TITLE: A Reinforcement Learning algorithm for optimal Virtual Network Functions allocations among 5G Edge Computing centers

MASTER DEGREE: Master's degree in Applied Telecommunications and Engineering Management (MASTEAM)

AUTHOR: Carlos Ruiz de Mendoza

ADVISOR: Bakhshi, Bahador. Mangues, Josep. Zeydan, Engin

SUPERVISOR: Villegas Garcia, Eduard

DATE: July, 2nd 2021


BY NC SA

Abstract

The fifth generation (5G) mobile networks are enabling operators and stakeholders to enhance and innovate new services in response to an increasing market demand. 5G architecture provides scalability and flexibility for adapting its infrastructure to a customizable communication system by means of Cloudification.

Softwarization and virtualization are key terms for upcoming industries that will require ultra-low latency, only possible if the infrastructure equipment that traditionally was centralized in the communication network core is physically moved closer to the user, at the network edge.

The main objective of this master thesis was to implement a Reinforcement Learning algorithm (Q-Learning Temporal Difference) aimed at next generation networks to optimally allocate Virtualized Network Functions (VNF) to 5G network Edge Computing (EC) centers.

In order to evaluate the algorithm performance and compare it, two more algorithms have been developed to achieve a solution under the same network circumstances. The first one, Best Fit, was inspired by a classical network load balancing algorithm (Weighted Round Robin), whereas the second, MDP, was approached through dynamic programming (Policy Iteration), having posed the network dynamics as a finite Markov Decision Process.

The several tests that have been carried out indicate that Q-Learning performs better than the Best Fit and almost as close as the MDP. It shows that the Q-Learning algorithm is able to allocate optimally the incoming VNF demands when EC centers' available resources are somehow restricted.

Keywords

5G, Cloudification, Softwarization, Virtualization, Virtualized Network Functions, Edge Computing, Reinforcement Learning, Markov Decision Process

Resumen

Las redes móviles de quinta generación (5G) permiten a los operadores y partes interesadas mejorar e innovar nuevos servicios en respuesta a la creciente demanda del mercado. La arquitectura 5G proporciona escalabilidad y flexibilidad para adaptar su infraestructura a un sistema de comunicación personalizable mediante tecnologías como la "*Cloudification*".

La softwarización y la virtualización son aspectos clave para las industrias futuras que requerirán de una latencia ultrabaja, sólo posible si los equipos de infraestructura que tradicionalmente estaban centralizados en el núcleo de la red de comunicaciones, son instalados físicamente más cerca del usuario, en el borde de la red.

El objetivo principal de esta tesis de maestría ha sido la de implementar un algoritmo de aprendizaje por refuerzo (Q-Learning Temporal Difference) dirigido a redes de próxima generación con el fin de asignar, de manera óptima, las funciones de red virtualizadas (VNF) en los distintos centros de "*Edge Computing*" (EC) de una red 5G.

Para poder evaluar el rendimiento del algoritmo y compararlo, se han desarrollado dos algoritmos más que obtienen una solución en las mismas condiciones de una red. El primero, Best Fit, está inspirado en un algoritmo clásico de balanceador de carga de red (Weighted Round Robin), mientras que el segundo, MDP, se ha enfocado a través de programación dinámica (Policy Iteration), habiendo planteado la dinámica de la red como un proceso de decisión finito Markoviano.

Las diversas pruebas que se han llevado a cabo indican que Q-Learning obtiene mejores resultados que Best Fit y se encuentra próximo, en cuestión de rendimiento, del MDP. Q-Learning, muestra que es capaz de asignar de manera óptima las distintas demandas de VNF cuando los recursos disponibles de los centros de EC están limitados.

Palabras clave

5G, Cloudification, Softwarization, Virtualization, Virtualized Network Functions, Edge Computing, Reinforcement Learning, Markov Decision Process.

Resum

Les xarxes mòbils de cinquena generació (5G) permeten als operadors i a les parts interessades millorar i innovar nous serveis en resposta a la creixent demanda del mercat. L'arquitectura 5G proporciona escalabilitat i flexibilitat per adaptar la seva infraestructura a un sistema de comunicació personalitzable mitjançant tecnologies com la "*Cloudification*".

La softwarització i la virtualització són aspectes clau per a les indústries futures que requeriran de latències ultra baixes, només possible si els equips d'infraestructura que tradicionalment estaven centralitzats en el nucli de la xarxa de comunicacions, són instal·lats físicament més a prop de l'usuari, a la vora de la xarxa.

L'objectiu principal d'aquesta tesi de mestratge ha estat la d'implementar un algoritme d'aprenentatge per reforç (Q-Learning Temporal Difference) dirigit a xarxes de pròxima generació per tal d'assignar, de manera òptima, les funcions de xarxa virtualitzades (VNF) en els diferents centres de "*Edge Computing*" (EC) d'una xarxa 5G.

Per poder avaluar el rendiment de l'algorisme i comparar-lo, s'han desenvolupat dos algorismes més que obtenen una solució en les mateixes condicions d'una xarxa. El primer, Best Fit, està inspirat en un algoritme clàssic balancejador de càrrega de xarxa (Weighted Round Robin), mentre que el segon, MDP, està enfocat a través de la programació dinàmica (Policy Iteration), havent plantejat la dinàmica de la xarxa com un procés de decisió finit Markovià.

Les diverses proves que s'han dut a terme indiquen que Q-Learning obté millors resultats que Best Fit i es troba pròxim en qüestió de rendiment de l'MDP. Q-Learning, mostra que és capaç d'assignar de manera òptima les diferents demandes de VNF quan els recursos disponibles dels centres d'EC estan limitats.

Paraules clau

5G, Cloudification, Softwarization, Virtualization, Virtualized Network Functions, Edge Computing, Reinforcement Learning, Markov Decision Process.

Acknowledgement

I would like to thank Josep, Engin and Bahador, my advisors at the CTTC, the total support they have given me and their advising all through this work. I would also like to thank Eduard, my supervisor at the UPC for the present master thesis, the help that offered me and his advice. Last but not least, I'd like to thank all the open source Python community.

List of figures

List of tables

Pg.

CONTENTS

INTRODUCTION

5G is the fifth generation of mobile network, a communication standard that introduces new enhancements with regard to legacy mobile network systems (4G). It accomplishes faster data speeds, up to 100 times faster than 4G and ultra-low latency times (<1ms), independently whether or not other devices are connected at the same time.

5G is no longer a hype, it is the present and the driver of hyperconnectivity, digital transformation and the economy of the future. It is a technological evolution that paves the way for endless products, services and applications. It provides the automotive industry with a wide spectrum of opportunities, for the development and deployment of automated driving and as an essential prerequisite for the future smart factory. 5G is, along with other disruptive technologies, such as the Internet of things (IoT), Artificial Intelligence (AI), robotics, cloud computing, virtual and augmented reality (VR/AR), etc., a key element in the digital transformation in which we are immersed.

According to [1], the autonomous vehicle industry is expected to generate $12.9 billion by 2026, in 5G related software expenditures and $39 billion in Edge Computing (EC) infrastructure to support such industry. Grand View Research [2] states that the 5G services market size is estimated to reach $664 billion with a compound annual growth rate (CAGR) of 46.2% by the year 2028.

5G networks will deploy more cells and antennas with advanced technologies such as virtual Radio Access Network (vRAN), allowing for partial or full virtualization of networks. 5G relies heavily on Edge Computing centers cutting the distance between the end user and the technological resources. By locating such centers at the network edge, data is no longer stored or processed at some distant data center which translates in a considerable low latency.

The latter offers flexible upgrades as well as innovations in services. The 5G technologies handle a huge increase in capabilities, all in a multi vendor environment with an ultra low latency data transmission emphasis. No wonder that 5G networks become such complex systems hindering the tracking down of a service outage or any other issue. The plethora of possibilities as far as provided services even makes it more difficult to determine the optimal way to provide services in the future and to foresee the demand of them in order to deploy an infrastructure in the most efficient way. With such a huge diverse vertical industry potential market, Mobile Network Operators (MNO) and services providers search for strategies to increase their revenues and lower their expenditures; Capital Expenditure (CAPEX) and Operating Expenditure (OPEX).

Such a complex system requires advanced methods to prevent potential failures and control of their networks. As stated by the 5G Infrastructure Public Private Partnership (5G-PPP) [3], AI and Machine Learning (ML) will be included as new technology forms to overcome such challenges. AI/ML allows MNO to bring to 5G customers its benefits, to improve the current model of centralized networks straining themselves to deliver the increasingly demanding services. AI/ML can efficiently manage resource deployment based on infrastructure required conditions and for the network to perform effectively. For MNO, it implies the possibility to forecast the overall demand and foresee a plan for 5G network and multiple-input and multiple-output (MIMO) site configuration as well as base stations (BS) deployments.

Objectives

This master thesis aims at the following purposes:

- Introduce the technological key aspects that drive 5G to accomplish some of the defined standards by the (3GPP). To present the 5Growth platform that aims to validate the operating of 5G systems deployed on vertical industries and optimize the performance, empowering automation and AI driven solutions.

- Introduce the fundamental theory aspects of Reinforcement Learning (RL) in order to understand some of the implemented algorithms.

- Implement a RL algorithm (Q-Learning Temporal Difference), a second RL algorithm (Policy Iteration) and a third algorithm inspired by a classical network load balancer implementation (Weighted Round Robin) to manage the optimal assignment of Virtualized Network Functions (VNF) demands over a 5G network and its EC centers.

- To conduct an analysis between the different algorithms from its simulation results to evaluate how good the Q-Learning algorithm performs against the other two algorithms.

Thesis outline

Chapter 1 broadly introduces the 5G expected features in terms of technological specifications to satisfy the three main foreseen relevant usage scenarios defined by the International Telecommunication Union, International Mobile Telecommunication 2020 (ITU IMT-2020). It also presents the concepts of Network

Functions Virtualization (NFV) and EC, technological key aspects for 5G. 5Growth architecture and its AI/ML platform are presented in order to contextualize the implementation of the Q-Learning algorithm. Finally all theory fundamentals regarding RL are explained so as to understand the functioning of the implemented RL algorithms.

Chapter 2 introduces the problem statement to be tackled as well as the different strategies than have been followed in order to accomplish each algorithm implementation. Details the aspects for modeling the required scenario, what it is called the environment in RL. Finally major considerations on each algorithm are presented in order to understand the complexity of each implementation.

Chapter 3 Analyzes the simulation results from each algorithm. Different tests are carried out in order to see how, either the EC available resources or the VNF demands, affect the performance on each algorithm. Important concepts, like learning rate, discount factor or exploration/exploitation are tested in order to see how it affects the final results of the Q-Learning algorithm.

Complementary resources are added in the annex in order to support some of the explanations in chapter 2 or provide extra simulation graphs to have a different perspective of the results.

CHAPTER 1. FUNDAMENTAL ASPECTS

This chapter gives a brief overview about the 5G key features related with usage scenarios. Section 2.2 introduces the elemental aspects of 5Growth platform and the innovations included concerning AI and ML. Section 2.3, 2.4 and 2.5 describes all theoretical fundamentals needed to understand the RL algorithms that further on will be implemented.

1.1. 5G

5G is the fifth generation of mobile communications systems. A standard beyond 4G Long Term Evolution (LTE) and LTE-Advanced. The increasing demand in data traffic continues growing year after year forcing the 4G to reach its limits. Hence it makes sense to define a whole new system rather than trying to improve an obsolete one.

Towards the end user, perhaps the most notable advantage of the 5G is the high speed and low latency. Further than being able to download ultra high resolution videos in a matter of seconds, the 5G represents a large technological

breakthrough giving birth to what is known as the 4th industrial revolution. If globalization changed the way of understanding business, now the industry will go one step further thanks to a total interconnection between machines and people. Figure 2.1 depicts a comparison of key capabilities between the 4G and 5G networks.

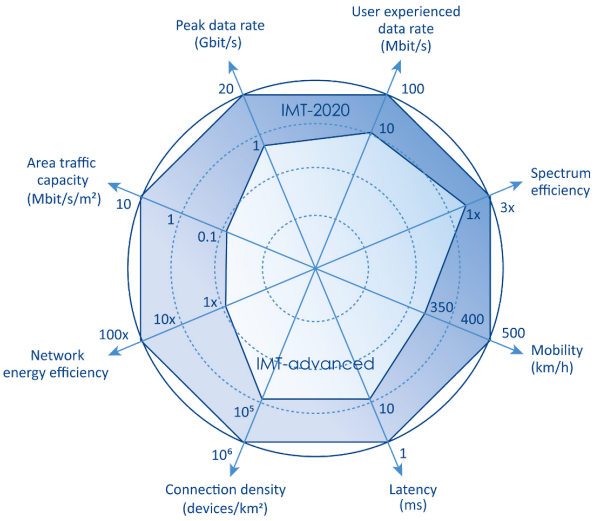


Fig. 2.1. Comparison of IMT-Advanced (4G) with IMT-2020 (5G) capabilities (source [4])

The ITU IMT-2020 defines three main foreseen relevant usage scenarios [4]:

1. **Enhanced Mobile Broadband** (eMBB): This scenario points to a usage case where high data rates and high user density exceed the actual legacy networks. eMBB is seen as addressing human-centric communication.

2. **Massive machine type communications** (mMTC): Aimed at pure machine communications, this dimension is designed to cope with a large number of devices connected to the internet, the well known IoT.

3. **Ultra-reliable and low-latency communications** (URLLC): This usage block aims at both human- and machine-centric communications. Provides the lowest latency which makes it ideal for those critical applications where real time communication is a must.

Currently, as of the date of this master thesis, 5G is in its 16th release by the 3GPP. It specifies the second phase network deployment complying with the ITU global requirements, offering superior data speeds, advanced solutions for network slicing, VNF or support for massive IoT simultaneous device communications, among others.

1.1.1. Network Functions Virtualization

NFV refers to a virtualized task previously performed by proprietary and dedicated hardware. NFV moves network functions away from dedicated hardware devices into software. This allows specific functions that required hardware devices in the past (e.g. Equipment Identity Register (EIR), routers, firewalls, etc) to work on Commercial-Off-The-Shelf (COTS) hardware, expediting cost-effective commercial deployments.

NFV allows 5G to customize and optimize resources tailored to different vertical markets[1]. This is achieved through Network Slicing where a physical network can be split into several virtual networks supporting multiple RAN.

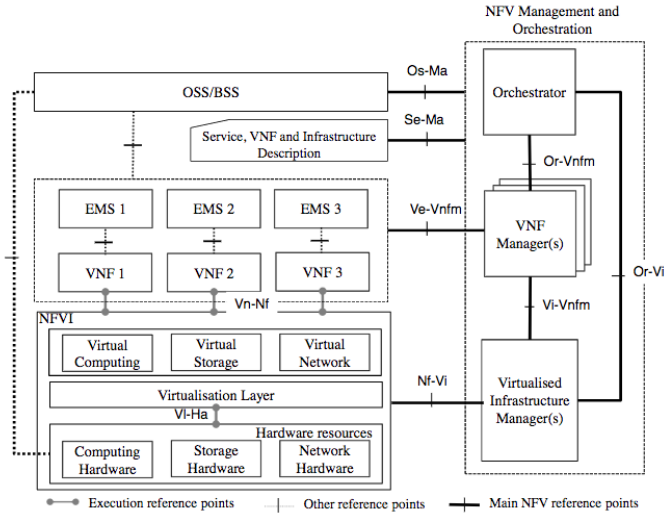


Fig. 2.2. The NFV reference architecture diagram from the European Telecommunications Standards Institute (ETSI) specifications (source [18])

As seen in figure 2.2, NFV interacts between COTS resources and VNF, controlled by the Element Management System (EMS) which at the same time is controlled by the MNO's Operations Support System/Business Support System (OSS/OBS). It is the NFV Management and Orchestration (MANO) task to orchestrate, provisioning VNF with resources during their life cycle.

NFV provides a number of advantages, CAPEX- and OPEX-wise, which revert in MNO 5G investments. Allows to minimize the costs of dedicated hardware expenditure in favor of COTS hardware. This also makes it great in terms of provisioning since there's no need for it. It also benefits power consumption as well as cooling, since several VNF can run from a single server. There's a total decoupling between software and hardware on physical networks breaking any compromise with End-of-Life (EOL) product lifecycle or its vendors.

[1] A vertical market is a market that englobes products or services in a particular industry.

The SDN concept goes hand in hand with NFV. SDN represents an architectural model in which networks use software or application programming interfaces to manage networking traffic and communicate with the underlying hardware infrastructure. This is a different approach than traditional networks which employed dedicated hardware devices to direct networking traffic. SDN is not tied up only to virtual networks, it can also control traditional hardware networks using software.

1.1.2. Edge Computing

5G offers advanced services for technical and business innovations through virtualization of the network functions. A motivation to implement cloudification, facilitating intelligent services which in turn optimizes connectivity, latency and other key performance indicators, in which vertical markets (e.g. energy, food, agriculture, manufacturing, automotive, etc) can use it to boost their businesses.

5G heavily relies on data centers, such centers may be centralized located or at the network's edge, closer to the end user. This forces MNO or independent service providers to forecast for server capacity, data storage, equipment cooling, etc.

Virtualized services run in software environments where COTS hardware is being used, evolving in such a way that BS processing power can and does take place in the cloud, leaving BS to overtake the cloud processing power when cloud's reach a certain threshold power limit.

A main advantage of EC centers in 5G is that it minimizes the latency because of the proximity between data centers and users as depicted in figure 2.3. Some of the data to be processed may be offloaded, at 5G data rates, from the edge into the user equipment (UE) and be processed in situ by the UE. All these characteristics facilitate some of the 5G specification standards depicted in figure 2.1.

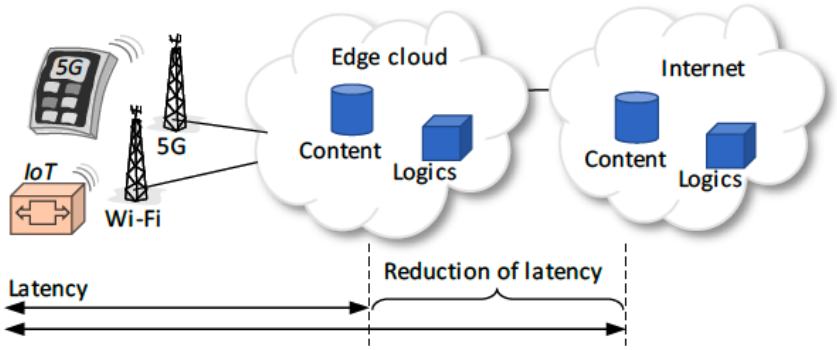


Fig. 2.3. Edge cloud (source [15])

EC centers are offering high profitable opportunities to established and new stakeholders. Opens the door for MNO to lease their cloud processing and capacity to third parties, as an Infrastructure-as-a-Service (IaaS) business model. Even the virtualization of the 5G itself generates a completely new level of opportunities. The MNO infrastructure can be composed of purchased and owned services, into account of deployment areas and the potential reaching on optimized Return Of Investment (ROI). The latter, brings up the term Service Level Agreement (SLA) which guarantees the service up-time, forcing the MNO or the service provider to adopt geo-redundant services dependending on how demanding the service is.

1.2. 5Growth

The present master thesis has been developed aimed at next generation networks, taking as context the 5Growth framework.

5Growth is a Horizon 2020 Infrastructure Public Private Partnership 5G-PPP phase-3 project that aims to validate the operating of 5G systems deployed on vertical industries and optimize the performance, empowering automation and AI driven solutions. Launched in 2019, is led by a consortium of 21 partners[2] across several European countries and it is planned to last until the February of 2022 with a total budget of 14M€.

1.2.1. 5Growth architecture

The 5Growth inherits the baseline architecture from 5G-Transformer [5], a phase-2 from the same 5G-PPP project mentioned earlier, enhancing usability, flexibility, automation, performance and security. The architecture englobes three main blocks as seen in figure 2.4:

5Gr-VS (5Growth Vertical Slicer): It is the storefront for vertical requests. A Vertical Service Blueprint (VSB) catalog allows verticals for provisioning services through an oriented interface with the OSS/BSS, employing a template as a starting point. Then, vertical requesters can tailor their necessities by specifying the demanding service-oriented parameters. The 5G-VS is in charge for managing the requested vertical service and the network slices, created by affording NFV network services requested to the Service Orchestrator (SO).

[2] Among the consortium partners is the *Centre Tecnològic de Telecomunicacions de Catalunya* (CTTC), the responsible entity that has supervised this present thesis.

5Gr-SO (5Growth Service Orchestrator): This core block provides network service and resource orchestration capabilities. Orchestrates NFV-Network Services (NFV-NS) either to a single or multiple domains, managing their life-cycle. It also serves the 5Gr-VS with a view of the services, since it receives the service requirement in the form of a Network Service Descriptor (NSD) describing the service requirements. It is the 5Gr-SO that, based on availability and resource capabilities, among other administrative domains, informed by the 5Gr-RL, decides the optimal configuration for the NFV-NS. Monitoring tasks take place at the 5Gr-SO for self-adaptation in front of demanding events, in order to avoid any SLA violation.

