"Application of Reinforcement Learning for optimizing the inventory management of clinical trials"

Escobar Nieves, Clara

ABSTRACT

On the one hand, clinical trials are of great importance in discovering new treatments for diseases. They teach research things that cannot be learned in the laboratory, that is, what does and does not work in humans. What's more, they are a really helpful tool for deciding whether the side effects of a new drug or treatment are acceptable compared to the potential benefits. Since results are not known at the beginning, this process is quite uncertain and effective management of doses is required. On the other hand, Reinforcement Learning is a Machine Learning paradigm different from Supervised Learning and Unsupervised Learning. Unlike these two methods, Reinforcement Learning is used when we want a system to accomplish a task by understanding the task by itself and the system must learn based on the "trial and error" rule. In recent years, several Reinforcement Learning applications have brought great advances in different fields. However, this new paradigm has not been used for optimizing the inventory management of clinical trials, so that's why this master thesis addresses this line of investigation. The contributions of this master thesis are: (1) The definition of the Markov Decision Process formulation for the current problem, (2) The application of Tabular Methods to solve the problem, and (3) The improvement of the used Tabular Methods by modifying their respective Q-Tables. Results show a considerable improvement after the Q-Table's modification. However, the system does not achieve a great performance mainly due to the high dimensionality of the prob...

CITE THIS VERSION

Escobar Nieves, Clara. *Application of Reinforcement Learning for optimizing the inventory management of clinical trials.* Ecole polytechnique de Louvain, Université catholique de Louvain, 2022. Prom. : Chevalier, Philippe. http://hdl.handle.net/2078.1/thesis:35158

École polytechnique de Louvain

Application of Reinforcement Learning for optimizing the inventory management of clinical trials

Author : **Clara ESCOBAR NIEVES**
Supervisor : **Philippe CHEVALIER**
Readers : **Julien HENDRICKX, Raphaël JUNGERS**
Academic year 2021–2022
Master [120] in Mathematical Engineering

Application of Reinforcement Learning for optimizing the inventory management of clinical trials

Clara Escobar Nieves

Supervisor: Philippe Chevalier

Master: Master in Mathematical Engineering

Readers: Julien M. Hendrickx and Raphaël Jungers

Dissertation presented to fulfill the requirements for the double Master's degree in Industrial Engineering and in Management Engineering at the Polytechnic University of Catalonia in the international exchange program with the Université Catholique de Louvain.

June, 2022

Declaration

I hereby confirm that this thesis was written independently by myself without the use of any sources beyond those cited, and all passages and ideas taken from other sources are cited accordingly.

Clara Escobar Nieves

Acknowledgements

First and foremost, I want to thank my thesis supervisor, Professor Philippe Chevalier, for giving me the chance to come to Louvain-la-Neuve and for supporting me constantly and answering all my questions and doubts.

I would also like to thank Professor Alejandro Lamas, because this project would not have been possible without his support.

As later in this document we will see, Reinforcement Learning is about an agent, which is put into an unknown environment and by trial and error, it has to learn how to behave to get the maximum possible reward.

In a sense, I must recognise that sometimes during the development of this project I felt like the agent. I had to learn about a new environment and its rules to satisfactorily reach my goal, which is the elaboration of this master thesis. However, unlike the agent, I had the unconditional support of my family, which has supported me to overcome all difficulties I had to face and believe in me even when I did not, as they always did throughout my life.

Université Catholique de Louvain
Louvain-la-Neuve, June 2022.

Clara Escobar Nieves

Abstract

On the one hand, clinical trials are of great importance in discovering new treatments for diseases. They teach research things that cannot be learned in the laboratory, that is, what does and does not work in humans. What's more, they are a really helpful tool for deciding whether the side effects of a new drug or treatment are acceptable compared to the potential benefits. Since results are not known at the beginning, this process is quite uncertain and effective management of doses is required.

On the other hand, Reinforcement Learning is a Machine Learning paradigm different from Supervised Learning and Unsupervised Learning. Unlike these two methods, Reinforcement Learning is used when we want a system to accomplish a task by understanding the task by itself and the system must learn based on the "trial and error" rule.

In recent years, several Reinforcement Learning applications have brought great advances in different fields. However, this new paradigm has not been used for optimizing the inventory management of clinical trials, so that's why this master thesis addresses this line of investigation.

The contributions of this master thesis are: (1) The definition of the Markov Decision Process formulation for the current problem, (2) The application of Tabular Methods to solve the problem, and (3) The improvement of the used Tabular Methods by modifying their respective Q-Tables.

Results show a considerable improvement after the Q-Table's modification. However, the system does not achieve a great performance mainly due to the high dimensionality of the problem.

Keywords: Reinforcement Learning, Tabular Methods, Q-Learning, SARSA, Inventory Management, Clinical Trials.

Contents

List of Figures

List of Tables

List of Equations

Acronyms

AI Artificial Intelligence.

CD Central Depot.

DNN Deep Neural Networks.

DP Dynamic Programming.

DQN Deep Q-Network.

LD Local Depot.

MC Monte Carlo.

MDP Markov Decision Process.

ML Machine Learning.

POMDP Partially Observable Markov Decision Process.

RL Reinforcement Learning.

SC Supply Chain.

SCN Supply Chain Network.

SL Service Level.

SLR Systematic Literature Review.

TD Temporal-Difference.

CHAPTER 1

Introduction

Nowadays, the correct management of inventories is increasingly important, not only because of the derived costs, but also for the environmental impact and the customer Service Level (SL), where Type 1 SL is the α SL and Type 2 SL is the β SL. Their differences and definitions will be explained later in more detail.

Regarding short-lived products, as is the present case, it is crucial to have effective inventory management systems in order to provide competitive advantages for companies and customers. Concerning clinical trials, it is obvious that this is a field of great importance, since the service quality implies consequences for the health of patients.

Efficient clinical trials management is required independently of the size of the trials, as it involves a huge investment of time, money and people. Human and financial resources are not unlimited. Consequently, it is crucial to ensure that clinical trials management is implemented efficiently.

Although most of the literature is based on the impact on patient's health, even using probabilistic prediction tools [2], one common factor observed in satisfactory trials management is the implementation of a correct management system [3]. In addition, trials management is an essential key competence to offering high-quality trials.

By using mathematical programming, companies and researchers have made approximate approaches to find near-optimal policies for inventories management. Nevertheless, Reinforcement Learning (RL) is based on learning what to do by associating situations with actions, to maximise a numerical reward signal. The agent does not know in advance what actions to take, so it must instead learn which actions produce the best reward by trying them out. As it will be later explained, in most interesting cases, the actions not only affect the immediate reward but also the following situations and therefore the future rewards [1].

Taking it into account, the research work carried out in this master thesis is characterized as follows:

By the application of Reinforcement Learning, find the best policies for optimizing the inventory management of clinical trials.

To begin this research, the first chapter is devoted to specifying the scope, methodology and objectives of this master thesis. Next, at the beginning of each new chapter, it will be also indicated what has been done.

1.1 SCOPE AND METHODOLOGY

This master thesis focuses firstly on how to express the current problem in order to solve it by the application of RL algorithms. This includes:

- Defining the problem from the RL point of view.
- Implementing the code that allows executing the RL algorithms.

In Chapter 2 the theoretical framework of RL will be presented, by placing RL inside the Machine Learning (ML) theory, explaining its main ideas and formulating the problem as an Markov Decision Process (MDP).

