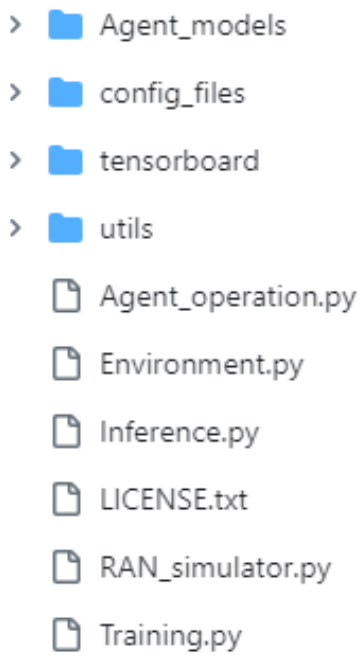


Fig. D.1: Repository folder structure.

class that adheres to the interface provided by the Gym library. Within this class, the action space, state space, and reward algorithm are implemented alongside the RAN simulator. In summary, this class contains the complete implementation described in Chapter 4.

The *training.py* file is used to train agents within the environment defined in *environment.py*. To train an agent you need to configure 4 variables: *num_cells*, *num_steps*, *num_episodes* and *exploration_fraction*. The number of cells determines whether the agent is trained in Scenario A or Scenario B. The environment configuration is based on the files located in the *config_files* folder. The configuration files in this folder are CSV files that specify the parameter values for the scenario and the RAN simulator. The number of steps, number of episodes and the exploration fraction are the 3 training parameters that have been studied in Chapter 5. Once the value of these variables is established, the DQN class of the stable_baselines3 library is used to train the agent.

The model produced after training is saved in a zip file named:

```
{num_cells}_cells_{num_steps}_steps_{num_episodes}_ep_{
    exploration_fraction}_fraction.zip
```

Reflecting the values of the training configuration variables. The training progress log is stored in the *tensorboard* folder. For graphical representa-

tion of the training logs, the TensorFlow visualization toolkit [40], TensorBoard, is used. The trained agent models are stored in the *Agent_models* folder, while their corresponding training logs can be found in the *tensorboard* folder.

The *inference.py* file enables the execution of inference procedures for one or more agent models. To perform inference, you need to specify the name of the .zip file containing the models and set the number of inference episodes. The resulting metrics for each model are displayed on the screen, and the outcomes of each episode are saved in a CSV file. This inference procedure is the one carried out in section 5.2 to study the performance of 24 agents. The metrics evaluated are those listed in table 5.6.

D.2 Agent operation

The second section of the code consists of the files *Agent_operation.py* and *RAN_simulator.py*. This code enables the trained agent to be applied across all cells in either of the two scenarios. That is, the operation of the agent as shown in section 5.3.

The *RAN_simulator.py* file serves as the equivalent of the environment class. However, it is not a Gym class and does not implement the state space and reward algorithm. This file implements the RAN simulator to emulate the behaviour of the 5G network. It receives actions from the agent, adjusts the bandwidth of the cells accordingly, and provides the agent with observations that describe the state of each cell.

In the *Agent_operation* file, a trained agent model is selected, and an initial state is set, including the number of PRBs assigned to each cell and their positions within the spectrum. The agent takes actions on the cell bandwidth until the optimization objective is reached in all cells in the scenario. This script plots the step-by-step evolution graphs shown in section 5.3.