5Gr-RL (5Growth Resource Layer): Responsible for managing local and transport resources. It provides resource orchestration and VNF instantiation as well as control over the physical transport, network, computing and storage infrastructure where network slices and services are executed.

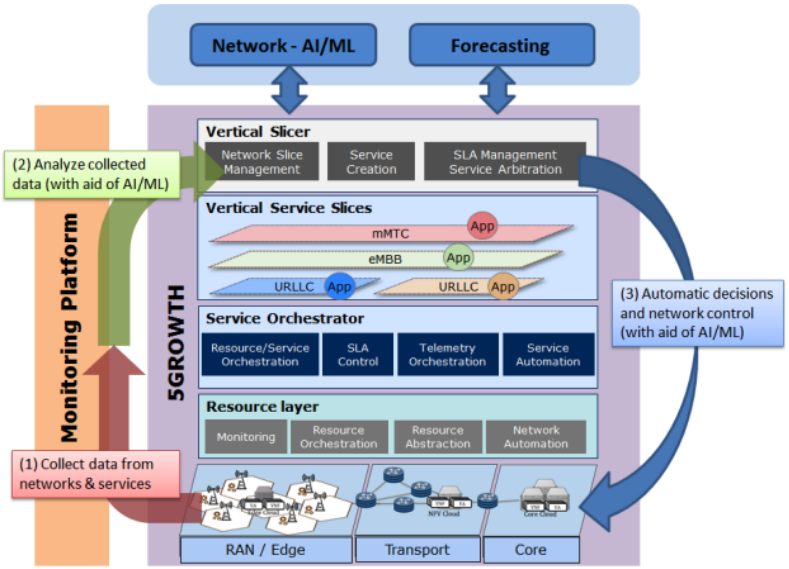


Fig. 2.4. 5Growth architecture framework(source [17])

1.2.2. AI/ML Platform (5Gr-AIMLP)

The Artificial Intelligence/Machine Learning platform (5Gr-AIMLP) [5] is, together with RAN support and Vertical-oriented Monitoring System (5Gr-VoMS), the main architectural innovations on 5Growth. The 3GPP TS 23.501 version 16.6.0 Release 16 [6] introduces mechanisms in order to automate the 5G core using ML and data analytics. One of the entities that allows for it, is the Network Data Analytics Function (NWDAF).

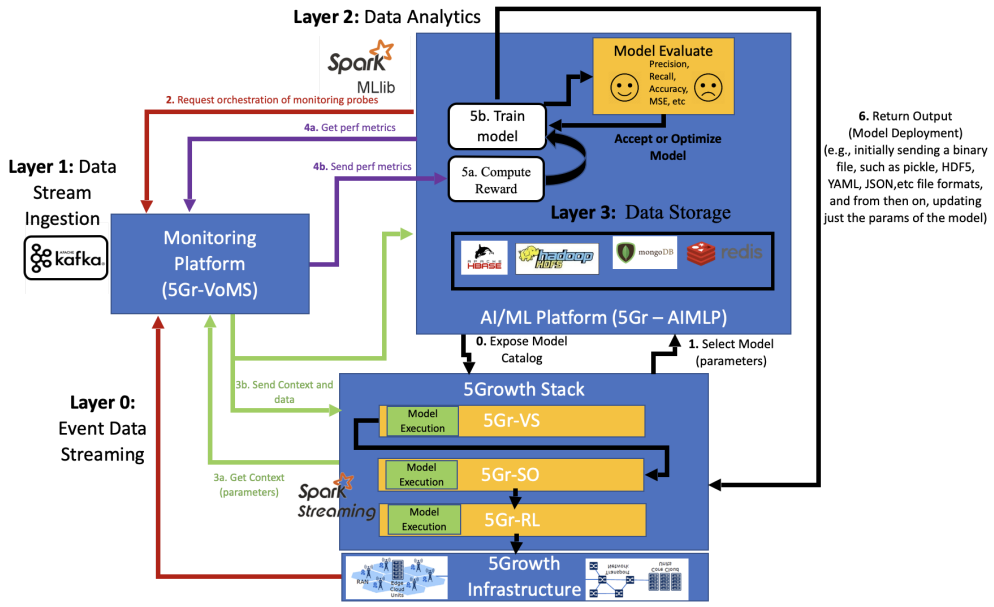


Fig. 2.5. 5Gr-AIML platform workflow (source [16])

The AIMLP presents several trained models[3] susceptible to be tuned or generate new ones from the composite of existing ones. The entity in need of decision making, in the literature known as the agent, will determine the needed ML model along with the parameters that the model must contemplate and what data needs to be monitored in the system. Through the 5Gr-VoMS, the AIMLP will require the data that must be monitored. For RL models, the agent obtains precise input model data from the 5Gr-VoMS. Such data is then also used by the AIMLP in order to optimize the model parameters. Once the model is tailored, it is passed to the agent to be executed with performance metrics coming from the 5Gr-VoMS.

This master thesis takes the 5Growth framework as reference at a general level and focuses on AI-related algorithms that could be run by the 5Growth architectural entities to take decisions on VNF placement according to certain operational requirements (e.g., guaranteeing an efficient use of edge resource while maximizing the number of deployed services over the shared infrastructure).

1.3. What is Reinforcement Learning?

1.3.1. Introduction

RL is a subfield of ML, an AI paradigm that mimics the human learning procedure based on behavioral psychology, concretely in operant conditioning. A key aspect

[3] A ML model is a file that has been trained to recognize certain patterns.

describing a learning method which employs rewards and punishments as a behavior result.

Back in 1951, Marvin Minsky (1927-2016), regarded as one of the fathers of AI, created a computer that continuously learned to solve a virtual maze. Such a computer was inspired by physiologist Ivan Pavlov (1849-1936), well known for using dogs to show how animals learned by positive (reward) and negative (punishment) reinforcement.

Humans gain knowledge with experience and outcomes. A child might learn that pushing a chair and climbing it to reach a cupboard will lead him to achieve a candy bar resulting in an immediate reward, but in the long term the kid will also learn that its parents' punishment will be less sweet, hence, the child will end up learning to ask for the sweets to its parents rather than taking it whenever pleases.

1.3.2. Reinforcement Learning (RL)

How do we transfer a human learning procedure into a computational approach?.

Let us define some formalisms concerning RL:

Agent: an entity that learns to take optimal actions by interacting with the environment in exchange for a reward. Such agents must be able to take into account each particular environment state.

Environment: the set of states that compound the space where the agent interacts with. The environment not only will accept the agent's actions but will also respond with the relevant agent's reward (positive or negative).

Action: each decision the agent makes when faced with an environment state.

State: represents the environment status after the agent has taken an action or when resetting the environment for the first time.

Reward: an environment feedback mechanism, in the form of a scalar, that lets the agent know how valuable its taken actions results. The reward encodes the challenge in such a manner that motivates the agent to learn the optimal action at each state.

In a RL scenario we encounter the agent which learns and interacts with the environment as shown in figure 2.6.

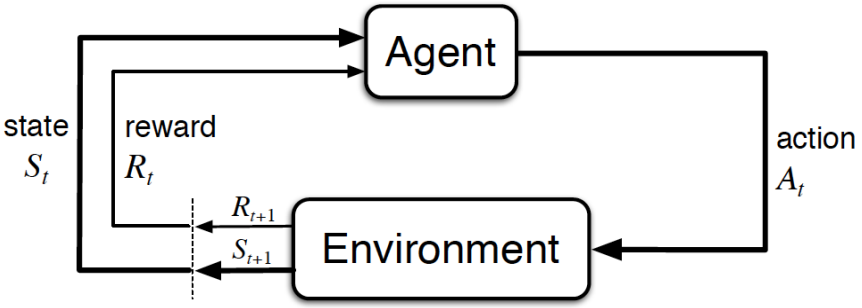


Fig. 2.6. Agent/Environment interaction (source [8])

The dynamics are as it follows:

- Each interaction between the environment and the agent is generated at discrete time steps; $(t = 1, 2, 3, \ldots)$.
- As a consequence of the agent's previous action (A_{t-1}) that lead it to the current state (S_t), it receives a reward from the environment (R_t).
- Based on what the agent perceives from the current state (S_t), selects an action (A_t), out of the set of possible actions; $A_t \in \{a_1, a_2, \ldots, a_n\}$. Such action provokes the environment to transition to a time step later state (S_{t+1}) and pass on the agent with a new reward (R_{t+1}).

RL consists of an agent in an environment, whose dynamics are unknown to it, to take the optimal decisions when confronted with a complex problem by means of trial and error. It is not told what actions to take, rather learn by itself which are the optimal chain of actions to accomplish the final goal, by maximizing a cumulative numerical reward. The agent learns which actions yield more profit and which don't. It is important to highlight that each taken action will affect all subsequent rewards, i.e., an action may lead to a highly rewarded state but neither of all possible subsequent actions will allow the agent to obtain a reward, whereas, if the agent would have chosen a previous less rewarded action, perhaps it would have allowed it to land onto a different state with higher rewarded actions.

1.3.3. Exploration and Exploitation dilemma

One of the RL challenges arises from the correlated actions for subsequent rewards, the dilemma between exploration and exploitation which, nowadays, remains unresolved. "*The agent has to exploit what it has already experienced but it also has to explore in order to make better action selections*" [7]. The latter is of relevant importance given that in stochastic scenarios, the agent must try all actions several times to gather reliable estimation of the expected reward.

1.3.4. Markov Decision Processes

Markov Decision Processes (MDP) model controlled stochastic dynamic systems, that is, systems whose evolution is subject to random factors and which can be modified by means of certain decision or control variables selection. MDP can describe a RL environment providing a mathematical model for learning sequential decision making, where actions not only provide immediate rewards but also subsequent states.

The environment must be fully observable and each current state characterises the process. Mathematically speaking, an MDP is a tuple of:

$$\Big\langle S, A, P, R, \gamma \Big\rangle \tag{1.1}$$

- S finite set of states
- A finite set of actions
- P state transition probabilities
- R reward function
- γ discount factor

1.3.4.1. Markov property

From figure 2.6, it is intuited that consecutive time steps raise a sequence of states, actions and rewards known as the *History*. Each state encapsulates all relevant information for the agent in order to take an action. Furthermore, the future state will be determined only by the current state where the agent is at a time step (t), and not by virtue of the past visited states; $\{s_1, s_2, s_3, \ldots, s_t\}$. This is the main concept of an MDP and it is known as the Markov property. A state (S_t) will be considered to accomplish the Markov property if and only if:

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, S_2, ..., S_t] \tag{1.2}$$

1.3.4.2. State transition probabilities

A Markov Process is a random process, and so are the random variables (R_t) and (S_t) which are defined by discrete distributions probabilities that will depend on the agent's action selection from the previous state. Therefore, there must be a probability for those random variables to occur at each time step that implicitly

defines a transition probability from state to state. Such probability can be defined as:

$$\sum_{s' \in S} \sum_{r' \in R} p(s', r|s, a) = 1, \forall\ s \in S, a \in A \tag{1.3}$$

The function p inherits the Markov property previously exposed and encloses the mathematical definition for an MDP. The next state (s') and reward (r) depends only on the precedent state (s) and action (a) taken. The state transition probability defines the MDP dynamics.

1.3.4.3. Reward function

The aim of the agent is to maximize the cumulative rewards until reaching its goal, in the literature is known as the *expected return* (G_t).

Defining the *reward function* (R_s) as the immediate reward an agent is envisaged to get from the state where is at:

$$R_s = \mathbb{E}[R_{t+1}|S_t = s] \tag{1.4}$$

allows for the *expected return* to also be characterized as a chain of expected immediate rewards, a Markov Reward Process:

$$G_t = R_{t+1} + R_{t+2} + ... + R_T \tag{1.5}$$

where T denotes an absorbing state, a terminal state when the agent achieves its goal or commits some error that forces the environment to restart again. Thus, we referred to such scenarios as *episodic tasks*, since the chain of consecutive states form an *episode*. Not all MDPs must have a terminal state as will be seen in chapter 2.4.3 when exposing the development of the present master thesis; such cases are known as *continuing tasks*.

What prevents an agent from returning infinitely to a state aiming to accumulate rewards, is the *discount factor,* $\gamma \in [0, 1]$, a value that determines how worth an immediate reward is. Hence, we can rewrite the *expected return as:*

$$G_t = \gamma^0 \cdot R_{t+1} + \gamma^1 \cdot R_{t+2} + \gamma^2 \cdot R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k+1} \tag{1.6}$$

When $\gamma \approx 1$, the agent will prioritize the expected future rewards rather than the immediate ones. On the contrary, if $\gamma \approx 0$, the agent will strongly take into account the immediate rewards.

1.3.4.4. Value Function

Up to now key elements for a RL scenario have been defined, nevertheless, there still is a need for a linkage between the agent's action and the state that transitions to, in terms of how good it is for the agent to be in a particular state. The (G_t) the agent might get for being in a certain state rather than other is computed by the *state-value function,* $(v(s))$:

$$v(s) = \mathbb{E}[G_t|S_t = s] = [\sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k+1}|S_t = s],\ for\ all\ s \in S \quad (1.7)$$

a function that pairs states to rewards which implicitly implies a relation between states and actions.

Another question that might arise is, how to compute the G_t if it depends on future rewards. Answer to it is, having the agent iterate over the state space and learn to map states to actions in order to gather the largest *expected return*.

The *value function* can be decomposed recursively into two parts and this particularity holds as a fundamental relationship property to introduce the *Bellman Equation,* an equation that relates the value of a given state and its successors states value:

$$\begin{aligned} v(s) &= \mathbb{E}[G_t|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ...|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + ...)|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma G_{t+1}|S_t = s] \\ &= \mathbb{E}[R_{t+1} + \gamma v(S_{t+1})|S_t = s] \end{aligned} \quad (1.8)$$

distinguishing:

- The immediate reward: R_{t+1}
- The discounted value of next state: $\gamma v(S_{t+1})$

An easy way to interpret the *Bellman Equation* is as a one step look ahead (backup diagram) from state (s) where we average all possible outcomes (s') together giving us the *value function* $v(s)$ from state (s).

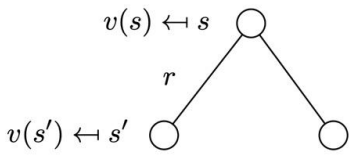


Fig. 2.7. Backup diagram for $v(s)$ (source [8])

Note: A Bellman equation process on a simulated environment is detailed in annex A.

1.3.4.5. Policy

It is clear that the aim for any RL scenario would be to implement an agent that generates the largest (G_t) accomplishing its goal in the minimum time steps. Such an agent's strategy is called *policy* (π) and it is under the agent's control:

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = s] \tag{1.9}$$

Equation 2.7 can also be interpreted in terms of (π) giving place to a *state-value function* definition*:*

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k \cdot R_{t+k+1}|S_t = s], \ for\ all\ s \in S \tag{1.10}$$

It can recursively be decomposed into the same elements as in equation 2.8 to obtain the *Bellman Equation* in terms of (π) as stated by Sutton R. S and Barto A. G. in their work [8]:

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \cdot [r + \gamma v_\pi(s')], \ for\ all\ s \in S \tag{1.11}$$

describing how good it was for the agent to follow its policy in a given state, taking an action and transition to another state. The immediate reward plus the

state-value function of the landed state determines how good the original state was.

There's another type of *value function* that rather than mapping states to rewards, relates states to actions, therefore it is called *action-value function* $q(s,a)$ and in terms of (π):

$$q_\pi(s,a) = \mathbb{E}[G_t|S_t = s,A_t = a] \tag{1.12}$$

can also be decomposed in order to obtain the *Bellman Equation:*

$$q_\pi(s,a) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a] \tag{1.13}$$

Equation 2.13 is describing the (G_t) starting from state (s), choosing action (a) and following policy (π).

Notice how the *state-value function* and the *action-value function* are related, both representing the same concept but at different levels of granularity. Whereas the *state-value function* denotes the expected reward averaged over all actions, the *action-value function*, at a higher resolution, provides which actions and states are the optimal ones.

1.3.4.6. Optimal policy

It is the agent's mission to find an optimal policy (π_*) with the purpose of expecting the maximum return. Out of all possible policies, there's at least one to be better or equally as good to all other policies[4] achieving the optimal *state-value function* and the optimal *action-value function*:

$$v_{\pi_*}(s) \geq v_{\pi'}(s), for\ all\ s \in S \tag{1.14}$$

$$q_{\pi_*}(s,a) \geq q_\pi(s,a), for\ all\ s \in S, a \in A \tag{1.15}$$

[4] The total number of policies can be computed by the number of the action space (A) to the power of the state space (S) number.

An agent will be able to determine an optimal policy by iterating over the state space and updating the state values from maximizing over $v_\pi(s)$ or $q_\pi(s,a)$:

$$v_*(s) = \max_\pi(v_\pi(s)) \tag{1.16}$$

$$q_*(s,a) = \max_\pi(q_\pi(s,a)) \tag{1.17}$$

1.4. Dynamic Programing

In most cases, obtaining the optimal solution in a RL scenario is an unresolvable challenge due to processing and memory computational costs. Not in all RL environments, an optimal solution to Bellman's equation can be computed. Sometimes the only way is an approximation to it. For some tasks, the state space is so large that the agent might not even face some of those states ever, hence not computing a state-value function nor an action-value function for such states. It should be recalled, as mentioned previously, that a way to obtain an optimal value function is by having the agent iterating through the state space.

In some RL projects, solutions can be expressed recursively in mathematical terms by means of a recursive algorithm. However, as previously discussed, the execution time of the recursive solution, normally of exponential order and therefore impractical, can be substantially improved by means of Dynamic Programming (DP). DP is a convenience method when the environment is fully known and the number of the state space does not reach large numbers. In such cases, the literature refers to it as *Model-based*, since all environment aspects are known; S (set of states), A (set of actions), P (state transition probability), R (reward function) and γ (discount factor).

Algorithms as *Policy iteration* allow to find optimal value functions that satisfies the *Bellman* optimality equations:

$$v_*(s) = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma v_*(s')] \tag{1.18}$$

or:

$$q_*(s,a) = \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q_*(s',a')] \tag{1.19}$$

1.4.1. Policy Iteration

Policy Iteration algorithm is structured in two sub-algorithms; *Policy Evaluation* and *Policy Improvement.* The first, computes the state-value function $(V_i(s))$ for a given policy $(\pi_i(s))$. The second, improves the previous given policy $(\pi_i(s))$ obtaining a new policy $(\pi_{i+1}(s))$.

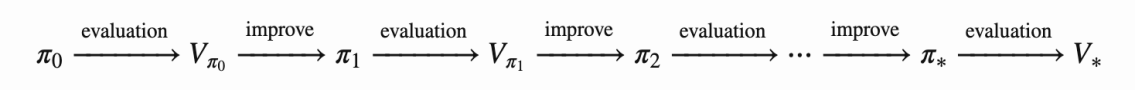


Fig. 2.8. Policy Iteration algorithm process (source [8])

Policy Iteration algorithm on a Model-based MDP computes the optimal agent's policy by treating the Bellman's equation as a linear contractive operator.

Sutton and Barto in [8] detail the Policy Iteration algorithm steps in order to estimate $\boldsymbol{\pi} \cong \boldsymbol{\pi}^*$ as:

1. Initialization

$V(s) \in \mathbb{R}$ *and* $\pi(s) \in A(s)$ *arbitrarily for all* $s \in S$

2. Policy Evaluation

Loop :

$\Delta \leftarrow 0$

Loop for each $s \in S$:

$$V(s) \leftarrow \sum_{s',r} p(s', r|s, \pi(s))[r + \gamma V(s')]$$

$\Delta \leftarrow max(\Delta, |v - V(s)|)$

Loop until $\Delta < 0$

3. Policy Improvement

$policy\_is\_stable \leftarrow True$

For each $s \in S$:

$old\_action \leftarrow \pi(s)$

$$\pi(s) \leftarrow argmax_a \sum_{s',r} p(s', r|s, a)[r + \gamma V(s')$$

$If\ old - action \neq \pi(s)$

$policy\_is\_stable \leftarrow False$

$If\ \ policy\_is\_stable$

Exit loop and return $V \approx v_*$ *and* $\pi \approx \pi_*$

else back to step 2. Policy Evaluation

Note: A Policy Iteration example is described in detail in annex B.

Note: A demonstration of linear contractive operators is described in annex C.

1.5. Q-Learning

Sometimes not all environment aspects are known as in model-based MDPs. Such cases are named by the literature as *Model-free* environments, since there's no need to know the state transition probabilities nor the reward function. The agent must learn from its own actions without a given policy, therefore it is also an *Off-Policy* model.

Q-Learning algorithm (C.J.C.H. Watkins 1989) [9] thrust the agent learning by assigning values to state and action pairs. A matrix, as a data structure, is used for the assignment, where rows represent each possible environment state and columns hold the Q-value for each possible action within the state. Q-values define how good an action is at a given state for the agent. Is the expected cumulative discounted reward for being in state (s) and taking action (a).

If all the Q-values were known a priori by the agent, it could use the information to make the appropriate decisions at every state, nevertheless, this is not the case at the beginning of an episode since it lacks such information. It is the agent's main goal to achieve this assignment as closely as possible. As Q-values depend not only on future rewards but also on current ones, it is needed to provide a method able to compute the final Q-value from immediate and local values.