Then in Chapter 3, an analysis of the present state of the art in several fields is done. This covers the latest RL applications, which do not include the present problem.

After analysing the present state of the art, the problem formulation is done in Chapter 4, to allow to "build" the problem in Chapter 5.

Then, the particular problem of the optimization of the inventory management of clinical trials will be solved through RL. Experimental results are developed in Chapter 6.

Finally, in Chapter 7 we explain our conclusions and propose further work.

1.2 OBJECTIVES

RL problems are commonly formulated as MDPs. That is why the first of the objectives of this master thesis is to find the elements in common between an MDP and the problem of inventory management of clinical trials, which will be treated as a sequential decision-making problem.

One of the main RL problems is the dimensionality of the observation space, in fact, there are various algorithms and methodologies for solving problems through the application of RL. Therefore, the second objective is finding the appropriate algorithm and methodology for the correct resolution of the problem.

Finally, as the main goal of the present project is practical problem solving, the target is to put all the theoretical concepts into practice by solving a particular case through the theory provided by RL.

As a summary of the previous paragraphs, the problem of inventory management of clinical trials will be formulated as an MDP, the most appropriate RL algorithm and methodology will be looked for and finally, the best management policies obtained will be analysed.

CHAPTER 2

Theoretical Framework

Before going more deeply into the resolution of the problem, it is necessary to explain some basic concepts to better understand the formulation and development of the problem.

In this chapter a set of definitions and key ideas related to MDPs, Artificial Intelligence and RL are presented.

2.1 MARKOV DECISION PROCESS

Markov Decision Processes (MDPs) are a classical formalization of sequential decision making. In this type of process actions have an important influence in not just immediate rewards, but also in subsequent situations or states and future rewards [1].

As later we will go into RL in detail, it is important to consider that MDPs allow mathematical modelling of the RL problem. Therefore, it is necessary to make some precise theoretical statements.

An MDP is defined as a five-tuple $M = \langle S, A, P, R, \gamma \rangle$, where:

S is the set of states.

A is the set of actions.

P is the state-transition function.

R is the reward function.

$\gamma \in [0, 1]$ is the discount factor.

The elements of the previously defined five-tuple can be related in the following way:

There is a decision-maker, called the agent, who interacts with the environment in which it is. The interactions between the agent and the environment occur sequentially over time. At each time step the agent obtains some representation of the state of the environment, then, from this representation, the agent selects an action to perform. The environment then transitions to a new state and the agent receives a reward as a consequence of the previous action.

The process described, which consists of selecting an action from a given state, transitioning to a new state and receiving a reward, occurs sequentially over and over again. It is formally defined as a trajectory, which shows the sequence of states, actions and rewards.

To better understand this process we can observe Figure 2.1. At each time step $t = 0, 1, 2, \cdots$, the agent receives the representation of the environment's state, $S_t \in S$. From this state, the agent selects an action $A_t \in A$ which gives us the state-action pair (S_t, A_t).

Time is increased, so one time step later, $t + 1$, as a consequence of having taken the previous action, the agent receives a numerical reward $R_{t+1} \in R$ and the environment is transitioned to a new state, $S_{t+1} \in S$.

Consequently, the trajectory that represents the described sequential process is:

$$S_0, A_0, R_1, S_1, A_1, R_2, S_2, A_2, R_3, \cdots$$

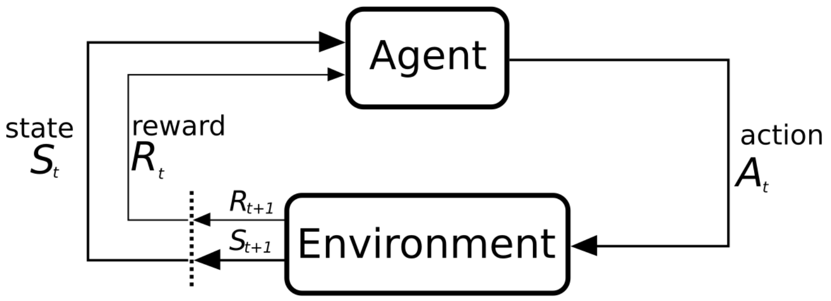

Figure 2.1: Diagram of a Markov Decision Process (Sutton and Barto, 2018) [1].

The state-transition function P defines the dynamics of the MDP because it indicates the probability of moving from one state to the next state. Here it is important to remember the property of MDPs, which states that the environment's response to the agent at time step t only considers the state and action at the previous time step $t - 1$ [4].

The reward function R is a scalar value that represents the reward the agent gets when transitioning from one state to the next one while performing an action $A_t \in A$.

The discount factor γ is used by the agent to adjust the importance of rewards over time. So if $\gamma = 0$ the agent is more interested in near time rewards that are more likely than late rewards, but if $\gamma = 1$, the agent will consider the possible future rewards based on the evaluation of its actions.

Once defined the discount factor γ and in order to understand the previous paragraph, it can be now defined how the return is calculated from γ and the rewards for continuing tasks. The discounted return is denoted as G_t and for every time step t is computed as the sum of subsequent rewards, taking into account that late rewards are multiplied by the discount factor raised to the power of the number of steps the agent is away. It is calculated as follows:

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \ldots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{2.1}$$

From Equation 2.1 it can be noted that if $\gamma = 0$, the discounted return will be the immediate reward without any subsequent state. It is commonly denoted as making the agent *"myopic"* (or short-sighted).

Since this is a sequential decision-making problem, the agent's goal is to maximise the total quantity of rewards it receives after having performed the corresponding actions in certain states. Therefore, the agent seeks to maximise the cumulative rewards it receives over time and not just the immediate reward.

Regarding the dimension of the different elements of the previously defined five-tuple, we will assume that the sets S, A, P and R have a finite number of elements. Therefore, the decision process is defined as a finite MDP.

2.2 ARTIFICIAL INTELLIGENCE

Nowadays, Artificial Intelligence (AI) is defined as a branch of science and technology that creates intelligent machines and computer programs to perform various tasks which require human intelligence. That is a system that mimics various functions that a human can do [5].

As a result, the tools developed by AI can be applied to different fields in order to find new and improved solutions to existing problems, such as in the field of transport [6], healthcare [7] and education [8], among others.

ML is an area that belongs to AI and is responsible for designing algorithms with the ability to learn by themselves, without being explicitly programmed.

Once AI and ML have been defined, it is necessary to consider the following three basic approaches: Supervised Learning, Unsupervised Learning and Reinforcement Learning.

Supervised Learning makes use of labelled datasets to train algorithms to classify data or predict outcomes accurately.

Unsupervised Learning is a method that uses ML algorithms to analyse and cluster unlabeled data sets. Unlike Supervised Learning, there is no prior knowledge.

Reinforcement Learning is distinguished from the other two computational approaches by its emphasis on learning by an agent from direct interaction with its environment, so exemplary supervision or complete models of the environment are not required.

2.3 REINFORCEMENT LEARNING

As previously defined, RL is one of the three basic ML paradigms and focuses on how the agent should act in the environment to maximise the cumulative reward.

In this section devoted to RL, we will make an explanation of the concepts of policies, value functions and their corresponding optimal policies and optimal value functions. Also, one of the most recurrent equations in RL literature will be considered, the Bellman equation.

After having explained those concepts, this section will attend the *exploration-exploitation trade-off*, also known as the *exploration-exploitation dilemma*, which is the tough choice between obtaining new knowledge and using that knowledge to improve performance. As a consequence, ε-greedy methods will be also explained.

2.3.1 Policies and Value Functions

To analyse the RL environment, it is necessary to consider the previously defined MDP five-tuple, as well as a policy, which will be denoted as π, the state-value function $v_\pi(s)$, and the action-value function $q_\pi(s, a)$.

These concepts can be defined as follows:

> **Policy $\boldsymbol{\pi}$** is a function that maps from perceived states of the environment to actions to be taken when the agent is in those states, based on probabilities of selecting each possible action from those states. So, if the agent follows policy π at time t, then $\pi(a|s)$ denotes the probability that $A_t = a$ if $S_t = s$. This means the probability of taking action a in state s at time t under policy π.
>
> **State-value function $\boldsymbol{v_\pi(s)}$** is an estimation of how good any given state is for an agent following policy π. Formally, it is defined as the expected return from starting from state s at time t and following policy π.
>
> **Action-value function $\boldsymbol{q_\pi(s, a)}$** similarly to the state-value function $v_\pi(s)$, action-value function estimates how good it is for the agent to take any given action from a given state while following policy π. Formally it can be defined as the expected return from starting from state s at time t, taking action a, and following policy π.

The latter two terms, $v_\pi(s)$ and $q_\pi(s, a)$, can be estimated from experience by the agent because as the agent is in constant interaction with the environment, it can keep an average for each state it encountered of the returns that have followed that state. As the interaction tends to infinity, by the law of large numbers, the average will converge to the state's value $v_\pi(s)$. Besides, if separate averages for each action taken in a state are maintained, these averages will also converge to the action's values, $q_\pi(s, a)$.

Regarding the mathematical formulation, we have the following:

State-value function $\boldsymbol{v_\pi(s)}$:

$$v_\pi(s) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s\right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s\right], \text{ for all } s \in \mathcal{S} \tag{2.2}$$

Action-value function $q_\pi(s, a)$:

$$q_\pi(s,a) \doteq \mathbb{E}_\pi \left[G_t \mid S_t = s, A_t = a\right] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right] \tag{2.3}$$

Once defined $v_\pi(s)$ and $q_\pi(s,a)$ their relationship can be formalized. Note that if :

$$\sum_a \pi(a \mid s) = 1 \tag{2.4}$$

It can be asserted that:

$$v_\pi(s) = \sum_a \pi(a \mid s) \cdot q_\pi(s,a) \tag{2.5}$$

2.3.2 Optimal Policies and Optimal Value Functions

One of the most common elements of many RL algorithms is the Bellman equation. This equation breaks up the value function into two parts:

- The immediate reward.
- The discounted future values.

Thanks to this equation, the computations are simplified. The Bellman equation is now presented for the state-value function $v_\pi(s)$ and the action-value function $q_\pi(s,a)$:

Bellman equation for the state-value function

$$v_\pi(s) = \mathbb{E}_\pi \left[R_{t+1} + \gamma v_\pi\left(S_{t+1}\right) \mid S_t = s\right] \tag{2.6}$$

Bellman equation for the action-value function

$$q_\pi(s,a) = \mathbb{E}_\pi \left[R_{t+1} + \gamma q_\pi\left(S_{t+1}, A_{t+1}\right) \mid S_t = s, A_t = a\right] \tag{2.7}$$

So, the Bellman equation relates the value of a state or state-action and the subsequent values.

Recap that the agent's goal is to maximise the total cumulative reward in the long term. The policy that maximises the total cumulative reward is the optimal policy.

When comparing policies, a policy π is said to be better than or the same as policy π' if the expected return of π is greater than or equal to the expected return of π' for all states. That means:

$$\pi \geq \pi' \text{ if and only if } v_\pi(s) \geq v_{\pi'}(s) \text{ for all } s \in S \tag{2.8}$$

Therefore, an optimal policy π_* meets that $\pi_* \geq \pi$ for all policies π. It is guaranteed that an optimal policy exists but may not be unique. There could be different optimal policies, however they all share the optimal state-value function and the optimal action-value function.

It is trivial that an optimal value function is the one that gives the maximum value compared to all other value functions. As the main objective when solving an MDP is finding the optimal value function, we can define the optimal state-value function and the optimal action-value function as follows:

Optimal state-value function $v_*(s)$

$$v_*(s) \doteq \max_\pi v_\pi(s) \tag{2.9}$$

Optimal action-value function $q_*(s,a)$

$$q_*(s,a) \doteq \max_\pi q_\pi(s,a) \tag{2.10}$$

The Bellman equation is used for estimating the value of states and states-actions as a function of the subsequent values. Once we have found the optimal value functions v_* and q_* the optimal policies can be obtained with the following expressions:

$$\begin{aligned} v_*(s) &= \max_a \mathbb{E}\left[R_{t+1} + \gamma v_*\left(S_{t+1}\right) \mid S_t = s, A_t = a\right] \\ &= \max_a \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma v_*\left(s'\right)\right] \end{aligned} \tag{2.11}$$

$$\begin{aligned} q_*(s,a) &= \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*\left(S_{t+1}, a'\right) \mid S_t = s, A_t = a\right] \\ &= \sum_{s',r} p\left(s', r \mid s, a\right)\left[r + \gamma \max_{a'} q_*\left(s', a'\right)\right] \end{aligned} \tag{2.12}$$

Finally, we can conclude that the Bellman equation is a keystone to find the optimal values of the value functions in order to obtain an optimal policy.

2.3.3 ε-Greedy Methods and the Exploration-Exploitation Trade-Off

Given that RL is an area of ML that teaches the agent to take actions in order to maximise rewards and that the agent has no knowledge about the different states, the actions it can take in all the states, the associated rewards it would obtain and neither the transition to resulting states, it is necessary for the agent to learn by interacting with the environment [9].

When facing a decision making problem, the agent can wonder if it is better to go for the decision that seems to be optimal or gather new information that could be helpful in order to make a better decision once it has more information because the current knowledge could be inaccurate.

In this case, there are two possible behaviours, exploitation and exploration, which can be described as follows:

Exploitation: Selecting one of the actions whose estimated value is the greatest, we will call these actions *"Greedy Actions"*.

Exploration: Selecting one of the non-greedy actions, in order to improve the estimate of the non-greedy action's value.

So, exploration pursues a long-term benefit because it allows the agent to improve its knowledge about each action that could lead to long term benefit, while exploitation makes use of the agent's current estimated value and acts greedily to get the most reward.

Based on these two possible behaviours, we can define two action-value methods, *Greedy Methods* and *ε-Greedy Methods*:

Greedy Methods: Selecting one of the actions with the highest estimated value, which is, one of the greedy actions. The objective of these methods is to exploit current knowledge to maximise immediate reward.

ε-Greedy Methods: Acting greedily most of the time but with a small probability ε choosing to explore.

ε-Greedy can be represented as follows:

$$A_t \leftarrow \begin{cases} \text{argmax}\, Q_t(a) & \text{with probability } 1-\varepsilon \\ a \sim \text{Uniform}\left(\{a_1 \dots a_k\}\right) & \text{with probability } \varepsilon \end{cases} \tag{2.13}$$

Expression 2.13 indicates that the agent's selection at time step t will be a greedy action with probability (1 - ε) or it may be random with the probability of ε.

To summarise, on the one side, exploitation is appropriate to maximise near time rewards, but on the other side, exploration may produce a greater total reward in the long run. If there are many time steps ahead, it may be better to explore the non-greedy actions to discover which of them are better than the greedy action.