If an action at a state causes an undesired result, it must be learnt not to take the same decision in that state. On the contrary it will be learnt to take the same action in that state whenever the result of doing so draws a positive result.

If all possible actions within a state yield a negative outcome, it will be convenient to avoid that particular state. That is, the agent must not take any actions from any other possible states that could transition it to that particular state. On the other hand, if all possible actions within a state result in a positive payoff, it will be a must to learn how to reach that particular state. As a matter of fact, this is what allows the propagation of state and action pairs rewards to adjacent states pairs.

Q-values can be computed by means of:

$$Q(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma max_{a_t+1}(Q(s_{t+1}, a_{t+1})) \tag{1.24}$$

Similarly, as explained in chapter 1.3.4.3, $\gamma \in [0, 1]$ will set the weight for future rewards. The sum of the immediate reward $(r(s_t, a_t))$, being in state (s_t) and taking action (a_t), and the best discounted Q-value $(\gamma max_{a_t+1}(Q(s_{t+1}, a_{t+1}))$ to be achieved, allows to obtain the optimal Q-value. In order to approximate a fixed point where all values converge, it is needed to initialize all the Q-values to a fixed random value (e.g. 0). Afterwards the agent will update the stored Q-value pairs (state, action) by iterating through the states considering as true the annotations taken from other pairs, some of which will have been approximated in previous steps.

1.5.1. Q-Learning (off-policy Temporal Difference (TD))

Equation 2.24 updates the Q-values somewhat abruptly in certain directions. By introducing a learning rate factor $(\alpha \in [0, 1])$ it is possible to control the variation of Q-values. The learning rate (α) defines in which grade the agent overrides the old data with new one. A factor $\alpha \approx 1$ will force the agent to take into account the most recent information, whereas $\alpha \approx 0$ will make the agent learn nothing.

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\Big(\underbrace{\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}}}_{\text{new value (temporal difference target)}} - \underbrace{Q(s_t, a_t)}_{\text{old value}} \Big)}^{\text{temporal difference}} \tag{1.25}$$

The new Q-value $(Q^{new}(s_t, a_t))$ is the weighted combination of the old Q-value and the new information that the agent must believe. The agent must iterate over the states to refine the optimal Q-value by updating the previous estimate.

$(r_t + \gamma max_a Q(s_{t+1}, a))$ could be interpreted as a Temporal Difference (TD) algorithm (one-step look-ahead) allowing the rest of the equation to project one step forward so the agent can forecast future trajectories for optimal outcomes. The (max) function is scanning over all Q-values for each possible action for the future state from the current state.

Q-Learning (TD) algorithm will converge to an optimal Q^* value allowing sufficient learning rate (α) and exploration throughout the environment states that satisfies the Robbins-Monro (Robbins and S.Monro, 1951) conditions [10][11].

Sutton and Barto in [8] also details a Q-Learning (off-policy TD) algorithm for estimating $\boldsymbol{\pi} \cong \boldsymbol{\pi}^*$:

$\alpha \in (0,1)\ and\ \epsilon > 0$
$Initialize\ Q(s,a)\ for\ all\ s \in S\ , a \in A(s)$
$Loop\ for\ each\ episode:$
$Reset\ environment$
$For\ each\ step\ of\ episode:$
$Choose\ action\ from\ state\ using\ greedy\ algorithm\ (\epsilon)$
$Receive\ r\ and\ observe\ s'$
$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \gamma max_a Q(s',a) - Q(s,a)]$
$s \leftarrow s'$
$Until\ s\ is\ terminal\ state$

1.5.2. ϵ-greedy algorithm

A drawback that Q-Learning encounters is that its agent can only learn from those updated Q-values from actions taken in visited states. There's no learning from the untaken actions. This brings us the Exploitation/Exploration dilema introduced in section 1.3.3. It might be interesting for the agent, specially at the beginning of its training[5], to adopt an explorer profile in order to visit as many states as possible and try as many actions as possible from those states. As the agent learning process reaches a certain level it is more advantageous to exploit those known actions that yield a higher reward when revisiting the known states.

Let us formalize the above discussed in probability terms. It can be determined that the higher the Q-value of an action, the higher the probability of being chosen. Even low Q-value actions may be eligible at the beginning of the training. It may also be convenient to align the exploration probability with the learning time the agent has taken so far. At the beginning, exploration is motivated, since what the agent could have learnt is not yet reliable due to the amount of possible choices that are left to try. As the agent learning progresses and time passes by, it might be more convenient to promote the exploitation from the known actions at known states. At this point the agent's knowledge is more trustful comparing it as it was at the very beginning of training. Sancho F. C. (2/3/2019) [12] summarizes the above in the following equation:

$$p(a_t|s_t) = \frac{e^{\epsilon \cdot t \cdot Q(s_t,a_t)}}{\sum_{a \in A_{st}} e^{\epsilon \cdot t \cdot Q(s_t,a_t)}} \tag{1.26}$$

[5] In Q-Learning there's no training concept as such, as there's in other ML fields (e.g. Recurrent Neural Networks, Regression models, etc). Nevertheless, we refer to training for the first episodes attempts where the agent is populating its Q-Table.

- ϵ is an exploration constant
- A_{st} is the set of actions that are possible on each state s_t
- t represents each time step
- $Q(S_t, a_t)$ is the current state Q-value

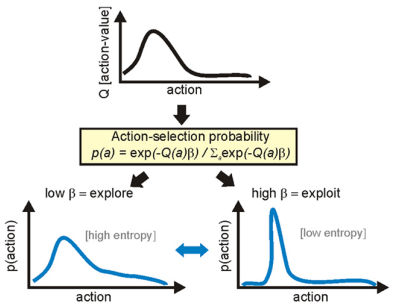


Fig. 2.9. Exploration and exploitation trade-off formalization (source [12])

As t increases due to the exponential function, actions with the highest Q-value are more prompted to be chosen.

A mathematical function can be described to represent the exploration-exploitation trade-off:

$$e\_e\_tradeoff \leftarrow random\ uniform\ number\ \in [0, 1] \tag{1.27}$$

$$a_t = \begin{cases} max\ Q(s_t, a_t) & if \quad e\_e\_tradeoff\ > \epsilon \\ random \quad a_t \in A & if \quad e\_e\_tradeoff\ < \epsilon \end{cases} \tag{1.28}$$

1.6. Summary

Chapter 1 has introduced the fundamentals to understand the context in which the proposed algorithms are going to be implemented. It also has described the theory aspects to understand how reinforcement learning works and has distinction between model-based and model-free scenarios, a very important consideration to take into account when implementing an MDP and a Q-Learning algorithm. It is clear that for implementing an MDP it is a requirement to have all contents of an MDP tuple, hence it will be necessary to compute all the state transition probabilities as well as all possible valid states to tackle the master thesis problem.

CHAPTER 2. WORKING ENVIRONMENT

This chapter provides the problem statement to be addressed. Section 2.3 details all the needed modeling that is required as far as implementing the environment to

which the agent must perform. Finally, main considerations for each individual algorithm are described.

All RL algorithms have been developed in Python 3.8 and under the Open AI Gym framework. As stated in its website "*Gym is a toolkit for developing and comparing RL algorithms...*" [14]. It comes out of the box with several modeled environments for a wide variety of RL applications. It also allows one to model a customized environment, as in the present thesis.

2.1. Problem statement

Mobile networks, integrating cloudification capabilities have to be flexible enough since the increasing user equipment data demand, has created the necessity to optimally manage resources at the network edges so as to adapt to the required services. MNO infrastructure requires monitoring the network status at all times to prevent failures.

A distributed edge cloud structure provides redundancy for service deployments in case of potential edge point failure but it also needs an optimal resource management to not trifle away unused resources, hence losing potential profits.

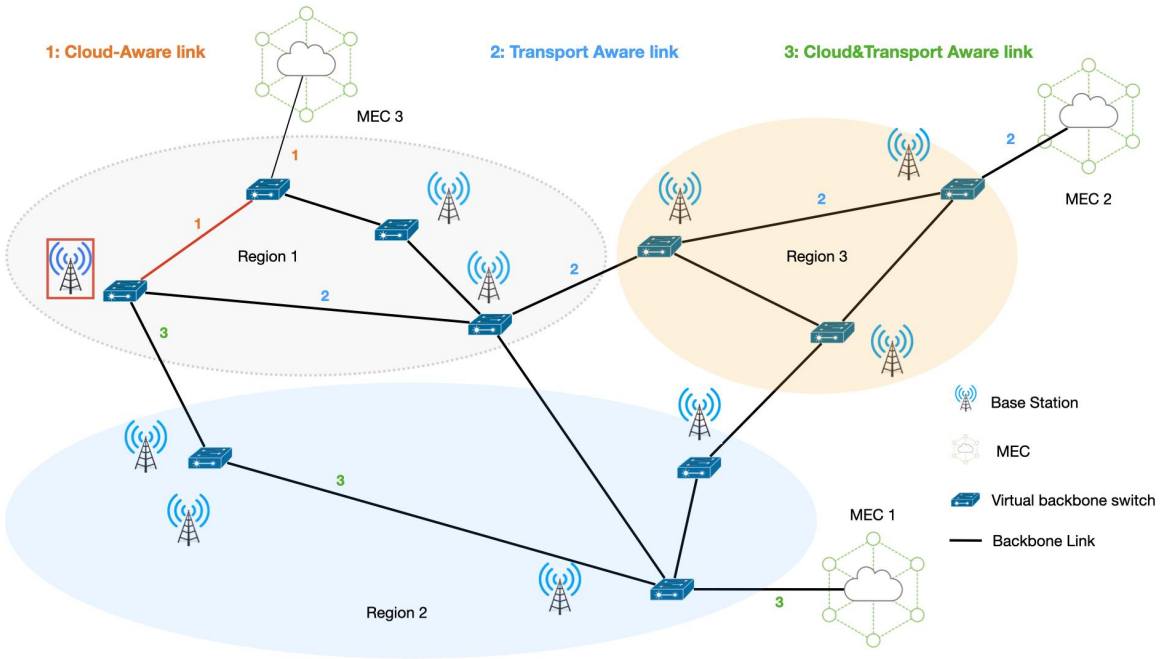


Fig. 3.1. Edge Computer nodes across different network regions

Whenever edge cloud Life Cycle Management (LCM) and transport Key Performance Indicators (KPI) are managed separately [13], it will be harder to provide scalability for the MNO. When running NFVs, LCM and transport KPI can

be managed from the Network Function Virtualization Orchestrator (NFVO), which allows modification of services without direct request from the OSS/BSS. Moreover, the NFVO is able to react to network events and decide what parameters to modify for the changing network event related service. The latter can be achieved by means of containers (e.g. Kubernetes) although such containers are not aware of other mobile network regions KPI's.

The figure 3.1 depicts a faulty EC backbone link (red link) precluding one of its nearby affected (red) base stations (BS) to be served for any requested service. By only considering cloud or transport aware parameters the requested service can not be satisfied. The selection of the EC center needs to depend on both cloud and transport parameters. A ML model could joint transport and cloud parameters for EC selection and even foresee having an EC running out of its resources or a link reaching its limit bandwidth.

2.2. Followed strategies for the thesis development

The aim of the present work is to implement a RL algorithm, more precisely a Q-Learning (off-policy TD) algorithm, to optimally allocate VNF requested by BSs to an EC center. RL is a good technique to solve complex challenges achieving long term results. The learning procedure is very similar to the human's making it ideal to reach almost perfect solutions. In RL when the agent learns to avoid an error, it is very unlikely to commit the same error. The agent can learn from its experiences.

There are other tabular RL algorithms like SARSA but it is an on-policy based algorithm, hence it needs a policy to estimate the value of the current policy whereas Q-Learning directly learns the optimal policy. SARSA on the other hand might show some conservative performance whenever the risk of gathering a large negative reward is close to the optimal policy path.

In order to accomplish a close scenario as stated in 2.1, the environment requires it to be custom modeled with specific dynamics to interact with the agent.

In order to evaluate the Q-Learning algorithm performance, two more algorithms have been implemented to be compared and extract final conclusions from the obtained results.

The first of the algorithms (Best Fit) has been inspired by the concept of a classic load balancing algorithm, Weighted Round Robin, adapted to the needs of the treated thesis matter. Each server has a weight based on the administrators criteria of their choosing. The server with the highest weight serves the client petitions. This approach follows a classical problem of client-server petition, setting aside the

differences. It increases efficiency and reduces node downtime. A load balancer algorithm improves resource utilization while also avoiding some nodes over load.

The second implemented algorithm (Model-based MDP) is a Policy Iteration algorithm using iterative policy evaluation for estimating the optimal agent's policy. For it, all state transition probabilities need to be computed in advance, as well as all possible valid states. This is an algorithm that allows one to obtain the optimal policy out of all possible policies, it is a way to compare how good the Q-Learning algorithm performs since the optimal policy can be achieved through Policy iteration.

Finally, a Q-Learning (model-free and off-policy Temporal Difference (TD)) algorithm is implemented, where not all environment key aspects are known and the agent will learn its optimal policy by means of trial and error, reaching its goal by maximizing the discounted cumulative state rewards from the environment.

The main objective is to have three algorithms performing under the same environment dynamics where an agent receives a reward for every optimal decision it takes on each state. By having the same environment it can be assured that the agent side is interacting with the environment exactly the same way across all algorithms, hence the agent's policy can be evaluated objectively without the environment's influence.

2.3. Modelings

2.3.1. Modeling the network

To tackle the problem stated in chapter 2.1 the following network structure is considered.

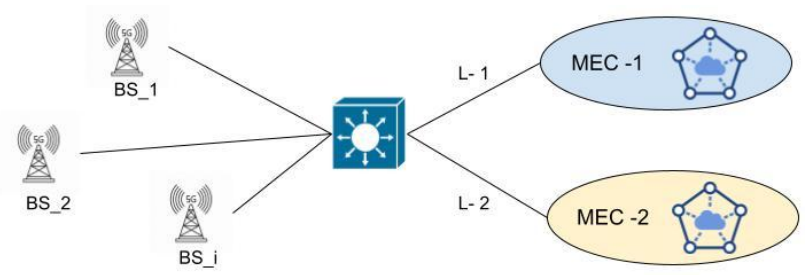


Fig. 3.2. Mobile Network infrastructure to be modeled

Such a scenario allows us to be able to model the problem from a theoretical perspective across all three algorithms and compare it with an optimal solution to see how close we are from it.

Two ECs network nodes will be modeled, $M1$ and $M2$[6]. Each EC will be defined by:

- Available number of Central Processing Units (CPU): $M1_{fcpu}$, $M2_{fcpu}$
- Number of used CPUs: $M1_{ucpu}$, $M2_{ucpu}$
- Available link bandwidth (BW): $M1_{fBW}$, $M2_{fBW}$
- Used link bandwidth: $M1_{uBW}$, $M2_{uBW}$

Therefore, at any time, the network status can be described by the following vector:

$[M1_{fcpu}$	$M2_{fcpu}$	$M1_{ucpu}$	$M2_{ucpu}$	$M1_{fBW}$	$M2_{fBW}$	$M1_{uBW}$	$M2_{uBW}]$

Table. 3.1. Network status vector

2.3.2. Modeling VNF requests

VNF requests are generated as a sequence of requests, each with its interarrival time as well as duration time and stored in a .csv file to be imported by the environment side.

A BS VNF request is compounded by:

- Arrival rate
- Departure rate
- VNF type

The VNF arrival rate (λ) defines the average number of VNF that a BS may request per unit of time. Whereas the departure rate (μ) defines the average number of VNF that are terminated from the system per unit of time. Both rates follow a Poisson distribution, many events follow such a distribution; like the number of patients arriving into a hospital emergency room, the number of failures of ultrasound machines for cleaning purposes or the number of requests to a server. A Poisson distribution describes the number of event occurrences, it is a stochastic and discrete process and its events are independent from each other.

[6] For convenience purposes M is employed to refer to mec, but from ahead now, M and mec will mean the same.

In spite of a Poisson distribution models the number of event occurrences, the interarrival time of such events can be modeled using a continuous exponential distribution.

Finally, the VNF type describes the different types of requests. Details the requested number of CPUs and BW for the VNF service.

$[request_{cpu}$	$request_{BW}]$

Table. 3.2. VNF request vector

Note: Annex D describes the modeling process for VNF requests.

2.3.3. Modeling the environment

The environment must be the same for all algorithms, that is, the interaction signals between the agent and the environment must be the same, as depicted in figure 2.6.

From modeling the network, it is clear that the possible actions are two, M1 and M2.

$$A_t \in \{mec1, mec2\}$$

The state must contain all the network information relevant to the agent, it needs to reflect not only the network status but the VNF request that arrives into the network to foresee the EC assignment.

$S_t \longleftrightarrow [request_{cpu}, request_{BW}, M1_{fcpu}, M2_{fcpu}, M1_{ucpu}, M2_{ucpu}, M1_{fBW}, M2_{fBW}, M1_{uBW}, M2_{uBW}]$

For the reward function, several strategies can be implemented. The agent could be positively rewarded when assigning a VNF to a EC, or be negatively rewarded when a VNF rejection occurs, or be both. During the development stage of Q-Learning, rewarding the agent positively and negatively, showed to diminish the agent's performance. It was decided to only reward the optimal actions to enhance the optimal policy. It should be recalled that rewards are reflected in the Q-Table in the form of state-action pair values that are constantly updated with each action. Having two policies, an optimal one and one to be avoided results in too much information for the agent to learn.

$$R_t = \begin{cases} 1 & \text{if } VNF \text{ served} \\ 0 & \text{if } VNF \text{ rejected} \end{cases}$$

The environment is also in charge of triggering each VNF request according to its interarrival time. Also, keeps track of each VNF departure time and terminates it when required to, returning back the MEC resources to the network. After each agent action, the environment updates the network status.

2.4. Algorithms considerations

2.4.1. Best Fit

The Best Fit algorithm is inspired by a classic load balancing algorithm, Weighted Round Robin. There's no RL concept in it since the "agent" algorithm does not learn anything. It only visualizes the current network state and assigns the incoming VNF to whichever MEC shows the highest network metric value.

The metric value is what is used to weight each MEC. First the MEC link congestion is computed, for it, it is obtained as the ratio between the used link BW over the total network BW.

$$a = \frac{mec_{uBW}}{total_{networkBW}} \tag{2.1}$$

Afterwards, the final metric (k) is computed, as the ratio of the link congestion (a) times the available number MEC CPUs over 100, plus the ratio between the remaining network link BW over the number of hops between the BS and the MEC.

$$k = \frac{a \cdot mec_{fcpu}}{100} + \frac{1-a}{num_{hops}} \tag{2.2}$$

The functioning of Best Fit is straightforward, first checks if all MECs have enough CPU and BW resources, if not a rejection is generated. If there are MEC resources it will check if there's only one MEC out of all MECs with resources, if so, the VNF will be assigned to that MEC. In case all MECs have the same amount of available resources, the assignment will be random. It could also be assigned based on proximity distance to the BS. Finally if all MECs have enough resources, the VNF will be assigned to whichever MEC proves to have the highest weight (k).

Note: A detailed Best Fit algorithm flowchart is described in annex F.

2.4.2. Model-based MDP

From chapter 2.4 it is known that an algorithm like Policy iteration when applied to a model-based MDP, it is possible to compute the one and only optimal policy out of all existing policies. For it, it is necessary to know all possible valid states as well as all state transition probabilities.

One may ask, at this moment, why bother with any other algorithm (Q-Learning) when it is possible to obtain the optimal policy with Policy Iteration? A Policy Iteration algorithm becomes very slow when the number of possible valid states reaches a certain number and not always the state transition probabilities are known nor all valid possible states.

Let us remember with an example what a state looks like in Q-Learning. Let's consider an initial network status as follows:

$$net_{status} = [8, 12, 0, 0, 1000, 850, 0, 0]$$

Let's also consider that the different VNF types are:

$$vnf1 = [3, 150]$$
$$vnf2 = [2, 300]$$

and $vnf1$ is the first to arrive into the network. The initial state that the agent sees is:

$$s_0 = [3, 150, 8, 12, 0, 0, 1000, 850, 0, 0]$$

Suppose that the agent chooses to assign the first VNF to M1 and that $vnf2$ is the next request to arrive:

$$s_1 = [2, 300, 5, 12, 3, 0, 850, 850, 150, 0]$$

Two different states have been generated, how many possible states are in total following the example? Certainly it would take some computational power to compute all possible states. Also, what are the transition state probabilities with such a state structure?

How to overcome these incognites in a model-based MDP to be solved through Policy Iteration? The structure of the state needs to be modified, the information encapsulated in the new structure will be the same to all effects.

The new state structure is as follows:

$$s_t = [[[m1], [m2]], [d]]$$

where:

d: is a vector of size equal to the number of VNF types describing if a VNF arrives or departs from the network. Its possible values are:

$$d = \begin{cases} 1, & \text{if} \quad new\ VNF\ arrives \\ -1, & \text{if} \quad active\ VNF\ is\ terminated \\ 0, & \text{if} \quad no\ VNF\ arrival\ nor\ terminated \end{cases}$$

A vector like $[0, 1]$ indicates that a VNF of type 2 is arriving into the system. A vector like $[-1, 0]$ indicates that a VNF of type 1 is terminated. Notice that only one vector value will be either ($1, -1$), whereas the rest will be 0. Vector value positions indicate the type of VNF.