CHAPTER 3

Literature Review

Requirements for proper inventory management are becoming more important since it is necessary to offer the best quality of service without having a large environmental impact and on operating costs.

Given that these requirements are increasingly demanding, it is necessary to apply new methods that provide the appropriate solutions, which is why this master's thesis' objective, is the application of RL for optimizing the inventory management of clinical trials.

This chapter begins with the present state of the art of the healthcare sector and its importance, then some articles concerning stochastic models and SL will be presented. To conclude, a literature review of the applications of RL will be done.

3.1 SYSTEMATIC LITERATURE REVIEW

To carry out this chapter, a Systematic Literature Review (SLR) approach has been adopted following the indicated procedure by Okoli and Schabram [10], which is, a standardized methodology that provides guidelines for developing theory in literature reviews.

The next eight-step guide for conducting a SLR was followed:

1. **Identification of the purpose:** Establishment of research objectives and intended goals.
2. **Draft protocol:** Detailed protocol to indicate how to execute the review.
3. **Practicality filter:** Disposal of articles deemed not useful for the investigation.
4. **Search for literature:** Explicit description of the details of the literature search and ensure the search's comprehensiveness.

5. Data extraction: Extract from each study the information that can be applicable and useful.

6. Quality appraisal: Establishment of the exclusion criteria for articles with not enough quality, depending on the research methodologies they employ.

7. Synthesis: Combination of the extracted facts and information from articles.

8. Review writing: After following the previous seven steps, it is necessary to report in sufficient detail the review's results.

In Figure 3.1 the eight-step guide is presented.

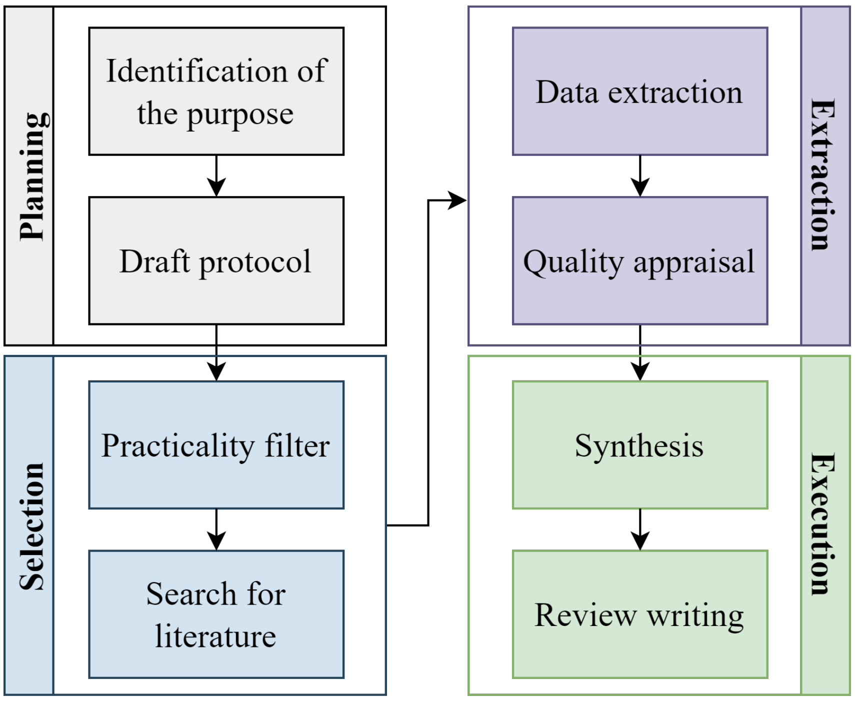

Figure 3.1: Steps of Systematic Literature Review (SLR). Source: Own elaboration.

As the scope of the present work is not to do an exhaustive literature review, each step is not explained in detail. Nevertheless, it is important to consider that studies were selected based on the following criteria:

- They must include the terms: Reinforcement Learning, Markov Decision Process, Inventory Management and Management of Clinical Trials. So the words used for the search were Reinforcement Learning, Markov Decision Process, Markov Chains, Reinforcement Learning Applications, Clinical Trials, Protocol, Service Level, Supply Chain Management, Perishable Inventory, Short-Lived Inventory and Inventory Systems.
- Based on the terms defined in the previous point, studies were searched in the Google Scholar, Scopus and ScienceDirect databases, and subsequently managed with the software Mendeley for its correct review and bibliographic referencing.
- Most recent studies published by recognized institutions have been prioritized.

It should be considered that the studies used as reference deal jointly with the mentioned topics, a fact that difficulties the corresponding analysis. Despite this and regarding the last step of the eight-step guide, review writing, the corresponding literature review is organized attending firstly to the present state of the art of the healthcare sector and its importance, secondly to stochastic models and SL and finally to different applications of RL methods.

3.1.1 State of the Art of the Healthcare Sector

In the field of management of clinical trials, many advances have been done. One of them is that since 2005 all clinical trials, before they may be considered for publication, should be registered in publicly available domains. This is a recommendation by the International Committee of Medical Journal Editors (ICMJE) [11].

Once the trial is accepted, the challenge is different. As recently seen in the current pandemic situation, the speed of the normal drug development pathway is unacceptable, therefore, there is considerable interest in expediting these procedures [12]. Although increasing the speed of the normal drug development involves different factors, its management can bring great benefits [3], like for example reducing drawbacks as well as costly and timely process with a high attrition rate [13].

One common factor observed in satisfactory trials management is the implementation of a correct management system, which is, the establishment and execution of effective management systems and techniques according to the needs of the trial. All clinical trials, regardless of the size, scope, costs or duration, require coordinated processes and systems [14].

The importance of correct management is known, in fact, there are several publications regarding protocols and guidelines for clinical trials management, as those presented in [15] and [16]. More explicitly, concerning the management of perishable items, there are studies to obtain the near-optimal inventory ordering policy by many different approaches:

- A dynamic programming and simulation-based approach are used for the inventory control of the platelet at the blood bank in [17].
- For the modelling of ordering policies of perishable pharmaceutical items an MDP is used in [18].
- By integer programming and a variable neighborhood search approach, the planning delivery routes for the supply of blood products to hospitals by a blood bank is done in [19].

Nevertheless, there is currently no branch of application of RL methods for the correct establishment of management policies for clinical trials that avoid the derived problems, since most of the failure can be due to practical problems with trial management rather than scientific problems.

Since the goal of this master thesis is not the evaluation of present inventory management methods of clinical trials but to apply RL algorithms for establishing management policies, we shall now proceed to the literature review relative stochastic models and SL, a previous step to the presentation of the current RL Applications.

3.1.2 Stochastic Modelling and Service Level

The existing gap between supply and demand has led to the development of several resource pooling strategies in the industry. Nevertheless, despite this intensive strategy's growth, their effective implementation is still a hurdle task for many Supply Chain (SC) actors. Furthermore, it is important to consider the SL requirements of customers, or as is the case of the present master thesis, patients.

As a consequence of the previous paragraph, the key challenge for suppliers and manufacturers is to maintain enough inventory in order to be able to supply all the demand on time.

In situations involving randomness, as it is the case, a probability model is often used to represent it, by assigning probabilities directly to events [20].

To go more deeply into this matter, the article [21], which has been classified as the most applicable to this master thesis, will be analysed first. It can be compared to the present problem regarding two aspects, the first one is that it considers a manufacturer that can be understood as our CD and many E-distributors, similar to our LDs. The second aspect is that it considers the type 2 SL, which as later it will be explained, is the proportion of demand that can be satisfied.

There are also differences:

- It is based on a profit model which considers customers and not patients.
- Following the same argumentative line of the previous bullet point, it classifies customers by their SL requirements and so they are served following a priority list. As a result, at least one customer's demand will be partially served and the rest of the demand, depending on their position in the list, might be fully satisfied or not. This is contrary to the point of view of the present master thesis, as all patients have the same rights to be treated.
- The objective is to determine the resource capacity level at the beginning of a period, instead of the amount sent to each E-distributor. These two concepts might be misunderstood because in our case we consider resupply orders.
- As it is based on an online sales model, the manufacturer ships directly to the customer, using dropshipping.
- A stochastic linear programming framework is used to solve the problem instead of RL.

As concluding remarks of this article, it can be said that in a system with several customers whose demands are i.i.d., there is no need to hold any safety stock to meet the required SLs, which are different for each customer, as long as they are strictly less than one. This is not the case of the present master thesis, but it is considered as a reference in articles related to the type 2 SL.

In stochastic processes, Markov Chains are commonly used, as in article [22], where this tool is used to determine the profit derived from the sale of goods a company that provides spare parts for commercial cars will get.

Also in the field of stochastic models, there are other applications as it is the case of [23], whose main goal is the design of a Supply Chain Network (SCN) considering SL and demand-side and supply-side uncertainties simultaneously. Although the objective of this article is far from that of this master thesis, it is important to consider its Table 1, which shows different solution approaches in the literature for the design of a SCN and from which it can be concluded that RL has not been employed. Therefore, this reinforces the idea that this master thesis carries out an innovative and differential work.

To get a better understanding of the impact of SL in SC the article [24] can be consulted. Furthermore, many strategies for customer SL have been studied as is the case of [25], where the proposed strategies take advantage of the network itself and consider transshipping the material between locations. This cannot be applied to the present master thesis, since as it will be explained later, as they are clinical trials, once they are sent to the LDs they cannot be sent to other LDs because they can be modified or adulterated.

To conclude this subsection, it is important to remark that the resource pooling problem can also be seen as the stochastic knapsack problem (SKP), where the objective is to maximise the probability of reaching a target reward level without exceeding the knapsack capacity. This perspective is the one used in [26], where the properties of three classes of allocation policies: responsive (with perfect hindsight), adaptive (with information updates), and anticipative (with forecast information), are studied. Its objective is by using the type 1 SL, that is, an event-oriented performance criterion that measures the probability that all customer orders arriving within a given time interval will be completely delivered from stock on hand without delay to compare their performances for both capacity minimization and revenue maximization.

3.1.3 Different Applications of Reinforcement Learning Methods

RL is a way to teach machines certain behaviours through interactions with the environment, learning from mistakes and by using rewards.

As aforementioned, in Supervised Learning there are labelled data sets and the goal is to predict labels of unlabeled examples. However, data collection is expensive, because it requires infrastructure, support and monitoring. Consequently, over the past years, new applications of RL have been considerably developed.

One commonly known goal of RL is the creation of a mobile robot capable of moving around in the physical world [27], but there are simpler applications that despite the present computational progress, are still hard problems:

Board games: The most known examples are chess and backgammon [28], where RL can beat human world champions.

Video games and robot simulations: It studies the use of AI methods to perform at human-level [29].

Self-driving cars: It is a complex RL application that aims to teach the agent, in this case the car, to drive autonomously [30].

Nevertheless, there are less commonly known RL applications. Next some of them are presented.

The PageRank algorithm introduced by Lary Page in 1999 is used to rank web pages and also users in social media [31]. It is based on a Random Walk, which is a random process that describes a path made of several random steps and that can be represented as a Markov Chain, where nodes represent web pages. As a consequence, there are attempts to optimize it by the application of RL as it is the case of [32].

Other fields of application are Recommender Systems, Computer Systems, Energy Systems, Finance and Healthcare [33].

Regarding this last field, which is the healthcare sector, the main scope is to create individualized treatment depending on patients' different clinical characteristics and requirements. The general idea is to take action as the environment changes to assign optimal regime to patients [34]. However, while this master thesis is being developed, there are no RL application studies for optimization of the inventory management of clinical trials explicitly.

In relation to the application of RL in the field of SC inventory management, diverse models have been considered:

- In [35], the authors proposed a case-based RL algorithm for inventory management of a multi-agent SC structure.
- With respect to inventory management of short-lived items as fresh agri-products and dairy products, Q-Learning and SARSA algorithms are studied in [36].
- A case-based myopic RL algorithm of an inventory management problem of the SC with two-echelons is presented in [37].
- In [38] RL is applied to a case example of a global SC that spans several geographic areas and logistics stages considering one single item.

To conclude this section, several current applications of RL have been indicated, nonetheless, the inventory management of clinical trials has not been studied by RL, so that is what will be done in the present master thesis.

CHAPTER 4

Problem Formulation

In Chapter 2 the theoretical framework for MDPs, RL and their common way of problem formulation have been established.

It is not surprising that an MDP can be defined in many ways, so the objective of this chapter, among others, is to provide the MDP formulation for the current problem in order to apply the appropriate RL algorithms.

The first part of this chapter is devoted to the general problem description, in which the problem will be explained from a conceptual point of view.

In the second part, the problem will be described as a sequential decision-making problem, a necessary step to conclude formalizing it as an MDP. Furthermore, it will be explained why Julia Programming Language has been chosen for its development

4.1 GENERAL PROBLEM DESCRIPTION

Regarding clinical trials, usually the available supply of drugs is limited. On the one side, it should be taken into account that the supply of drugs of the clinical trial will be made as large as possible, and on the other, that the typical shelf life of products is short. As a consequence, short regular shipments are planned each time a new batch is produced.

As we are considering a CD from where drugs will be sent to the corresponding LDs, the main decision will be how much inventory to send to each LD and hence how much inventory will be left in the CD for potential intermediate resupplies if needed, that is to avoid a shortfall of drugs. Therefore, the size of the resupply shipment must also be decided.

Other considerations to take into account are:

- It is not on the scope of the problem how inventory is managed between LDs and investigation sites.
- The expected demand rate is not constant over time, so shipment sizes will be different in each period.
- Due to legislative requirements it is not possible to send doses back to the CD or between LDs, because they could be modified or adulterated.
- The cost of a shipment does not depend on the depot to which it is sent, so shipment cost will be independent of the LD of destination.
- Stock holding cost, lead times and lifetime of the drugs will be not considered.
- As we are considering the demand for drugs, we will use the β Service Level (Type 2), which is a quantity-oriented performance measure that describes the proportion of total demand in a reference period which is delivered without delay from stock on hand, that is, the expected proportion of demand that is satisfied.

In Figure 4.1, a conceptual description of the problem is shown. Note that demand is assumed to follow a discrete uniform distribution.