$m1$: is a vector indicating the number of alive/active VNF types deployed in MEC 1.

$m2$: is a vector indicating the number of alive/active VNF types deployed in MEC 2.

As an example, an $m1$ vector as $[2, 3]$ indicates that MEC 1 is serving 2 VNF of type 1 and 3 VNF of type 2. The position of each vector value indicates, not only the quantity but also the type of VNF deployed. The position of each vector value and the quantity of it, encapsulates the resources that are being used and implicitly the available ones.

Following the previous example, in a model-based MDP the initial state and the second one would be:

$$s_0 = [[[0, 0], [0, 0]], [1, 0]]$$
$$s_1 = [[[1, 0], [0, 0]], [0, 1]]$ [7]

A state like:

$$s_t = [[[1, 0], [0, 0]], [-1, 0]]$$

[7] Notice how the state structure holds the same information as in the Q-Learning state structure: [1, 0] = [5, 850], [0, 0] = [12, 850], [0, 1] = [2, 300]. Rearranging it: [2, 300, 5, 12, 3, 0, 850, 850, 150, 0].

Would indicate that a VNF of type 1 must be killed. Since the environment keeps track of the assigned VNF, it knows to which MEC was the VNF assigned, hence it can return the resources back to its belonging MEC.

The other element required is each state transition probability, which can be computed based on the Competing Exponentials theorem [13] from the Poisson process used to model the VNF inter arrival and departure times.

Note: Refer to annex F to see a detailed explanation on how to compute the state transition probability applying the Competing Exponential theorem.

2.4.3. Q-Learning

As mentioned in chapter 2.2, the purpose of the present work is to evaluate the Q-Learning algorithm against two more algorithms. Best Fit and MDP can be tested directly without the need of a previous training, the first one because it only takes into account the current state, hence, there is no need to populate a data structure with state-value functions. The second one, computes the optimal policy beforehand and once generated, no matter the VNF sequence, it will know how to optimally assign each VNF to its corresponding MEC.

The Q-Learning algorithm differs somehow from the rest of algorithms, it needs to populate the Q-Table previously, to test it later on with the same VNF sequence test files. For such, the Q-Learning will be trained with several VNF sequences files before putting it to evaluation. To prevent the possibility of the Q-Learning algorithm facing an unknown state (during the evaluation stage) and failing when trying to search it in its Q-Table, we'll modify the algorithm so it can also learn during the test stage. The agent will be able to distinguish between known states and unknown ones.

Finally, the Q-Learning hyper-parameters and the sub-algorithm parameters, e-greedy, need to be configured according to chapter 2.2. In the problem that it is intended to be solved there is no terminal state. In a real scenario, VNF arrives into the system in an infinite sequence, VNF are demanded and later on terminated. The problem to be tackled is a continuing tasks scenario, hence, the agent must be aware of it. For such, a suitable configuration of the Q-Learning hyper-parameters needs to be accounted for.

2.5. Summary

Chapter 2 has raised the problem to be addressed. The objective is to implement a Q-Learning algorithm to assign, in an optimal manner, the BS incoming VNF

demands to a 5G network EC. In order to evaluate such an algorithm performance, two more algorithms are implemented to compare them against each other. The concept of weighted EC, has also been introduced by means of computing the network's metric for the Best Fit algorithm. It has been described the needed modification the state structure must take in the MDP algorithm and the reasons for that. Finally it has been justified why Q-Learning needs a training stage prior to testing it with the common VNF sequence files.

CHAPTER 3. SIMULATION RESULTS

Chapter 3 shows the results across several simulation tests. Annex G describes the used tests strategy in order to cross-validate each algorithm and verify it is implementation correctness performance. A test for different values of the Q-Learning hyper-parameters demonstrates the influence of such parameters in order for the Q-Learning algorithm to reach convergence when learning. Section 4.3 proves how important it is to properly tune the parameters regarding the exploration and exploitation Q-Learning sub-algorithm. More detailed simulation graphs can be found in annex I regarding the Exploration/Exploitation tests. Finally, sections 3.3, 3.4, 3.5 and 3.6 show the results when comparing each algorithm for different values either in EC capacity or VNF demands.

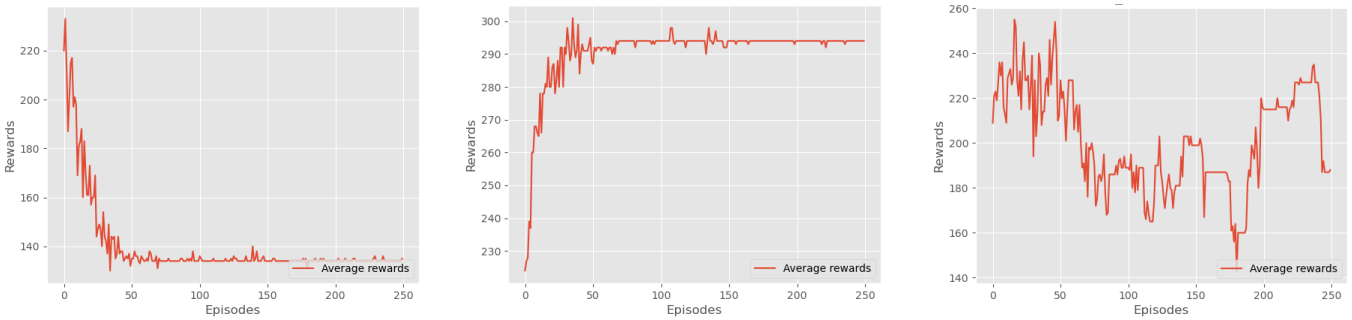
3.1. Influence of Q-Learning parameters

The effect of the Q-Learning hyper-parameters, learning rate (α) and discount factor γ, have been explained in chapter 2.5. In order to configure the suitable settings, several test runs are performed with different α and γ values.

Continuous tasks, as the tackled problem in the present work, force the agent to make a trade off by achieving a high reward in the long term but giving sufficient relevance to each current state value. It is interesting for the agent to learn enough, in order to allocate the VNF demands optimally, but remembering how it performed when faced with different sequences of VNF demands containing that particular VNF request. From figure 4.1 it can be seen that mid values for α and γ shows an optimal starting point for the best learning curve, converging from episode 60.

Settings	Values
No. training files	1
No. episodes	250
No. demands/episode	500

Table 4.1. Simulation settings



$\alpha = 0,\ \gamma = 0.001$ $\alpha = 0.5,\ \gamma = 0.5$ $\alpha = 0.9,\ \gamma = 0.9$

Fig. 4.1. Agent's learning performance curves during the training stage

Figure 4.1 represents the average reward that the agent achieves for 250 episodes for different α and γ values with the purpose of showing the hyper-parameters effect in the agent's learning performance. Nevertheless, an extensive number of simulations were carried out and can be seen in annex H.

3.2. Exploration and Exploitation algorithm parameters

The implemented sub-algorithm in Q-Learning (e-greedy) that determines how much of an explorer or exploiter the agent is, has certain parameters that precisely, defines for how long (for how many episodes) is going to adopt such profiles. Those parameters are:

- The initialized exploration rate: $\epsilon = 1$
- The exploration probability at start of episode: $\epsilon_{max} = 1.0$
- The minimum exploration probability to reach: $\epsilon_{min} = 0.001$
- The rate at which the ϵ value decays after each episode: ϵ_{decay}
- The episode number of all episode range: $episode_i$

The ϵ value decays after each episode by means of:

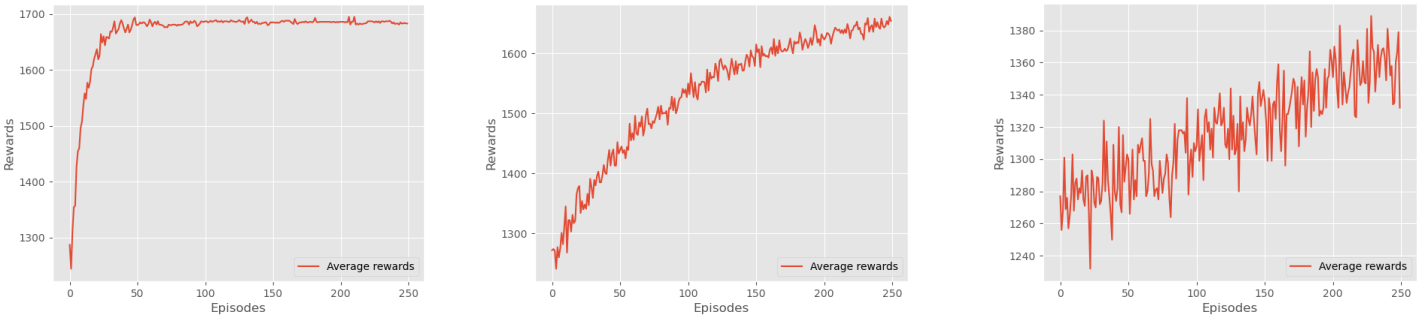
$$\epsilon = \epsilon_{min} + (\epsilon_{max} - \epsilon_{min}) \cdot e^{(-\epsilon_{decay} \ \cdot \ episode_i)} \qquad (4.3)$$

The decay rate (ϵ_{decay}) has an enormous influence for achieving convergence on the agent's learning performance. Depending on the number of episodes, a very small rate (ϵ_{decay}) might provoke the agent to never get to exploit what has already learned. By leaving a minimum epsilon value ($\epsilon_{min} = 0.001$), we open the door for the agent to explore from time to time, even towards the end of the training stage,

that's the reason why certain peaks can be seen along the performance curve once it has converged (Fig. 4.2 left graph).

Settings	Values
No. Training files	1
No. episodes	250
No. demands/episode	2000

Table 4.2. Simulation settings



$\epsilon_{decay} = 0.1$ $\epsilon_{decay} = 0.01$ $\epsilon_{decay} = 0.001$

Fig. 4.2. Exploration/Exploitation influence on the agent's learning performance

Numerous tests were executed with different ϵ_{decay} (annex I) to study the performance of the agent in the modeled environment in order to discover the balance between the number of training episodes and the number of VNF demands. It should be noted, that the agent is performing in a continuous task environment and that there is no terminal state, the main goal of the agent is to optimally allocate the VNF demand to an EC center (at every state) and at the same time be able to remember certain patterns of consecutive incoming VNF demands. For such, it needs to remember what has already been learned and not be motivated to learn everytime faces a known VNF demand in a different VNF sequence. Hence, it needs to explore for a certain time and gather the knowledge to exploit it.

3.3. Rejection ratio with respect to VNF arrival rate

In this simulation, the performance of the algorithms are evaluated according to different values of VNF arrival rates, (λ). For such, we alter the initial values for the arrival VNF rates multiplying them by a certain factor. The objective is to analyze how the inter-arrival times between demands affect the EC resources. We force the agent to deal with consecutive incoming requests in short periods of time and with large inter-arrival times.

In simulation 1, the short arrival and departure rate for VNF of type 2, provokes that such demands arrive and terminate before a new demand arrives, returning the consumed resources back to its EC center to which it was assigned. This short inter-arrival time between VNF forces that VNF of type 1 endures over several timesteps, shortening the number of possibilities for the agent to optimally assign requests to MECs. The agent, for most of the time, is only learning how to assign VNF of type 1, the reason why performance between MDP and Q-Learning is equal, as seen in figure 4.3 (parametrized factor 0.2). As the inter-arrival times increases between demands, the MDP computes a better policy than Q-Learning and it can be seen that the gap between both algorithms are very close.

Settings	Values
Initial network status	[4, 12, 0, 0, 1000, 400, 0, 0]
No. Training files	10
No. VNF demands/training	500
No. episodes/training	250
No. Test files	20
No. VNF demands/test	500
λ	(3, 2)

Table 4.3. Simulation settings

Value/Simulation	Simulation 1	Simulation 2	Simulation 3
Parameterized factor	λx0.2	λx1	λx2
Arrival rate	(0.6, 0.4)	(3, 2)	(6, 4)
Departure rate	(1, 0.5)	(1, 0.5)	(1, 0.5)
VNF types	[(1, 300), (3,50)]	[(1, 300), (3,50)]	[(1, 300), (3,50)]

Table 4.4. VNF parametrized arrival rate

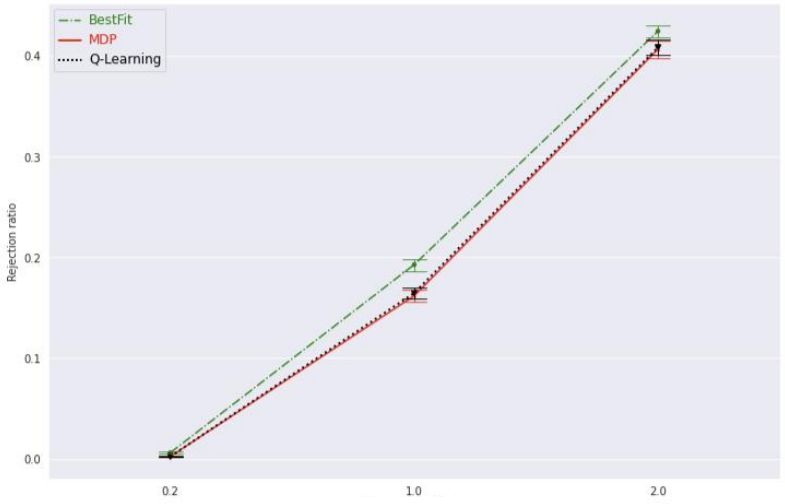


Fig. 4.3. Performance according to different VNF arrival rates

X-axis on Figure 4.3 represents the different values by which the VNF arrival rate has been multiplied with. On the Y-axis we find the rejection VNF ratio for each algorithm test performance. Annex J offers boxplots to see the distribution of rejections for every algorithm and test.

3.4. Rejection ratio with respect to EC capacity

In this simulation, the performance of the algorithms are evaluated with different values for MEC CPU cores and BW capacity by means of multiplying each mec's capacity by a factor: 0.8, 1 and 1.2. The VNFs inter-arrival and departure rates follow the same settings as in simulation 2 from section 3.3.

This test allows us to analyze how the agent in the Q-Learning performs as we increase the available resources on each EC center and how it differs from the MDP. It also allows us to see how the agent allocates each VNF demand to an EC center, does it assign VNFs of the same type to a certain EC or does it try to balance the available resources as it does the Best Fit across all EC centers.

Settings	Values
No. Training files	10
No. VNF demands/training	500
No. episodes/training	250
No. Test files	20
No. VNF demands/test	2000
M1, M2: [CPU, BW]	[[5, 1000], [10, 400]]

Table 4.5. Simulation settings

Value/Simulation	Simulation 1	Simulation 2	Simulation 3
Parameterized factor	0.8	1	1.2
M1, M2 CPU cores	[4, 8]	[5, 10]	[6, 12]
M1, M2 BW	[800, 320]	[1000, 400]	[1200, 480]

Table 4.6. Mecs parametrized capacity

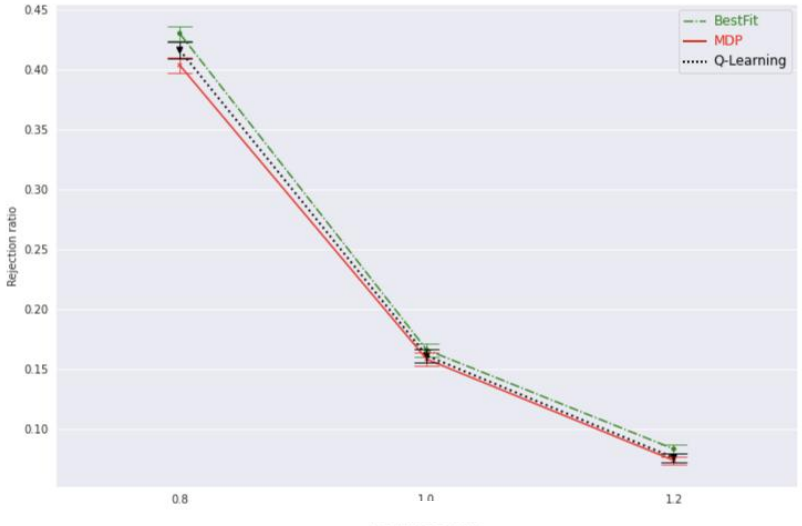


Fig. 4.4. Performance according to different mecs capacity values

Figure 4.4 shows the performance difference between MDP and Q-Learning when EC resources are somewhat scarce, the Q-Learning's policy is better than the Best Fit but is worse than the MDP. The randomness of the Q-Learning during the exploration stage, determines somehow the starting point to which to build the path for the optimal policy. As the exploitation stage starts to be applied, the probabilities of changing what is already known are less proun. Besides, the available CPU resources in EC 1 are consumed very quickly making the rest of the outcome somehow deterministic. As the available resources are increased on each EC, Q-Learning appears to have more margin to learn and to approach the ideal policy as the MDP.

Note: Refer to annex J for boxplots version graphs.

3.5. Rejection ratio with respect to EC resource heterogeneity

In this simulation, we fix the number of available resources in EC 1 and we increase the available number of CPUs in EC 2 while we decrease the available BW link in EC 2 also. The aim is to see how each EC parameter affects the agent's decision. The VNFs inter-arrival and departure rates follow the same settings as in simulation 2 from section 3.3. The performance of the algorithms are evaluated in different CPU cores and BW by means of:

$$mec2_{cpu} = \beta \cdot mec1_{cpu} \ ; \ mec2_{BW} = \frac{1}{\beta} \cdot mec1_{BW} \qquad \textbf{(4.1)}$$

The VNFs follow the same settings as in simulation 2 from section 3.3.

Settings	Values
No. Training files	10
No. VNF demands/training	500
No. episodes/training	250
No. Test files	20
No. VNF demands/test	2000
M1: [CPU, BW]	[4, 1000]

Table 4.7. Simulation settings

Value/Simulation	Simulation 1	Simulation 2	Simulation 3
Parameterized factor	$\beta = 1$	$\beta = 2.5$	$\beta = 3$
M2 CPU cores	[4]	[10]	[12]
M2 BW	[1000]	[400]	[333]

Table 4.8. Mec 2 capacity parametrized factor

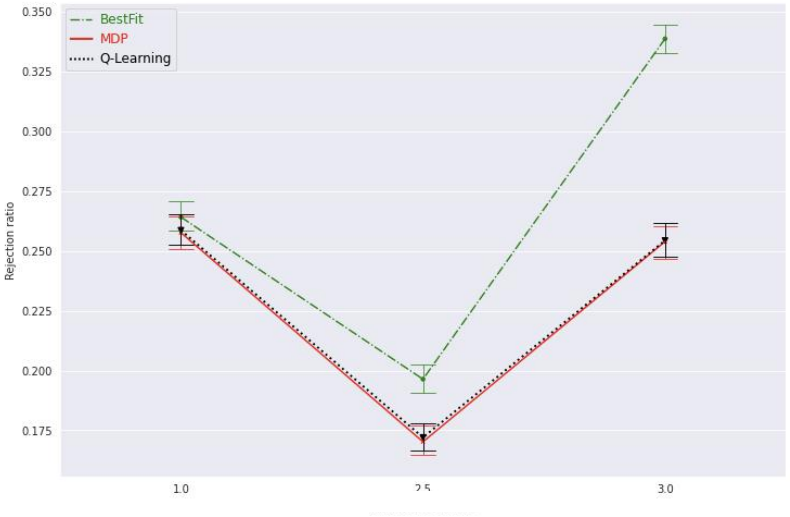


Fig. 4.5. Performance according to resource heterogeneity

In simulation 3 although having increased the number of available CPU cores for EC 2, the BW has been reduced. This leaves EC 2 to only be able to serve a VNF of type 1, 6 of type 2 or a mix of both until reaching the maximum MEC capability, whereas in simulation 2 it can accept not only one VNF of type 1 but also two VNF of type 2, or 8 of type 2 or a mix of both before running out of resources. Nevertheless, it can be appreciated how the Q-Learning algorithm performs similarly to MDP when EC resources are tight, as seen in figure 4.5.

Note: Refer to annex J for boxplots version graphs.

3.6. Rejection ratio with respect to VNF demand heterogeneity

In this simulation we proceed the same way as in the previous one but affecting the VNF demands values. For the case, we leave VNF of type 1 static and we modify the values for VNF of type 2. We've increased the available EC resources to leave enough margin for the Q-Learning to learn how to assign each VNF demand. For VNF of type 2, as we increase the number of CPU requests, we decrease the required link BW. The VNFs inter-arrival and departure rates follow the same settings as in simulation 2 from section 3.3.

Algorithms are evaluated in different CPU cores and BW VNF demands by means of:

$$VNF2_{cpu} = \beta \cdot VNF1_{cpu} \; ; \; VNF2_{BW} = \frac{1}{\beta} \cdot VNF1_{BW} \qquad (4.2)$$

Settings	Values
Initial network status	[8, 12, 0, 0, 1000, 1000, 0, 0]
No. Training files	10
No. VNF demands/training	500
No. episodes/training	250
No. Test files	20
No. VNF demands	2000
VNF1: [CPU, BW]	[2, 300]

Table 4.9. Simulation settings

Value/Simulation	Simulation 1	Simulation 2	Simulation 3
Parameterized factor	$\beta = 1.5$	$\beta = 2$	$\beta = 4$
VNF2 CPU	[3]	[4]	[8]
VNF2 BW	[200]	[150]	[75]

Table 4.10. VNF 2 demands parametrized factor

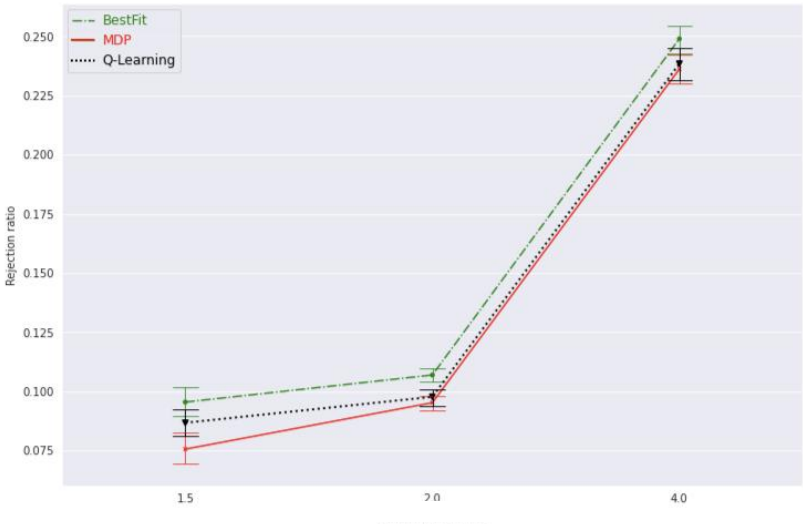


Fig. 4.6. Performance according to VNF demand heterogeneity

It can be seen from figure 4.6 that Q-Learning is quite far in performance from the MDP when VNF demands are small and EC resources are enough. Again, it seems that Q-Learning performs better when resources from EC are constrained for higher demanded VNF requests performing close to the optimal policy of MDP.

Note: Refer to annex J for boxplots version graphs.

CONCLUSIONS

There is a striking performance similarity between the Q-Learning and the MDP, especially when both algorithms are faced with limited EC available resources. Q-Learning does not obtain the optimal policy that MDP does, even though it is very close to reaching it. Nevertheless, Q-Learning is a Model-free based algorithm whereas MDP needs to know all environment dynamics including all possible valid states beforehand, an arduous task to meet in a real world scenario. MDP is a great optimization RL model for controlled and small environments, where all conditioning factors are well known, which is not the 5G case.

It also has been shown how Q-Learning performs better than Best Fit in all cases, hence confirming that implementations with intelligence could perform better than the classical load balancer algorithms like the implemented in the present thesis. Q-Learning has proved to work well considering cloud and transport network parameters, which was the main aim during the problem statement.

Modeling the environment for all three algorithms has been a challenging task since not only the agent's side was important but also the environment in order to simulate as closely as possible a network scenario by not influencing the agent's decisions.

Future research lines to develop from this thesis, would be to integrate Neural Networks (NN) in the Q-Learning approach, like Deep Q-Networks (DQN). Although Q-Learning has proved to perform quite well, the fact of having to store all Q-values in a data structure makes it infeasible for very large scenarios. The increase is exponentially, as new possible actions are added, becoming an impractical model due to the computational cost it requires. By integrating neural networks, there is no need to save anymore the states and its Q-values, since the NN can approximate a Bellmans' equation solution to obtain the Q values.

The present master thesis has been somehow constrained by the time and processing power that simulations require, which denotes how important powerful computers are required when running RL models, specially if intended to run DQN or any other NN to accomplish optimal results.

ACRONYMS

3GPPP	3rd Generation Partnership Project
5G	Fifth Generation
5G-SO	5Growth Service Orchestrator
5GPPP	Fifth Generation Partnership Project
5Gr-AIMLP	Artificial Intelligence/Machine Learning platform
5Gr-RL	5Growth Resource Layer
5Gr-VoMS	Vertical-oriented Monitoring System
5Gr-VS	5Growth Vertical Slicer
AI	Artificial Intelligence
AR	Augmented Reality
BW	Bandwidth
CAGR	Compound Annual Growth Rate
CAPEX	Capital Expenditure
CDF	Cumulative Distribution Function
COTS	Commercial-Off-The-Shelf
CTTC	Centre Tecnològic de Telecomunicacions de Catalunya
DP	Dynamic Programming
EB	Exabytes
EC	Edge Computing
EIR	Equipment Identity Register
eMBB	Enhanced Mobile Broadband
EMS	Element Management System
EOL	End-of-Life

gNB	5G Radio Base Station
IaaS	Infrastructure as a Service
IMT-2020	International Mobile Telecommunication 2020
IoT	Internet of Things
ITU	International Telecommunication Union
KPI	Key Performance Indicators
LCM	Life Cycle Management
LTE	Long Term Evolution
MANO	Management and Orchestration
MDP	Markov Decision Processes
MEC	Multi-Access Edge Computing
MIMO	Multiple-Input and Multiple-Output
ML	Machine Learning
mMTC	Massive machine type communications
MNO	Mobile Network Operators
NFV	Network Functions Virtualization
NFV-NS	NFV-Network Services
NFVO	Network Function Virtualization Orchestrator
NSD	Network Service Descriptor
NWDAF	Network Data Analytics Function
OPEX	Operating Expenditure
OSS/OBS	Operations Support System/Business Support System
PDF	Probability Density Function
QoS	Quality of Service
RAN	Radio Access Network
ROI	Return Of Investment

SDN	Software Defined Networking
SLA	Service Level Agreement
SO	Service Orchestrator
TD	Temporal Difference
UE	User Equipment
URLLC	Ultra-reliable and low-latency communications
VNF	Virtualized Network Functions
VR	Virtual Reality
vRAN	Virtual Radio Access Network
VSB	Vertical Service Blueprint

REFERENCES

[1] Research and markets. 2021. “Autonomous Vehicle Market by Autonomy Level, Powertrain Type, Components, and Supporting Technologies including 5G, AI, and Edge Computing 2021 - 2026.” Research and Markets. Report. Pdf. https://www.researchandmarkets.com/reports/5241675/autonomous-vehicle-market-by-autonomy-level?utm_source=BW&utm_medium=PressRelease&utm_code=r4vssq&utm_campaign=1500931+-+Global+Autonomous+Vehicle+Market+2021-2026%3a+AI%2c+5G%2c+and+Edge+Computing+Solutio.

[2] Grand View Research. 2021. “5G Services Market Size, Share & Trends Analysis Report By Communication Type (FWA, eMBB, uRLLC, mMTC), By Vertical (Manufacturing, IT & Telecom, BFSI), By Region, And Segment Forecasts, 2021 - 2028.” ID: GVR-3-68038-435-2. Report. Pdf. https://www.grandviewresearch.com/industry-analysis/5g-services-market.

[3] 5G-PPP Technology board. 2021. “AI and ML – Enablers for Beyond 5G Networks.” DOI: 10.5281/zenodo.4299895. V1.0. AI and ML – Enablers for Beyond 5G Networks. Pdf. https://5g-ppp.eu/wp-content/uploads/2021/05/AI-MLforNetworks-v1-0.pdf.

[4] Dahlman, Erik, Stefan Parkvall, and Johan Sköld. 2018. *5G NR The next generation wireless access technology*. 1st Edition ed. N.p.: Academic Press. ISBN: 9780128143230. Book. https://www.elsevier.com/books/5g-nr-the-next-generation-wireless-access-technology/dahlman/978-0-12-814323-0

[5] Mangues-Bafalluy, Josep, and . et al. 2020. “5Growth: AI-driven 5G for Automation in Vertical Industries.” In *5Growth: AI-driven 5G for Automation in Vertical Industries*. ISBN:978-1-7281-4355-2. IEEE Xplore. Paper. Pdf. https://ieeexplore.ieee.org/document/9200919/authors#authors.

[6] ETSI Technical specification. 2020. “5G; System architecture for the 5G System (5GS) (3GPP TS 23.501 version 16.6.0 Release 16).” V16.6.0. ETSI. Pdf. https://www.etsi.org/deliver/etsi_ts/123500_123599/123501/16.06.00_60/ts_123501v160600p.pdf.

[7] Saasha, Nair. 2020. “To explore or not to explore, that is the question.” Medium. Web. Last accessed (June, 2021) https://medium.com/analytics-vidhya/rl-series-3-to-explore-or-not-to-explore-1ff88e4bf5af.

[8] Sutton, Richard S., and Andrew G. Barto. 2018. *Reinforcement Learning An introduction*. Second ed. N.p.: The MIT Press. ISBN: 9780262039246. Book. Pdf. https://mitpress.mit.edu/books/reinforcement-learning-second-edition

[9] Cornish Hellaby Watkins, Christopher J. 1989. “Learning from Delayed Rewards,” Ph.D. Thesis. ResearchGate. Ph. D Thesis. Pdf. https://www.researchgate.net/publication/33784417_Learning_From_Delayed_Rewards.

[10] Robbins, Herbert, and Sutton Monro. 1951. “A Stochastic Approximation Method,” Annual of Mathemathical Statistics 22 400- 407. DOI: 10.1214/aoms/1177729586. Paper. Pdf. https://projecteuclid.org/journals/annals-of-mathematical-statistics/volume-22/issue-3/A-Stochastic-Approximation-Method/10.1214/aoms/1177729586.full.

[11] Daswani, Mayank, Marcus Hutter, and Peter Sunehag. 2013. “Q-learning for history-based reinforcement learning.” PMLR 29:213-228,. Research Gate. Conference paper. Pdf. https://www.researchgate.net/publication/287339270_Q-learning_for_history-based_reinforcement_learning.

[12] Sancho Caparrini, Fernando. 2019. “Aprendizaje por refuerzo: algoritmo Q Learning.” Fernando Sancho Caparrini. Web. Last accessed (june, 2021) http://www.cs.us.es/~fsancho/?e=109.

[13] Li, Jimmy. 2014. “MIT 6.041SC Probabilistic Systems Analysis and Applied Probability, Fall 2013.” In *MIT 6.041SC Probabilistic Systems Analysis and Applied Probability, Fall 2013*. Youtube. Video. Last accessed (june, 2021) https://www.youtube.com/watch?v=eUPpqZFSqdw.

[14] Open AI. 2015. “Gym.” Open AI Gym. https://gym.openai.com/.Web. Last accessed (june, 2021)

[15] Penttinen, Jyrki T. 2019. *5G Simplified ABCs of Advanced Mobile Communications*. 1st ed. Atlanta, GA: Independently Published, 2019. ISBN: 1086032608, 9781086032604. Book. Pdf. https://books.google.es/books/about/5G_Simplified.html?id=O6Q1yQEACAAJ&redir_esc=y

[16] X. Li, A. Garcia-Saavedra, X. Costa Perez, C. Jesus Bernardos, C. Guimaraes, K. Antevski, J. Mangues, J. Baranda, E. Zeydan, D. Corujo, P. Iovanna, G. Landi, J. Alonso, P. Paixao, H. Martins, M. Lorenzo, J. Ordoñez-Lucena and D. López , " 5Growth: An End-to-End Service Platform for Automated Deployment and Management of Vertical Services over 5G Networks " in IEEE Communications Magazine, vol. 1, pp. 1-1, May 2021. DOI: 10.1109/MCOM.001.2000730 https://ieeexplore.ieee.org/document/9422344

[17] Andres Garcia Saavedra (NEC), Xi Li (NEC), Josep Xavier Salvat (NEC), Ioannis Stavrakakis (NKUA), Nancy Alonistioti (NKUA), Sokratis Barmpounakis (NKUA),... . 2019. "D2.1: Initial Design of 5G End-to-End Service Platform." 1.0. 5Growth Deliverables. Pdf. https://5growth.eu/wp-content/uploads/2019/11/D2.1-Initial_Design_of_5G_End-to-End_Service_Platform.pdf.

[18] ETSI Group specification. 2013. "Network Functions Virtualisation (NFV); Architectural Framework." V1.1.1. pdf. https://www.etsi.org/deliver/etsi_gs/NFV/001_099/002/01.01.01_60/gs_NFV002v010101p.pdf.

[19] Sigaud, Olivier. 2020. “Dynamic Programming,” Reinforcement Learning, Class about Dynamic Programming. Youtube. Video. Last accessed (June, 2021) https://www.youtube.com/watch?v=wOLBxDA6SRY&list=PLe5mY-Da-ksWV330WbfazLUyOuR59sers&index=3.

ANNEX

Annex A: Value function iteration demonstration

With the purpose of demonstrating how to obtain the value function from the states of an environment, let's assume the following scenario [19]. An agent must reach a certain state, rewarded with 1, colored green. Whenever falls in the red state, receives a punishment of -1. In order to motivate the agent to reach the green state as quickly as possible, we will apply a discounting reward of -0.04 for every movement it does until reaching the final state. This way we assure that the agent will try to learn the shortest path.

s_3	s_4	s_5	1
s_2		s_6	-1
s_1	s_7	s_8	s_9

The action space is: up, down, left and right. Unfortunately, the agent movements are not totally reliable, that is, 20% of the times will end in a state other than the intended to. Whereas for 80% of the time it will achieve its desired destination. Opposite undesired movement to the intended one are not considered. We've just defined our state transition probability.

As mentioned in chapter 2.3.4.4 the Bellman equation computes the value function in a backup propagation method. Given the initial described circumstances, will start from the final optimal policy and its value functions.

$\rightarrow$	$\rightarrow$	$\rightarrow$	1
$\uparrow$		$\uparrow$	-1
$\uparrow$	$\leftarrow$	$\leftarrow$	$\leftarrow$

0.812	0.868	0.918	1
0.762		0.660	-1
0.705	0.655	0.611	0.388

From all possible policies, the one above is the optimal one. Suppose our agent starts at state 1, notice how the optimal path goes through states 2, 3, 4 and 5 avoiding states 7, 8 and 6 due to the high risk of landing in the red state. States 7, 8 and 9 add an extra cost to the agent if it did start on those states (remember there's -0.04 for every movement) but even though the expected return is greater than the risk of ending in state 6 or in the red state.

No matter in which state our agent starts from one the optimal policy is computed, with the optimal state value functions it will know how to reach the final state in the shortest possible path. Each of the value functions will tell the agent the expected return it will perceive until getting to the final state.

For convenience we will assume a discounting factor of $\gamma = 1$.

How do we verify the optimal state value functions? Let's verify state 5 which is the last state that allows it to end in the terminal state.

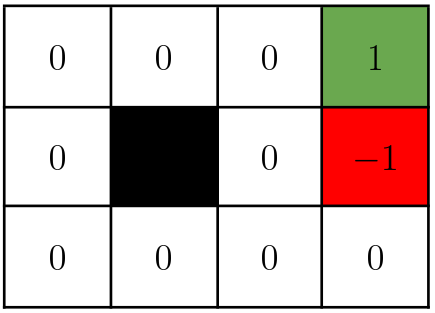
$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a) \cdot [r + \gamma v_\pi(s')], \ for\ all\ s \in S$$

$$(0.8 \cdot 1 + 0.1 \cdot 0.660 + 0.1 \cdot 0.918) - 0.04 = 0.918$$

The optimal movement is to the right and that's what the first term is telling, there's 80% chances to move to the right and get the final reward of 1. But there's also a 10% chance of ending down and the state value function of state 6 is 0.660. There's also a 10% probability of moving up which will provoke the agent to remain in state 5 which has an expected return of 0.918. Finally, we must discount -0.04 no matter where the agent ends up.

In order to verify the value function of state 5 we had to know previously what was the value from itself and from state 6. Also we would need to know the value itself of state 6 and from state 8, and same thing with the rest of states.

The very first time our agent would see the environment, all states would be initialized to:



It is a common practice to initialize all states to cero, although random values for each state would also work.

Those would be the value functions for each state, no matter from which the agent would start.

After applying the Bellman equation on the first iteration on each state except state 5, the value functions would be:

$$(0.8 \cdot 0 + 0.1 \cdot 0 + 0.1 \cdot 0) - 0.04 = -0.04$$

State 5 is different because there is an optimal movement that will allow the agent to get to the final state and earn a reward of 1.

$$(0.8 \cdot 1 + 0.1 \cdot 0 + 0.1 \cdot 0) - 0.04 = 0.76$$

-0.04	-0.04	0.76	1
-0.04		-0.04	-1
-0.04	-0.04	-0.04	-0.04

Notice how if our agent would start from any state other than 5, it would not know which movements to try in order to reach its goal, the expected return is negative in all of them. This is the reason why at the beginning it is interesting to have an explorer agent.

On the second iteration when computing the state value functions, states 4 and 6 will have new values because state 5 already has a higher expected return. In the third iteration, states 3, 4, 5, 6 and 8 will update their state values for the same reason as before, states 4 and 6 have updated values. This is dynamic programming, iterating over each state computing the state value function until it converges to a fixed point reaching the optimal state value function on all possible states as we saw at the very beginning.

Annex B: Policy Iteration demonstration

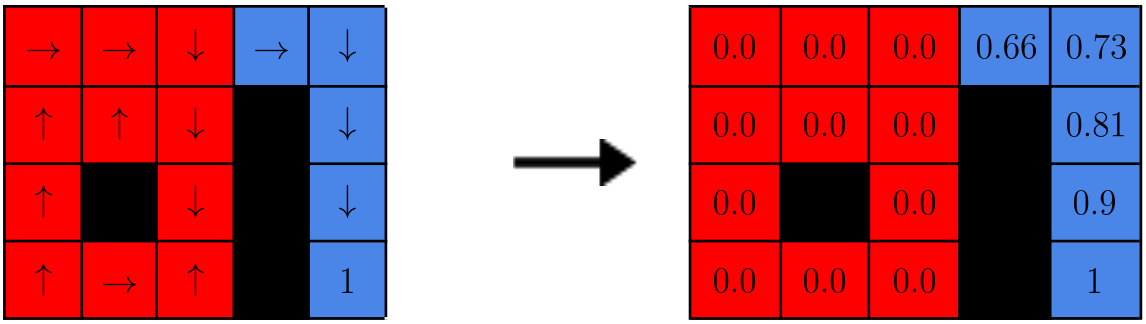
B2.1. Policy Evaluation

Let's consider an MDP [19] representing an *Episodic Task* environment within a maze, where the agent's goal is to reach from an initial state, to a terminal state, (s_{16}) in the minimum time steps (t).

s_1	s_5	s_{11}	s_{12}	s_{13}
s_2	s_6	s_{10}		s_{14}
s_3		s_9		s_{15}
s_4	s_7	s_8		s_{16}

For such a mission, the agent counts with four possible actions: left, right, up or down.

In policy evaluation we have a given policy and from it, it is computed the state-value function (refer to annex A), let's assume the following agent's policy:



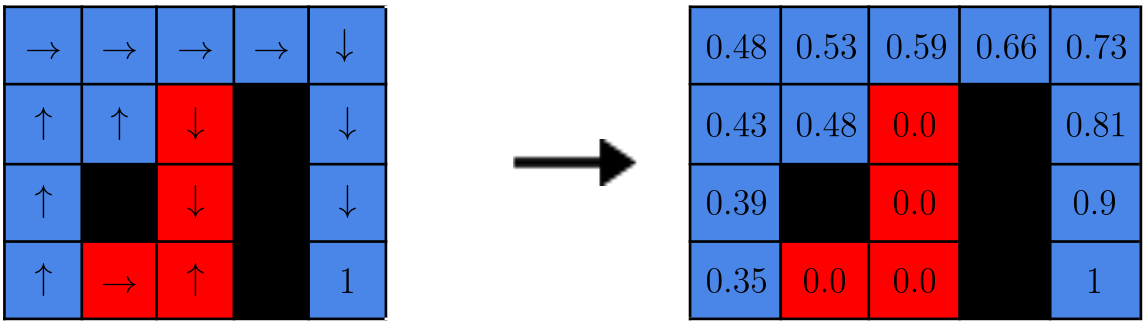
$\pi_0(s) \leftarrow given\ policy$ $\qquad$ $V_0(s) \leftarrow evaluate(\pi_0(s))$

All states colored in red have a state-value function equal to cero. No matter in which state (among the colored in red) the agent starts from, it will always end up in a loop between states (s_8) and (s_9) for the given policy. Reason why none of those states are good in order to reach the terminal state (s_{16}).

It is only from (s_{12}) where the state-value function of all successive states rises in value as the final state (s_{16}) is approached. Hence, implicitly defining the optimal policy for those last states colored in blue.

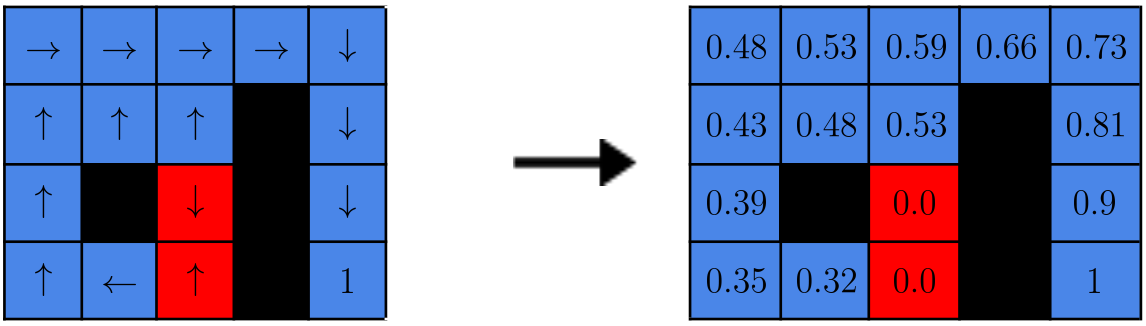
B2.2. Policy improvement

After computing the state-value function for the given policy, it is then improved. Notice how changing the action in (s_{11}) triggers the update of several states when computing again the state-value function for the new policy.