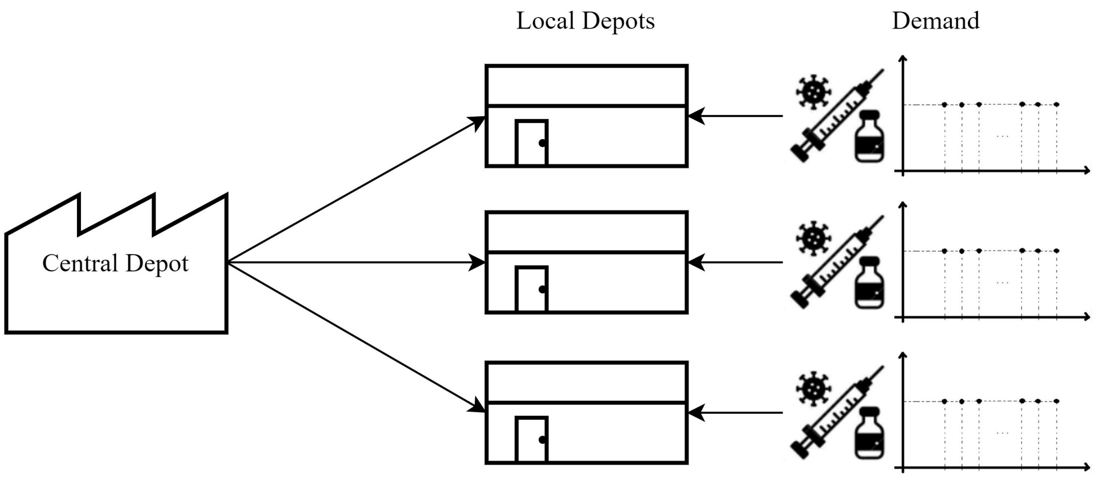

Figure 4.1: Conceptual description of the problem. Source: Own elaboration.

Furthermore, to obtain a better understanding of the problem, some parameters have to be defined:

Period: Is the elapsed time between the availability of two successive product batches, assuming that periods are of identical length.

CD resupply: Is a shipment happening when a new batch of products becomes available.

LD resupply: Are the doses shipped between two CD resupplies.

We will consider the *CD resupply* as fixed and beyond the scope of the problem and we will try to minimise only the *LD resupply*, avoiding missing doses before the next *CD resupply*, when the CD will be full again.

Finally, we can conclude that the objective of the problem will be to minimise the *LD resupply*, and therefore the derived costs while ensuring a very low number of doses that cannot be supplied from the LDs.

4.2 PROBLEM DESCRIPTION AS AN MDP

As aforementioned in Chapter 2, MDPs are a formalization of sequential decision-making problems. In addition, most of the RL methods need the problem to be described as a finite MDP.

More specifically, an MDP is defined as a five-tuple $M = \langle S, A, P, R, \gamma \rangle$, where S is the set of states, A is the set of actions, P is the state-transition function, R is the reward function and $\gamma \in [0, 1]$ is the discount factor.

So in the present problem, this five-tuple can be defined as:

- The state $s \in S$ is the inventory left in the CD and each LD, considering demand, the previous quantity that had already been sent and the number of periods remaining until the next CD resupply.
- The action $a \in A$ can be generalized as sending a certain quantity of doses from the CD to the LD. The system decides the quantity to be sent from the CD to each LD based on the quantity of doses that is left on each LD and in the CD, which depends on the demand of each LD.
- $p \in P$ gives for each state and action the probability distribution over states. That means the probability of ending in state $s' \in S$, given that the agent starts in state $s \in S$ and takes action $a \in A$.
- The reward function $r(s, a)$ provides the reward or the punishment when taking action $a \in A$ at state $s \in S$. As it has already been indicated, in this master thesis the objective is to reduce the number of shipments for satisfying demand, and so minimise the derived cost. The reward function will have a punishment approach using a fixed cost per shipment and a penalty term for the non-served doses.
- $\gamma \in [0, 1]$ is the discount factor, which will indicate how much the agent cares about late rewards relative to near time rewards.

As it will be later explained, when programming, $s \in S$ and $a \in A$ will be modelled as vectors.

It is important to consider that on the one side, the environment will be deterministic as periods of time t and temporal horizon T cannot be changed, but on the other side, it will be stochastic as demand rate won't be constant over time, thus it is not unique and cannot be completely determined by the agent.

4.2.1 Why Julia Programming Language

Nowadays in the ML environment, Python is known as the number one programming language followed by MATLAB and C/C++. It is trivial that in order to accelerate ML applications, a good balance between both efficiency and simplicity programming language is required, so that's why Julia is becoming more popular among ML specialists [39].

The main reasons for this are:

- Python and MATLAB programming languages are commonly used to solve large-scale tasks but their performance is slow. They also have very strict requirements for memory and computing power. Additionally, MATLAB is a commercial software [40].
- C/C++ is known for its high efficiency but it is difficult to use.
- Regarding Julia:
  - It is easy to use as Python and MATLAB.
  - It is efficient as C/C++, and as it was designed with a focus on numerical and scientific computation for ML, Julia is highly recommended for tasks involving intensive data processing.
  - It is an open-source language whose syntax is similar to MATLAB.
  - There are several ML algorithms written in Julia, and therefore, several applications, as those mentioned in [39].

Although Julia is a new programming language with less community and tooling support, it has many advantages over the other existing programming languages. In the next subsection we will analyse the Julia interface *POMDPs.jl*, which will be used for solving the current problem.

4.2.1.1 POMDPs.jl Interface

MDPs are the foundation upon which RL is built, so knowledge of MDPs is a prerequisite to understand Partially Observable Markov Decision Processs (POMDPs).

The *POMDPs.jl* interface makes possible working with both MDPs and POMDPs, because it allows users to:

- Represent problems as MDPs and POMDPs.
- Solving problems.
- Running simulations.

In order to trace the roots of *POMDPs.jl* it it necessary to go back to the early days of Julia in 2012, when Mykel Kochenderfer was working at MIT's Lincoln Laboratory. Nowadays he is an Associate Professor at Stanford University and the director of the Stanford Intelligent Systems Laboratory (SISL). Mykel Kochenderfer and Jeff Bezanson (Julia co-founder) developed an aircraft collision avoidance system for commercial airplanes using an early version of Julia.

In 2017 the Stanford Intelligent Systems Laboratory team led by Zachary Sunberg published the paper called "*POMDPs.jl: A Framework for Sequential Decision Making under Uncertainty*" [41]. *POMDPs.jl* is not an isolated Julia package, it's an interface which works with an expanding collection of several complementary packages that are under active development by the Julia Community.

The *POMDPs.jl* eco-system can be used for many different applications, but in this master thesis we are going to make use of it to optimize the inventory management of clinical trials. Later its implementation will be explained in more detail.

To summarise, in this chapter the problem has been analysed in two ways. Firstly, it was conceptually described and secondly it was characterized as an MDP and also explained why the Julia Programming Language was chosen for the computations. Now, we continue with the problem resolution, by first "building" the problem and secondly analysing the obtained results.

CHAPTER 5

Problem Architecture

Before going deeply into the development of the project, we consider necessary to explain the used algorithms. To properly do that, first, in this chapter a set of definitions and essential knowledge regarding the techniques, algorithms and resources used are given.

Then, we will apply those concepts for optimizing the inventory management of clinical trials.

5.1 PRELIMINARY CONCEPTS

First, it is necessary to describe the following concepts:

5.1.1 Episodic and Continuous Tasks

In a RL problem, there are two different types of tasks that can be defined as follows:

Episodic Tasks: They last a finite amount of time, which is denoted as an episode.

Continuous Tasks: They never end.

Based on the previous definition, the current problem is an episodic task delimited by the availability of two successive product batches, assuming periods of identical length.

5.1.2 Model-Free and Model-Based

Regarding RL, there are two different computational strategies, model-based learning and model-free learning. The main difference between them is whether the agent learns or acts.