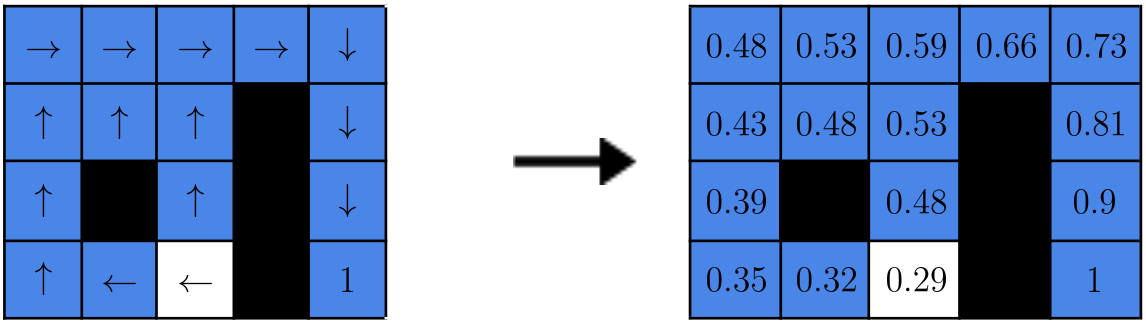
$\pi_1(s) \leftarrow improve(\pi_0(s), V_0(s))$ $V_1(s) \leftarrow evaluate(\pi_1(s))$

After the first iteration, it is still possible to improve the policy by changing actions in (s_7) and (s_{10}).



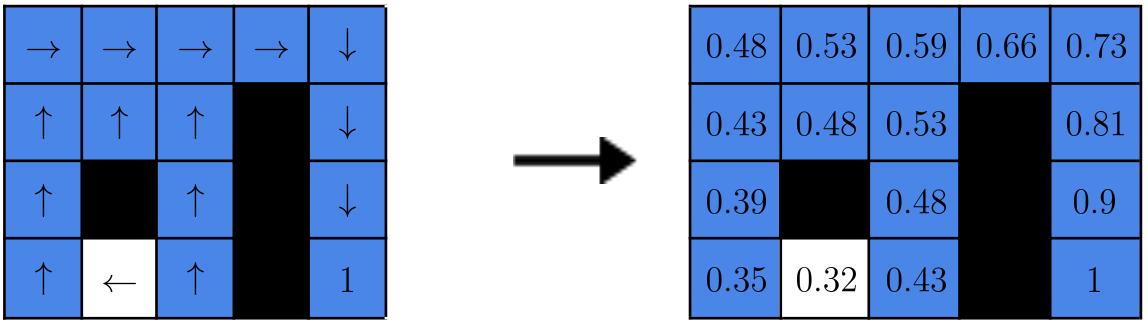
$\pi_2(s) \leftarrow improve(\pi_1(s), V_1(s))$ $V_2(s) \leftarrow evaluate(\pi_2(s))$

We iterate again over the last policy to improve it by changing actions in (s_8) and (s_9).



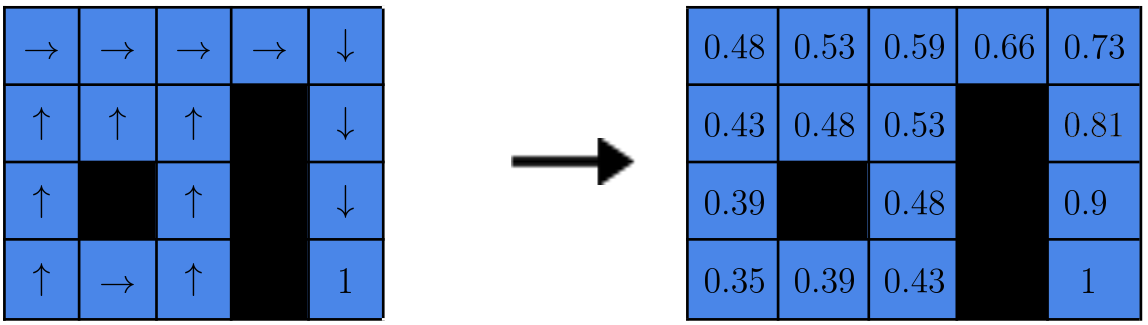
$\pi_3(s) \leftarrow improve(\pi_2(s), V_2(s))$ $V_3(s) \leftarrow evaluate(\pi_3(s))$

Although it could seem that the optimal policy has been reached, the last improvement in (s_8) is not the action gathering the highest state-value function, instead of taking action to the left, it would have been better to take an up action since (s_9) is higher in value than (s_7). Hence, we still can improve it by changing the action again.



$\pi_4(s) \leftarrow improve(\pi_3(s), V_3(s))$ $V_4(s) \leftarrow evaluate(\pi_4(s))$

By repeating the above steps we end up with the optimal policy and also with the optimal state-value functions.



$\pi_5(s) \leftarrow improve(\pi_4(s), V_4(s))$ $V_5(s) \leftarrow evaluate(\pi_5(s))$

Annex C: Contractive operator

Assuming T^* as a Bellman operator such that:

$$V_n T^* \rightarrow V_{n+1} \tag{B.3.1}$$

if T^* is contractive, it can be assure that by means of iterative application of T^*, V_n will converge to a fixed value. Let's call Bellman operator the application:

$$V^{\pi}_{n+1}(s) \leftarrow \sum_{s',r} p(s', r|s, a)[r(s, \pi(s)) + \gamma V(s')] \tag{B.3.2}$$

- If $\gamma < 1$, T^* is contractive
- It will converge to an optimal value and policy
- Policy iteration:

    a. Policy evaluation:

$$V^{\pi}_{n+1} \leftarrow T^* V^{\pi}_n \tag{B.3.3}$$

    b. Policy improvement:

$$\pi'(s) \leftarrow max_{a \in A} \sum_{s',r} p(s', r|s, a)[r(s, a) + \gamma V(s')] \ \forall s \in S \tag{B.3.4}$$

Annex D: Poisson Process

Let $x_1, x_2, \cdots, x_k$ be random events in such that:

x_1: is the interarrival time between the process start and the first event
x_2: is the interarrival time between the first and the second incoming event.
x_k: is the interarrival time between the $k-1$ and k event.

The exponential distribution of the arrival times can be defined by:

$$X_k = \exp(\lambda)$$

where λ, is the average occurrences event rate. If the Probability Density Function (PDF) for X_k is:

$$P_X(t) = \lambda e^{-\lambda t}$$

and the Cumulative Distribution Function (CDF) is:

$$F_X(t) = P(X \leq t) = \int_0^t \lambda e^{-\lambda x} dx = 1 - e^{-\lambda t}$$

Then the interarrival times in a Poisson process can be modeled applying the inverse CDF technique feeding different random values following a uniform distribution $t \in (0, 1)$:

$$F_X^{-1}(t) = -\frac{ln(1-t)}{\lambda}$$

In Python it can be implemented by using the Numpy library and its function *random.exponential()*:

```
import numpy as np

vnf_interArrival = np.random.exponential(1.0 / arrival_rate)
vnf_killTime = np.random.exponential(departure_rate)
```

Annex E: Best Fit flowchart

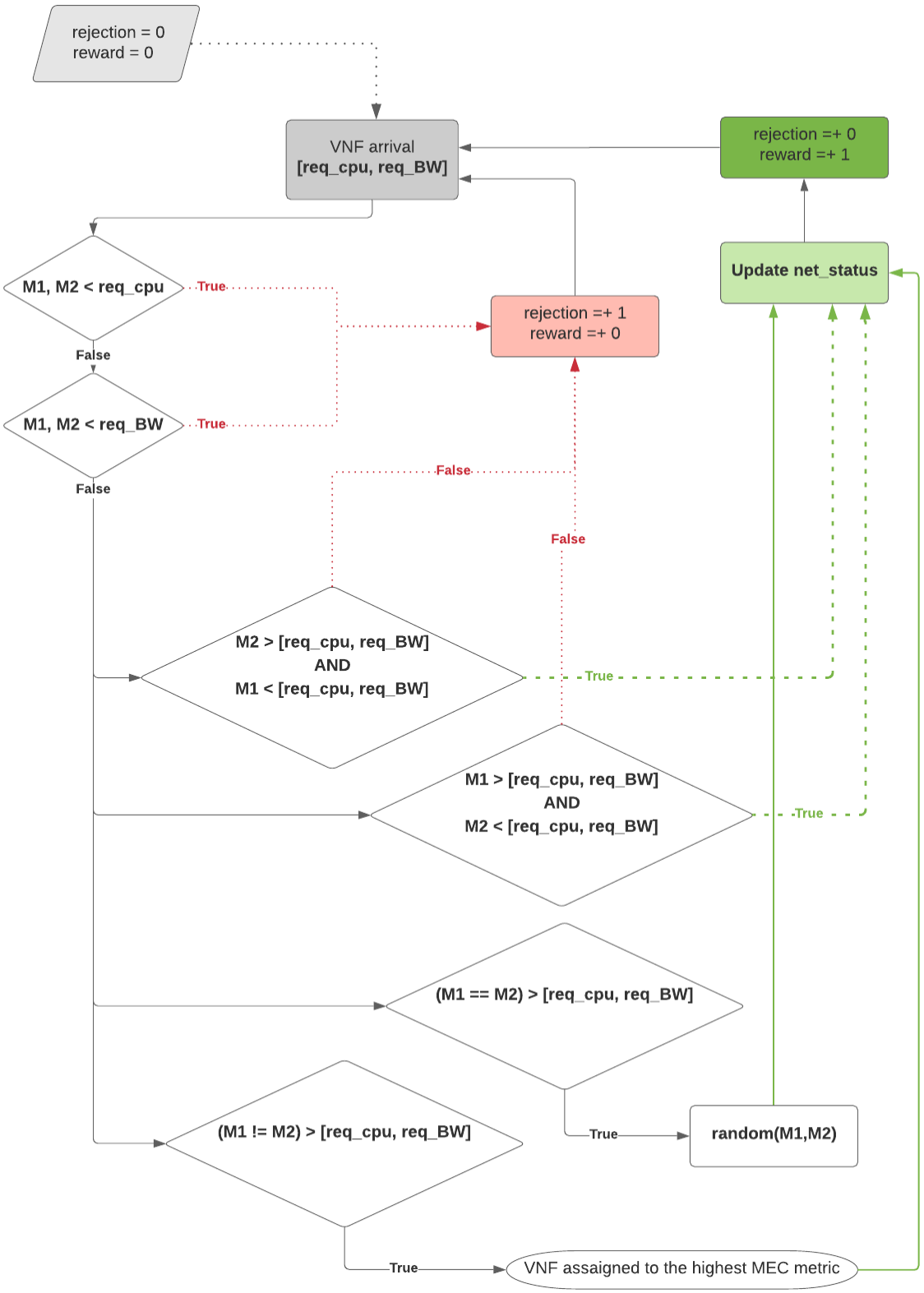


Fig. F.1. Best Fit algorithm flowchart

Annex F: Computing state transition probabilities

State transition probabilities are key, in order to compute the optimal agent's policy. A state transition probability may describe the probability of an agent success outcoming after choosing an action, i.e., a robot may have a failure probability for each of its actions. In the present thesis, the state transition probabilities are computed from the Poisson process applied to the arrival VNF demands into the network.

Recalling the state structure defined in chapter 3.5.2, the state transition probability can be computed by applying the Competing Exponential theorem [13].

Let us consider the following scenario:

$num_{mecs} : 2$
$max_{mec\ capacity} : 2$
$arrival_{rate} : (\lambda_1 = 2), (\lambda_2 = 4)$
$departure_{rate} : (\mu_1 = 0.25), (\mu_2 = 1)$
$vnf1_{demands} : (2, 1000)$
$vnf2_{demands} : (4, 2000)$

E.g. 1:

$s_0 \leftarrow [[[1, 0], [0, 1]], [1, 0]]$

State (s_t) is describing one active VNF of type 1 in mec 1 and one active VNF of type 2 in mec 2 and a new VNF, of type 1, arriving into the system.

According to the previous state, all possible state transitions for each possible agent's action are:

If the agent chooses mec 1 to assign the incoming VNF, the following state will either receive a new VNF arrival or a VNF departure. In case of a new arrival, the new state will be:

1. $s_1 \leftarrow [[[2, 0], [0, 1]], [1, 0]]$ for a VNF type 1 arrival

2. $s_2 \leftarrow [[[2, 0], [0, 1]], [0, 1]]$ for a VNF type 2 arrival

If instead of a new arrival, there's a departure, then the new state will be:

3. $s_3 \leftarrow [[[2, 0], [0, 1]], [-1, 0]]$ for a VNF type 1 departure
4. $s_4 \leftarrow [[[2, 0], [0, 1]], [0, -1]]$ for a VNF type 2 departure

Probabilities for each of the possible new states is computed as:

$$p(s_1) \leftarrow \frac{\lambda_1}{\lambda_1 + \lambda_2 + 2\mu_1} = 0.26666666666666666$$
$$p(s_2) \leftarrow \frac{\lambda_2}{\lambda_1 + \lambda_2 + 2\mu_2} = 0.5333333333333333$$
$$p(s_3) \leftarrow \frac{2\mu_1}{\lambda_1 + \lambda_2 + 2\mu_1} = 0.06666666666666667$$
$$p(s_4) \leftarrow \frac{\mu_2}{\lambda_1 + \lambda_2 + \mu_2} = 0.13333333333333333$$

Checksum of probabilities:

$$\sum_{t=1}^{4} p(s_t) = 1.0$$

In case the agent would have chosen mec 2, the procedure would be the same.

E.g.2:

Let us see an example from another possible state.

$$s_0 \leftarrow [[[1, 1], [0, 1]], [0, -1]]$$

Killing a VNF is not an agent's action, it is the environment's mission, hence the agent will find itself into a new state. As it can be seen, only a VNF of type 2 will be killed, so the first four possible states (there's a total of 8) will be:

All new possible valid states:

1. $p(s_1) \leftarrow [[[1, 0], [0, 1]], [1, 0]]$
2. $p(s_2) \leftarrow [[[1, 0], [0, 1]], [0, 1]]$
3. $p(s_3) \leftarrow [[[1, 0], [0, 1]], [-1, 0]]$
4. $p(s_4) \leftarrow [[[1, 0], [0, 1]], [0, -1]]$

State transition probabilities:

$$p(s_1) \leftarrow \frac{\mu_1}{\mu_1 + \mu_2} \cdot \frac{\lambda_1}{\lambda_1 + \lambda_2 + \mu_1} = 0.13793103448275862$$
$$p(s_2) \leftarrow \frac{\mu_1}{\mu_1 + \mu_2} \cdot \frac{\lambda_2}{\lambda_1 + \lambda_2 + \mu_2} = 0.27586206896551724$$
$$p(s_3) \leftarrow \frac{\mu_1}{\mu_1 + \mu_2} \cdot \frac{\mu_1}{\lambda_1 + \lambda_2 + \mu_1} = 0.017241379310344827$$

$$p(s_4) \leftarrow \frac{\mu_1}{\mu_1 + \mu_2} \cdot \frac{\mu_2}{\lambda_1 + \lambda_2 + \mu_2} = 0.06896551724137931$$

Annex G: Algorithms Cross-validation tests

A way of verifying if all algorithms are correctly implemented is by cross-validation between them. By specifying certain scenarios and testing them, it is possible to check for any code issues.

Test 1:

$net_{status} : [0, 12, 0, 0, 0, 400, 0, 0]$
$num_{mecs} : 2$
$max_{mec\ capacity} : 12$
$arrival_{rate} : (\lambda_1 = 3), (\lambda_2 = 2)$
$departure_{rate} : (\mu_1 = 1), (\mu_2 = 0.5)$
$vnf1_{demands} : (1, 300)$
$vnf2_{demands} : (3, 50)$

Notice in the network status that only mec 2 has resources, so all algorithms can only assign VNF to such mec, making the algorithms outcoming deterministic.

```
TIMESTEP: 192.114       -> VNF ACCEPTED |       [[[0, 0], [1, 1]], [1, 0]]             |s/a:[6,0]
bw1: 0
bw2: 350

a) Action: 0
Timestep: 192.114       -> VNF REJECTED |       s/a:[6,0]

       ·) Arrival time@: 192.114
       ·) Killing time@: 194.217
       ·) VNF of type: 0

Episode: 1
Average reward: 486.0
Episode reward: 486.0
Rejections: 514
Served VNF: 486

Total rejections for every test: [532, 518, 491, 501, 517, 549, 528, 500, 538, 514]
Average rejections number: 518.8

Processing time: 8.7270

(base) imac-de-carlos:MDP carlos$
```

Fig. G.1 MDP test results

```
192.114: OK          [1, 300] |       0      8      0      4      0      50     0      350     | Q:[5,1]
192.114: REJECTED             [1, 300] |     0      8      0      4      0      50     0      350     | Q:[5,1]



Episode: 1
Average reward: 486.0
Episode reward: 486.0
Current epsilon: 1.0000
Rejections: 514
Served VNF: 486
Processing time: 0.0238
States: 16

Total rejections for every test: [532, 518, 491, 501, 517, 549, 528, 500, 538, 514]
Average rejections number: 518.8

(base) imac-de-carlos:RL_Q_LearningV4 carlos$
```

Fig. G.2 Q-Learning test results

```
2A
192.114: REJ            ([1, 300], None)        [0, 8, 0, 4, 0, 50, 0, 350]      | vnf: 999



Total rejections for every test: [532, 518, 491, 501, 517, 549, 528, 500, 538, 514]
Average rejections number: 518.8

(base) imac-de-carlos:Greedy carlos$
```

Fig. G.3 Best Fit test results

Test 2:

$net_{status} : [4, 12, 0, 0, 1000, 400, 0, 0]$
$num_{mecs} : 2$
$max_{mec\ capacity} : 12$
$arrival_{rate} : (\lambda_1 = 3), (\lambda_2 = 3)$
$departure_{rate} : (\mu_1 = 1), (\mu_2 = 1)$
$vnf1_{demands} : (1, 300)$
$vnf2_{demands} : (1, 300)$

```
TIMESTEP: 168.185       -> VNF ACCEPTED |       [[[1, 0], [1, 0]], [1, 0]]             |s/a:[14,0]
bw1: 300
bw2: 300

a) Action: 0
   after action (0): [[2, 0], [1, 0]]

TIMESTEP: 168.261       -> KILLING VNF |        ([[2, 0], [1, 0]], [-1, 0]) killing from mec 2  |s:[75]
   after kill: [[2, 0], [0, 0]]

Episode: 1
Average reward: 532.0
Episode reward: 532.0
Rejections: 468
Served VNF: 532

Total rejections for every test: [490, 447, 467, 430, 464, 461, 467, 482, 452, 468]
Average rejections number: 462.8

Processing time: 7.5474

(base) imac-de-carlos:MDP carlos$
```

Fig. G.4 MDP test results

```
168.261: ** KILLING VNF ** | vnf_arrival_time@: 165.249 | [1, 300] back to MEC: 1
RESOURCES UPDATED AFTER KILLING: [2, 12, 2, 0, 400, 400, 600, 0]



Episode: 1
Average reward: 532.0
Episode reward: 532.0
Current epsilon: 1.0000
Rejections: 468
Served VNF: 532
Processing time: 0.0184
States: 8

Total rejections for every test: [490, 447, 467, 430, 464, 461, 467, 482, 452, 468]
Average rejections number: 462.8

(base) imac-de-carlos:RL_Q_LearningV4 carlos$
```

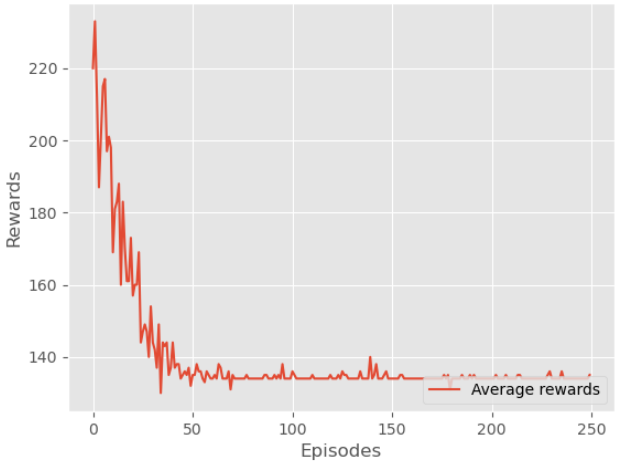
Fig. G.5 Q-Learning test results

```
168.261: KILL          ([1, 300], 1)          [2, 12, 2, 0, 400, 400, 600, 0]  | vnf_arrival_time@: 165.249



Total rejections for every test: [490, 447, 467, 430, 464, 461, 467, 482, 452, 468]
Average rejections number: 462.8

(base) imac-de-carlos:Greedy carlos$
```

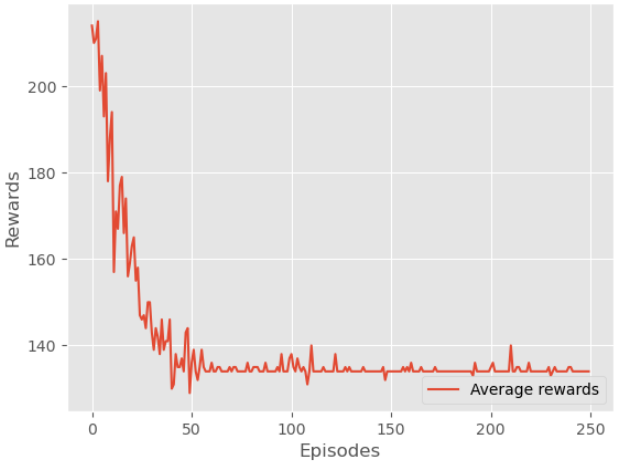
Fig. G.6 Best Fit test results

In this test, although two VNF are defined, they have the same values, in reality, it is like having only one VNF type. There's not enough complexity for either the Q-Learning or the MDP to learn, ending in a deterministic outcome when assigning VNF to mecs.

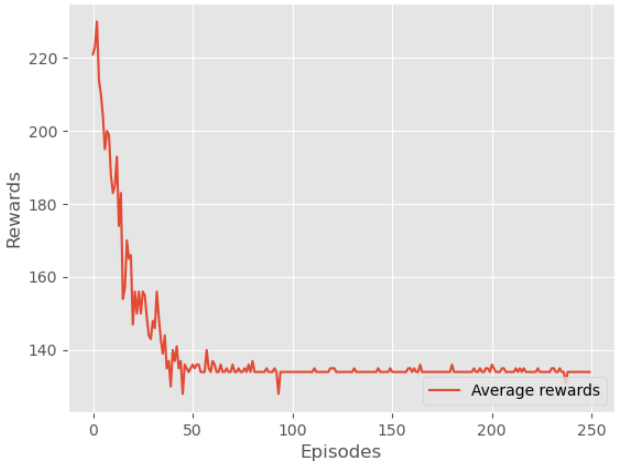
Annex H: Influence of Q-Learning parameters simulation results



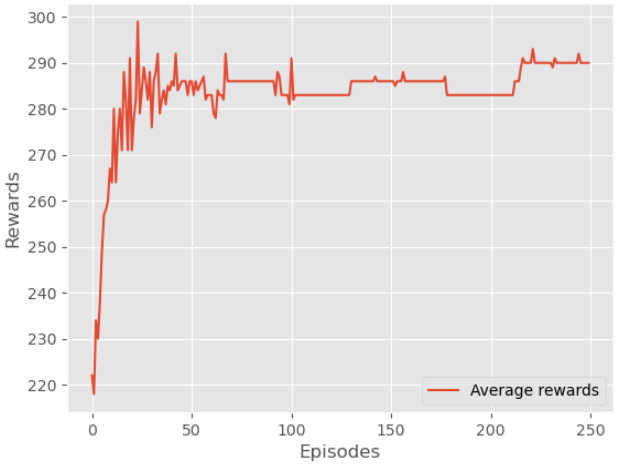
$\alpha = 0; \gamma = 0.001$



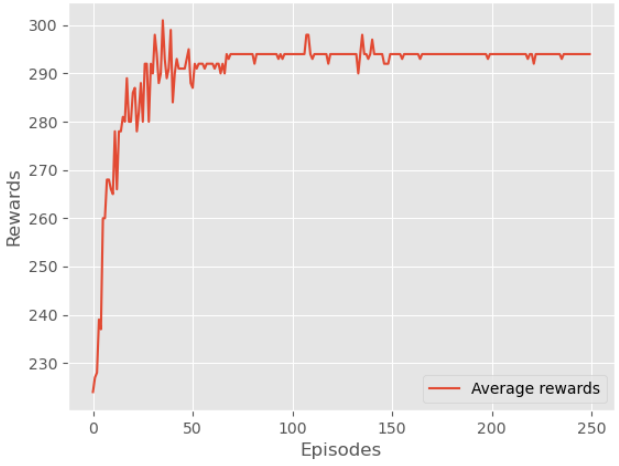
$\alpha = 0; \gamma = 0.5$



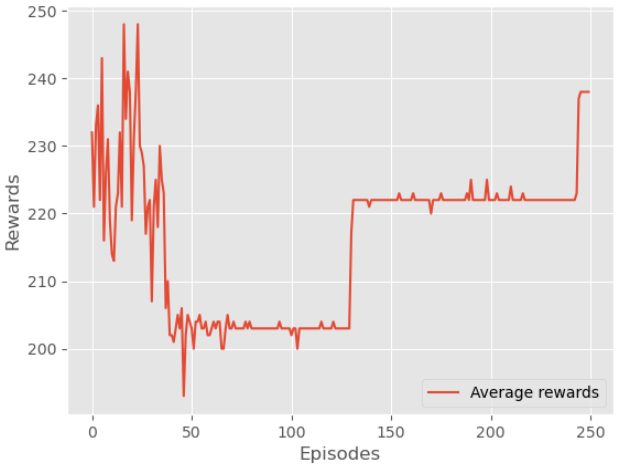
$\alpha = 0; \gamma = 0.999$



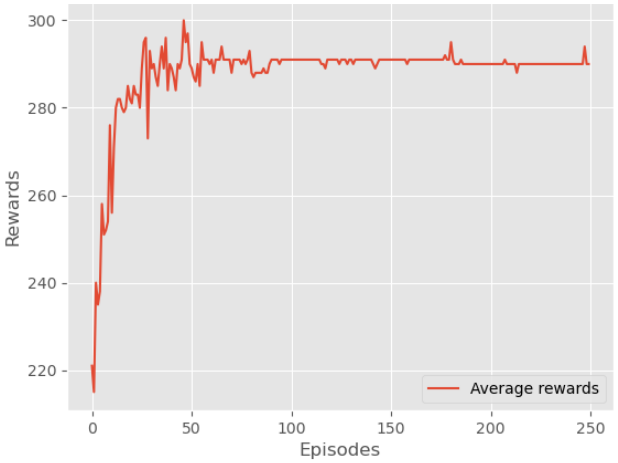
$\alpha = 0.5; \gamma = 0.001$



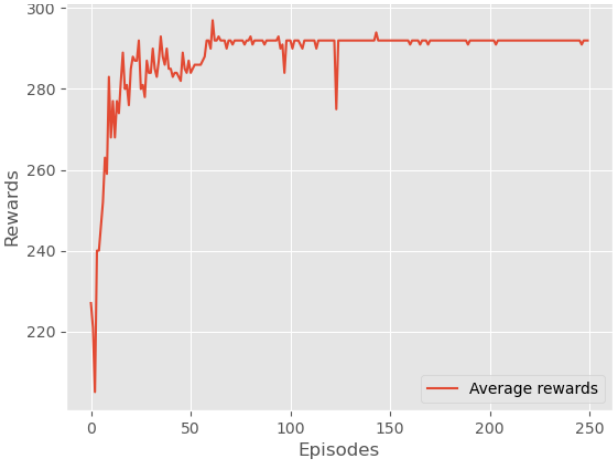
$\alpha = 0.5; \gamma = 0.5$



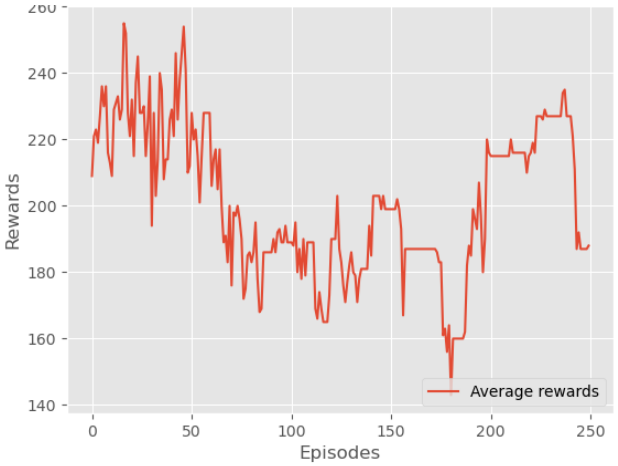
$\alpha = 0.5; \gamma = 0.999$



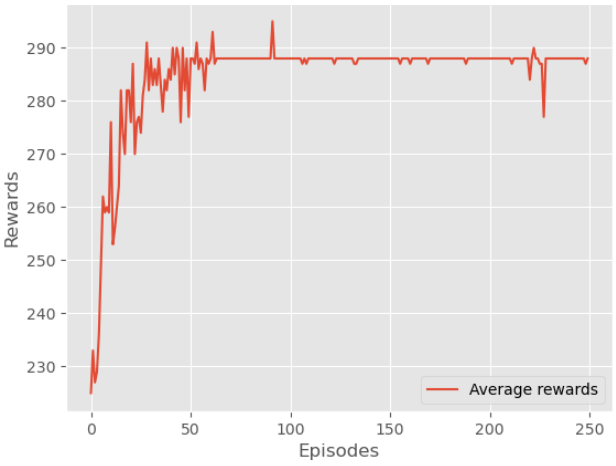
$\alpha = 0.999; \gamma = 0.001$



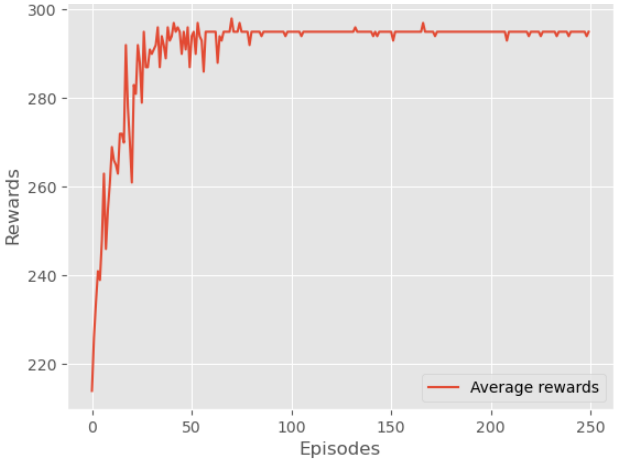
$\alpha = 0.999; \gamma = 0.5$



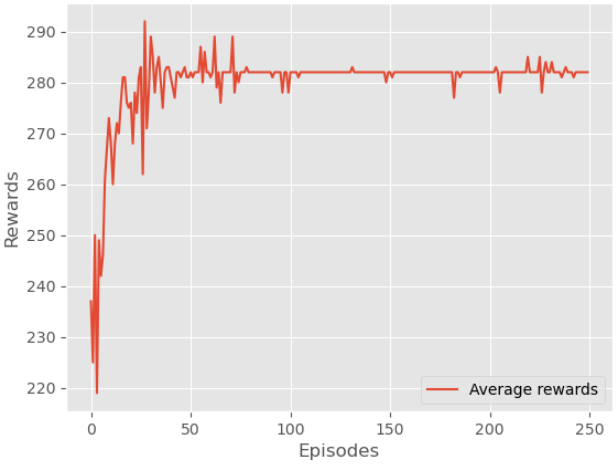
$\alpha = 0.999; \gamma = 0.999$



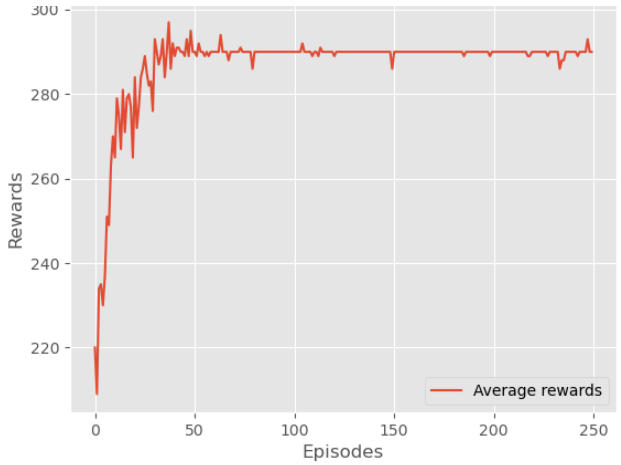
$\alpha = 0.001; \gamma = 0$



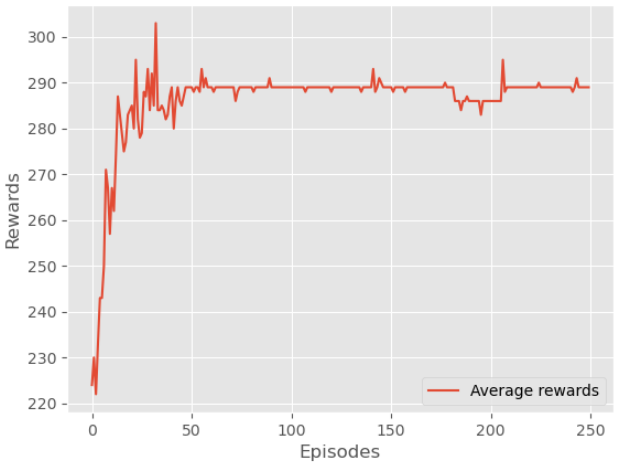
$\alpha = 0.5; \gamma = 0$



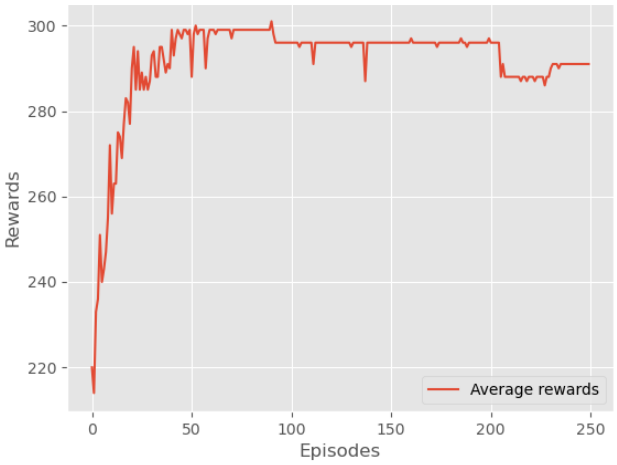
$\alpha = 0.999; \gamma = 0$



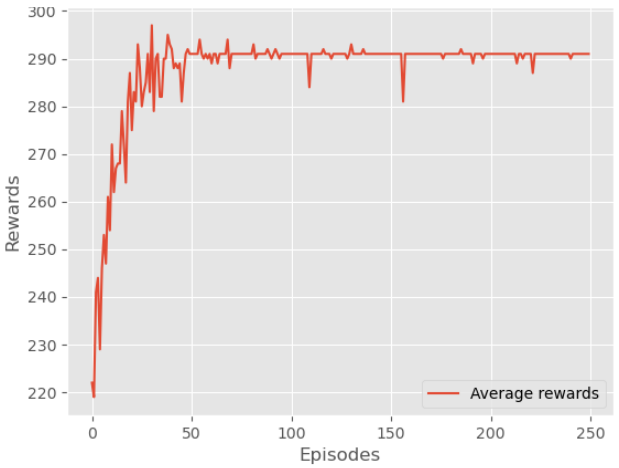
$\alpha = 0.001; \gamma = 0.5$



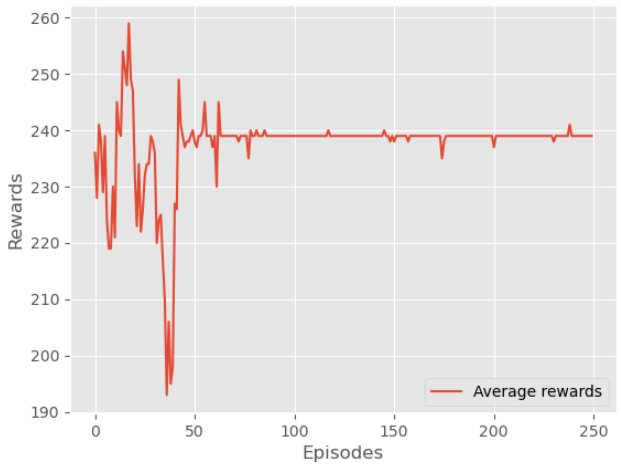
$\alpha = 0.5; \gamma = 0.5$



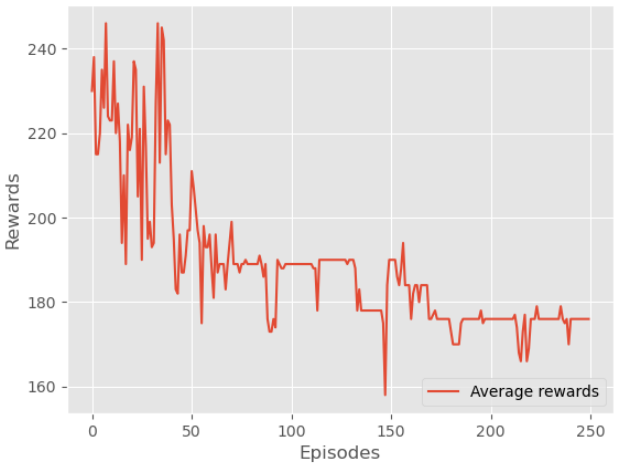
$\alpha = 0.999; \gamma = 0.5$



$\alpha = 0.001; \gamma = 0.999$



$\alpha = 0.5; \gamma = 0.999$



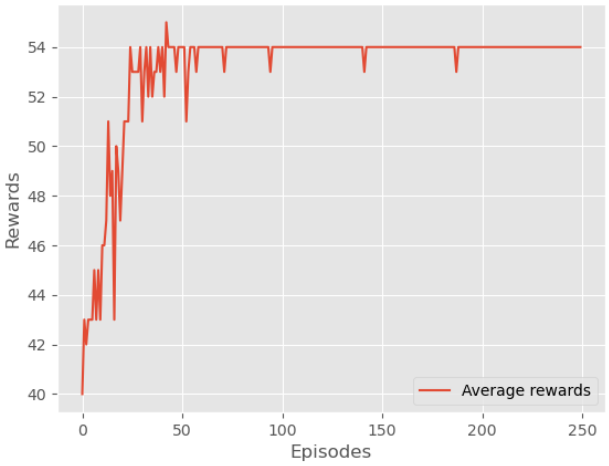
$\alpha = 0.999; \gamma = 0.999$

Fig. H.1. Influence of Q-Learning parameters

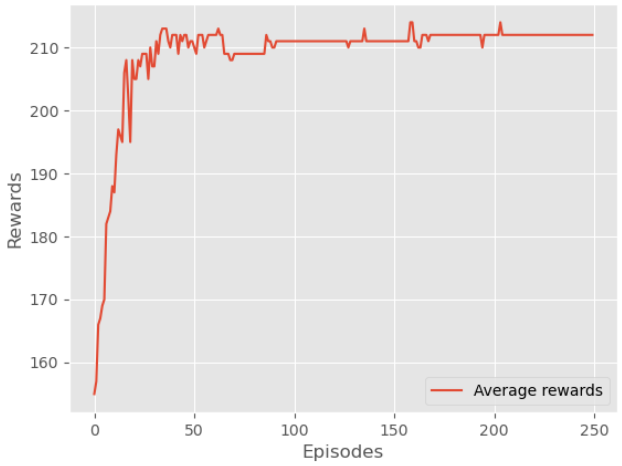
Annex I: Exploration and Exploitation simulation results

Training files	10
ϵ_{min}	0.001
No. episodes/training file[8]	250

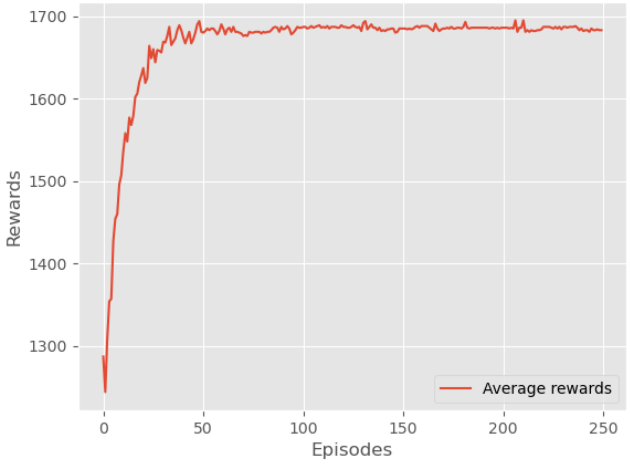
Fig. I.1. Simulation settings



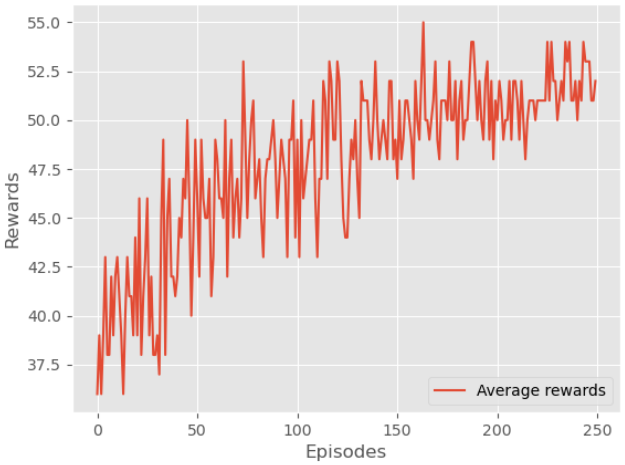
$\epsilon_{decay} = 0.1$
No. VNF /episode = 60