In model-based learning the goal is to create a model of the environment in order to maximise the reward, which is getting the distributions of the next state S_{t+1} and the reward R_{t+1} for knowing the optimal actions.

On the other side, model-free learning does not use neither of the predictions of the next state S_{t+1} nor of the reward R_{t+1}. As a consequence, this computational strategy uses samples from the environment.

It is important to consider that, just because there is a model of the environment, the agent has to be model-based. In fact, in the present problem, there is a model of the environment but it is not necessary to learn it. Instead, the agent can learn a policy using algorithms that sample from experiences like Q-Learning and SARSA. As a consequence, it is better to evaluate $Q(s, a)$ rather than $V(s)$. This is because as we are doing model-free RL, we do not know how states transition into new states.

In the next section, we will see explicitly the difference between model-based or model-free algorithms by looking at the algorithms formulas and see whether they use the transition function.

5.2 ALGORITHMS

Algorithms are the central part of this master thesis, therefore, most of the resources and time were devoted to them.

From the beginning, it was decided to face the problem with an innovative perspective, which is through the application of RL. We apply the two most commonly used RL algorithms: Q-Learning and SARSA.

5.2.1 Temporal Difference Learning

Temporal-Difference (TD) methods are widely used in RL, they are a combination of the Dynamic Programming (DP) methods, which are model-based, and the Monte Carlo (MC) methods, which are model-free.

MC learning is the simplest approach to model-free learning, as it is purely based on trial and error. It estimates value functions and finds optimal policies based on experience. Furthermore, this is an episodic algorithm, because it runs a complete episode in order to update the value of a state. If we take a look at Equation 5.1, we notice that only at the end of the episode, when the discounted return G_t is known, $V(S_t)$ is updated.

$$V\left(S_t\right) \leftarrow V\left(S_t\right)+\alpha\left[G_t-V\left(S_t\right)\right] \tag{5.1}$$

On the other side, TD learning, only waits until the next time step to do the update. Looking at Equation 5.2, we notice that in order to update $V(S_t)$, it uses the observed reward R_{t+1} and the estimate $V(S_{t+1})$.

$$V\left(S_t\right) \leftarrow V\left(S_t\right)+\alpha\left[R_{t+1}+\gamma V\left(S_{t+1}\right)-V\left(S_t\right)\right] \tag{5.2}$$

So, TD exploits the Markov property, which tells us that the evolution of the MDP only depends on the present state and not on the previous states.

In TD, similarly to MC, the agent learns from sampled experience. This means that no model of the environment's dynamics is needed. Like in DP, TD methods bootstrap, so contrary to MC, it is not necessary to wait for a final outcome, just until the next time step. Therefore, this method can learn from incomplete episodes. Consequently, TD updates are based on another estimate.

Q-Learning and SARSA are TD algorithms. They are explained in more detail down below.

5.2.1.1 Q-Learning

Q-Learning [42] is one of the most popular model-free RL algorithms. That means that this algorithm estimates the optimal policy without estimating the transition and reward functions.

It should be considered that the Q in Q-Learning stands for quality, with which the model finds its next action by improving its own quality. We should also consider that Q-Learning is an off-policy method, which means that it assumes following a greedy policy (despite the fact that it's not following a greedy policy) in order to estimate the total discounted future reward.

Q-Learning and SARSA use a table known as Q-Table, with a row for each state and a column for each action, where they calculate the maximum expected future rewards for the (s, a) pair. So this table gives the agent the best action at each state.

Q-Learning uses the Bellman Optimality Equation to update the Q-Values for each (s, a) pair. This procedure is called Value-Iteration and is done until the Q-Function converges to its optimal.

We have said that it is a model-free algorithm. In fact, if we take a look at Equation 5.3, we can clearly see that it does not use any probabilities of the MDP. And as it was previously stated in Chapter 2, R_{t+1} is the obtained reward after the agent takes the action, and it is not necessarily known in advance.

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha\left[R_{t+1} + \gamma \max_a Q\left(S_{t+1}, a\right) - Q\left(S_t, A_t\right)\right] \tag{5.3}$$

5.2.1.2 SARSA

The name SARSA [43] refers to *State-Action-Reward-State-Action*, because it uses each element of the quintuple (S_t, A_t, R_{t+1}, S_{t+1}, A_{t+1}).

This algorithm is quite similar to Q-Learning, but the main difference between them is that SARSA is an on-policy algorithm. It means that it only needs one policy, and so it estimates the value of the policy that is being followed.

$$Q\left(S_t, A_t\right) \leftarrow Q\left(S_t, A_t\right) + \alpha\left[R_{t+1} + \gamma Q\left(S_{t+1}, A_{t+1}\right) - Q\left(S_t, A_t\right)\right] \tag{5.4}$$

By looking at Equation 5.4, we can notice that two actions are selected, A_t and A_{t+1}. These actions follow the current policy.

5.2.1.3 Differences Between Q-Learning and SARSA

Both SARSA and Q-Learning use the Bellman Optimality Equation, but if we compare Equations 5.3 and 5.4, we observe that in Q-Learning the only restriction over the next action is that it has to maximise the Q-Value for the next state. Q-Learning assumes that after the next step, the agent takes the best action forever.

The updating rule of the Q-Value is what differs SARSA from Q-Learning. In SARSA it is necessary to know the next action of the policy in order to perform an update step, while Q-Learning assumes using the optimal policy. Q-Learning can be understood as a SARSA max method.

In Chapter 2, we have seen the exploration-exploitation trade-off and ε-greedy methods, so we know about the importance of balancing exploration and exploitation. Both SARSA and Q-Learning follow a ε-greedy policy, which guarantees the exploration. The main difference is:

- SARSA is an on-policy method because the new action A_{t+1} is chosen using the same ε-greedy policy as the action A_t, the one that led the agent to state S_{t+1}.
- Q-Learning is an off-policy method because it chooses the new action greedily but not following the current policy.

To understand this more precisely, let's take a look at Figure 5.1:

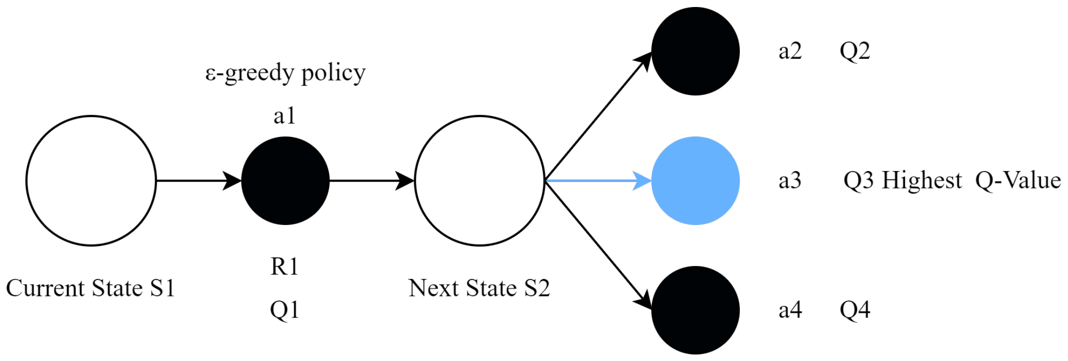

Figure 5.1: Q-Learning algorithm. Source: Own elaboration.

From the current state $S1$, the agent follows an ε-greedy policy to select the current action $a1$. Then, the agent gets a reward $R1$ and transitions to state $S2$. Now, it is time to update the estimated Q-Value $Q1$ for the current state and selected action by using the Q-Value from the next state. This is where the difference between Q-Learning and SARSA occurs.