$\epsilon_{decay} = 0.1$
No. VNF /episode = 250

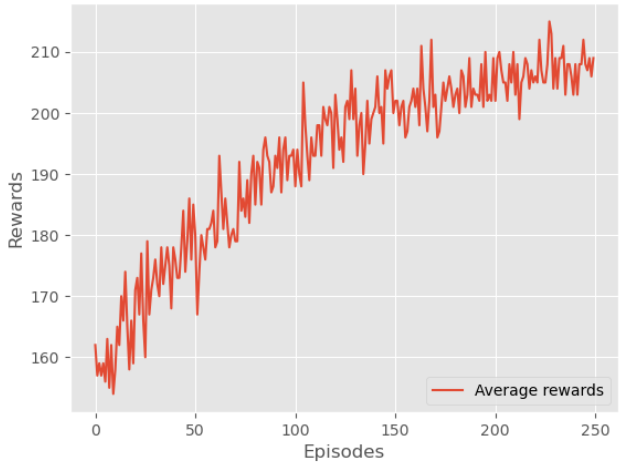


$\epsilon_{decay} = 0.1$
No. VNF /episode = 2000

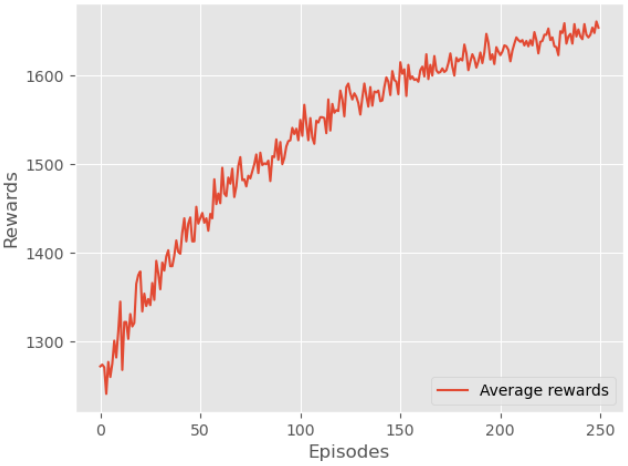


$\epsilon_{decay} = 0.01$
No. VNF /episode = 60

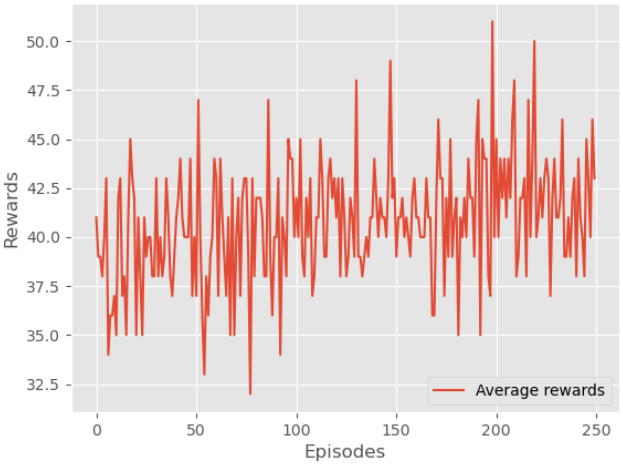
[8] For the first 9 simulation graphs.



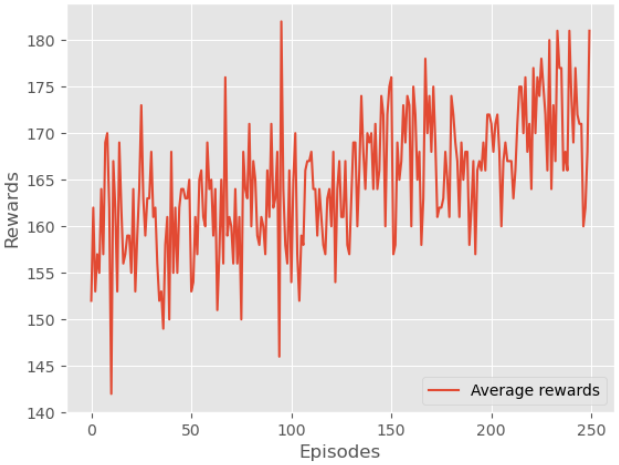
$\epsilon_{decay} = 0.01$
No. VNF /episode = 250



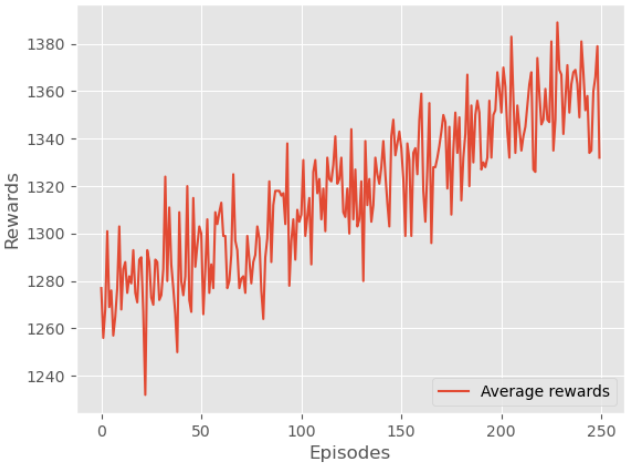
$\epsilon_{decay} = 0.01$
No. VNF /episode = 2000



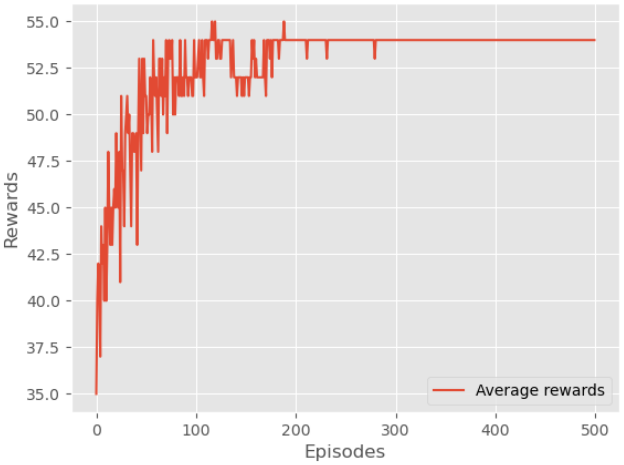
$\epsilon_{decay} = 0.001$
No. VNF /episode = 60



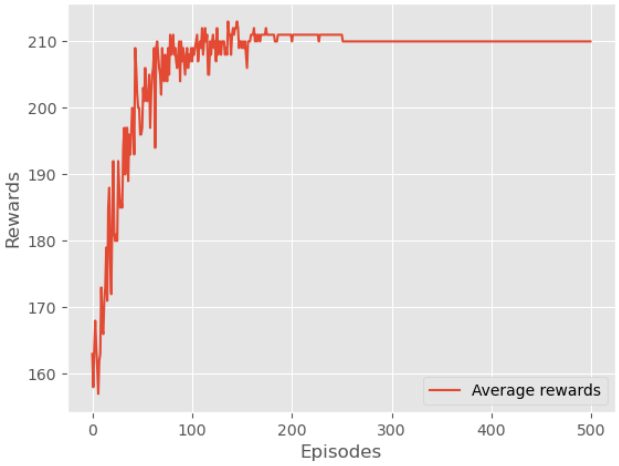
$\epsilon_{decay} = 0.001$
No. VNF /episode = 250



$\epsilon_{decay} = 0.001$
No. VNF /episode = 2000

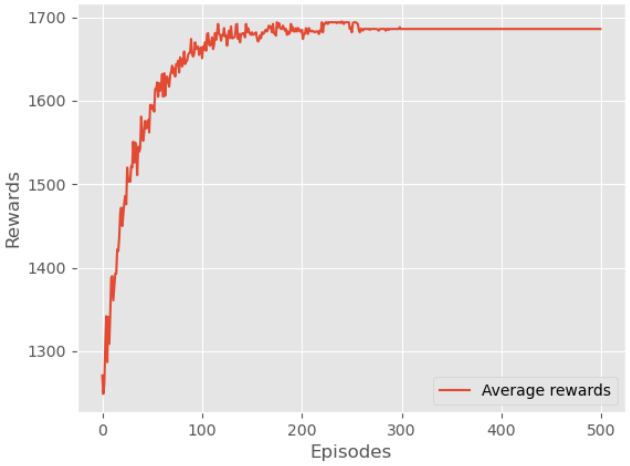


$\epsilon_{decay} = 0.03$
No. episodes/training file = 500
No. VNF /episode = 60



$\epsilon_{decay} = 0.03$

No. episodes/training file = 500
No. VNF /episode = 250



$\epsilon_{decay} = 0.03$

No. episodes/training file = 500
No. VNF /episode = 2000

Fig I.1. Exploration and exploitation results for different ϵ_{decay} values

Annex J: Boxplots for simulations 3.3, 3.4, 3.5 and 3.6

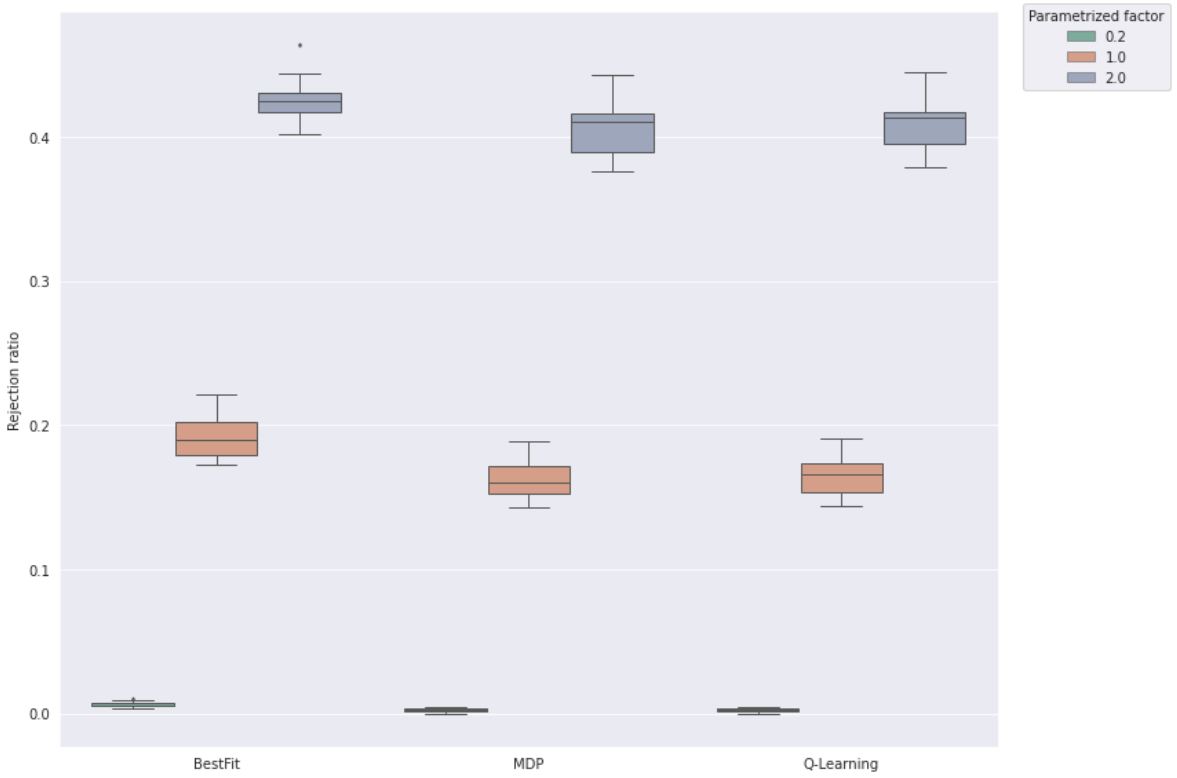


Fig. J.1. Rejection rate with respect to arrival rate

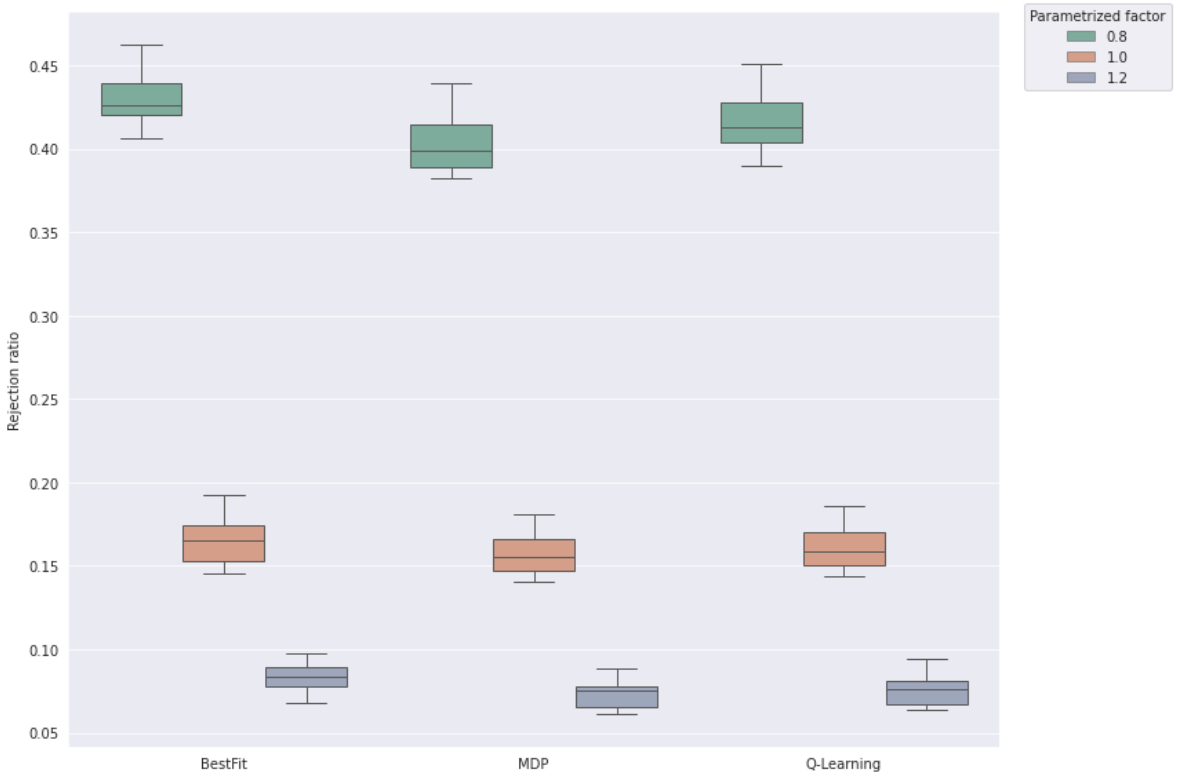


Fig. J.2. Rejection rate with respect to capacity

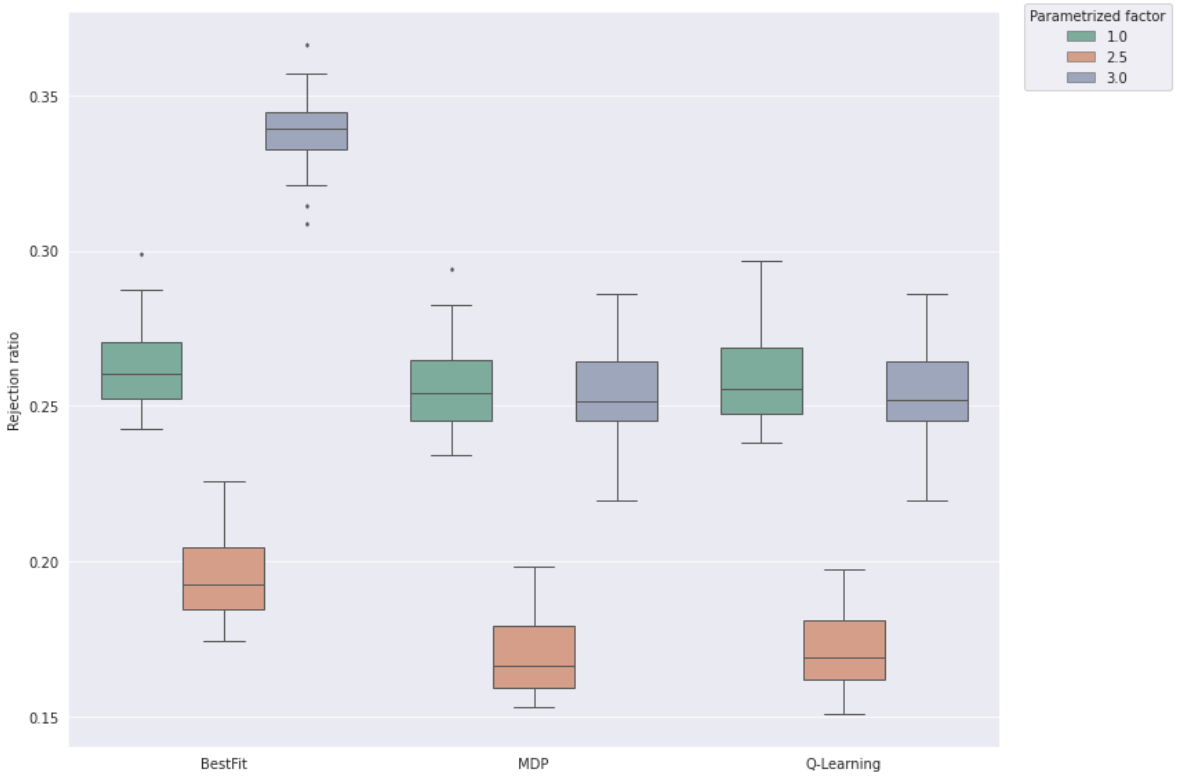


Fig. J.3. Rejection rate with respect to resource heterogeneity

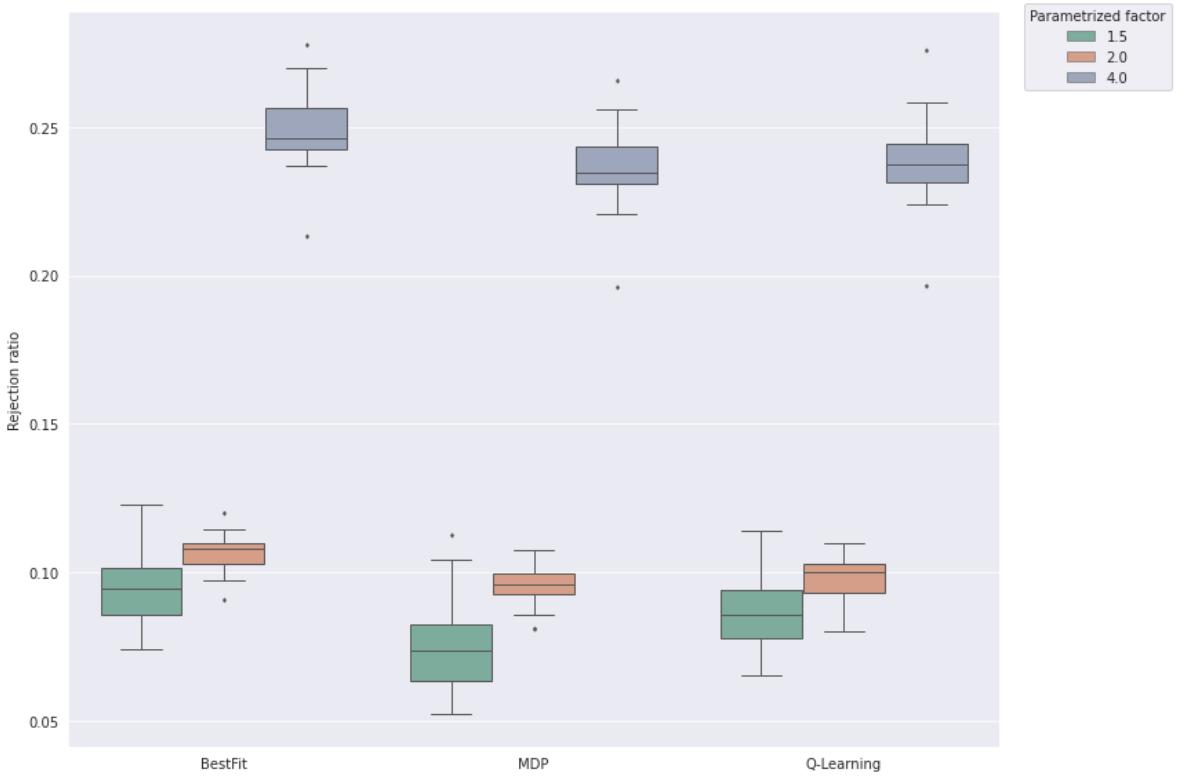


Fig. J.4. Rejection rate with respect to demand heterogeneity