In the next state, the agent can take several actions. With Q-Learning it uses the action $a3$ which has the highest Q-Value $Q3$. Action $a3$ is not necessarily the action that the agent will finally end up taking when it reaches the next time step. In SARSA, the next action is chosen using the same ε-greedy policy as the previous action.

In consequence, as SARSA is an on-policy method, it will learn the optimal ε-greedy policy, which means that the Q-Value function will converge to an optimal Q-Value function but with respect to the ε-greedy policy. Following [1], as long as all state–action pairs are visited an infinite number of times, we expect that as ε goes to 0 ($\varepsilon = 1/t$) , SARSA will converge to the overall optimal policy.

SARSA can be understood as a "conservative algorithm", while Q-Learning as an "aggressive algorithm". One clear illustration of this is the *Cliff Walking Example* from [1]. As a consequence, if fast results are needed, Q-Learning should be chosen. However, if mistakes are costly, then SARSA is the best option.

Finally, remark that both SARSA and Q-Learning are value-based algorithms, which means that they estimate the policy by first estimating the associated value function.

Once the theoretical foundations of Q-Learning and SARSA methods have been established, we proceed to build the necessary system for the current problem resolution.

5.3 BUILDING THE SYSTEM

Previously, we have defined the principles of RL and also indicated the importance of the MDPs formulation in the RL frame. Furthermore, in Chapter 4, the current problem has been described as an MDP. From these previous theoretical concepts, the goal of this section is to put all of them into practice.

Before doing the corresponding implementation, first, an approach to the current MDP is made. Second, the differences between training and testing will be explained.

5.3.1 MDP Approach

As aforementioned, it is necessary to express the problem as an MDP so as to apply the RL methods. Starting from the previous simplified characterization as an MDP, now, the objective is to explain it in more detail in order to solve this problem.

5.3.1.1 State-Space Definition

The State-Space S is the set of all the possible states in the current problem. As it was pointed out, each possible state is represented as a vector containing the following information:

- Inventory left in the CD.
- Inventory left in each LD.
- The number of periods remaining until the next CD resupply.

To better understand this, let's consider a state as *State(16, [5, 3, 4], 1)*, where the first term 16 indicates the inventory left in the CD, the three second terms ([5, 3, 4]) indicate the inventory left in each LD and the last term 1, the number of periods remaining until the next CD resupply.

Consequently, we can state that:

- Inventory in each LD cannot be more than the capacity of the LD, nor negative. The latter means unsatisfied demand, which will be represented with a value of zero stock and each missing dose will be multiplied by the penalty cost.

- The sum of the inventory left at the LDs cannot be greater than the difference between the maximum CD inventory and its remaining inventory.

  For example, considering the initial moment, when a CD resupply occurs, the maximum CD inventory is 20, LDs are empty and there are 10 remaining periods until the next CD resupply. This means *State(20, [0, 0, 0], 10).* In the next period there will be 9 remaining periods and the inventory left at the CD has to be 20 minus the number of doses sent to the LDs. If we send 3 to the first one, 1 to the second and 2 to the third one, and we assume that there is no demand during that period, the next state will be: *State(14, [3, 1, 2], 9).*

- It is trivial that we cannot send more than the inventory left at the CD. So, if we are in *State(14, [3, 1, 2], 9)*, the sum of all the doses sent to the LDs cannot be greater than 14 .

5.3.1.2 Action-Space Definition

Similar to the State-Space S, the Action-Space A is the set of all the possible actions. Each possible action is represented as a vector indicating the number of doses sent to each LD.

Following a similar procedure as we did with the State-Space, let's consider an action as *Action([3, 1, 2])*, which indicates sending 3 doses to the first LD, 1 to the second LD and 2 to the third one.

Notice that similarly to the State-Space, this vector will get bigger if we have more LDs or smaller if we have less. This means *Action([...Number of LDs]).*

5.3.1.3 Demand Definition

Uncertain situations are those that cannot be predicted with certainty. However, probability is a measure of how likely it is that a situation will occur.

In our case, the expected demand rate is not constant over time, so shipment sizes will be different in each period for each LD.

We will consider equiprobability, which is a discrete uniform distribution where each possible demand configuration will have the same probability of occurring. So, if there are n possible demand configurations, each of them will have a probability of occurring equal to $1/n$.

These demand configurations, similarly to states and actions, are represented as vectors. Each term of the demand vector, will denote the demand of doses in the corresponding LD.

So, if we consider the demand *Demand([-2, -1, -1])*, which indicates having a demand of 2 doses in the first LD and a demand of 1 dose in the second and third LD, and we also consider the previous examples from the State-Space S and Action-Space A definition, we have the final state *State(14, [1, 0, 1], 9).* This is represented in Figure 5.2.

Finally, remark that the previous example was just for giving an approach to the problem. Later, several configurations will be considered.

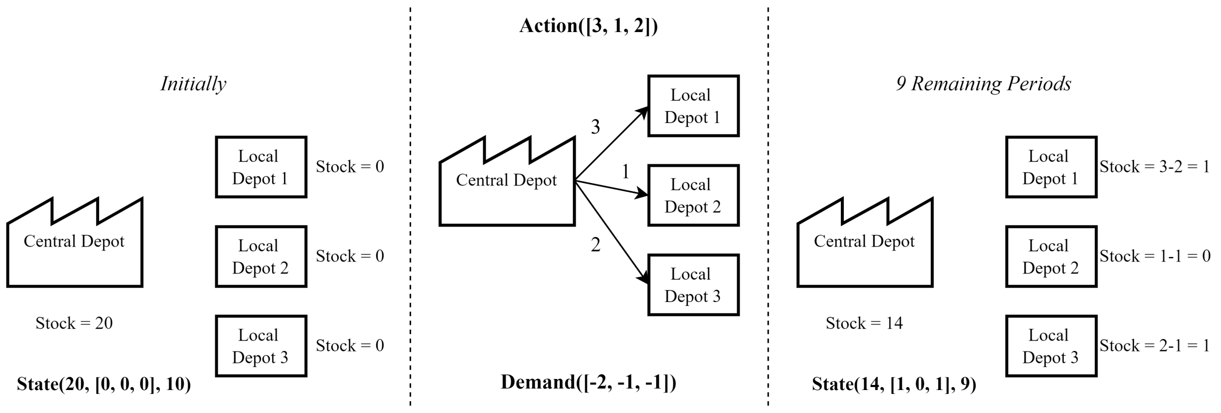

Figure 5.2: State transition given an action and demand. Source: Own elaboration.

5.3.1.4 Transition and Reward Functions Definition

The transition function defines the probability of transitioning to each state, given the current state and action, while the reward function establishes the recompense received, which means that it maps every possible transition (s, a, s') to a real reward value.

When working with large MDPs, as it is the current case, it becomes quite arduous to define the transition and reward functions. In this situation it is preferable to work with a model that returns samples of the next states. This is what is done in this master thesis.

POMPDs.jl provides two ways of defining an MDP:

Explicit interface: Where the transition probabilities are explicitly defined.

Implicit interface: Also known as the generative interface, it uses functions to return states instead of distributions, as in the explicit definition.

Thus, as we will see later, we will work with large state and action spaces, so we will use the implicit interface, which we will denote as the generative interface.

To conclude this section, let's consider Figure 5.2 and Figure 5.3, in which the initial state is represented in green, actions in yellow, states in blue and the terminal state in purple. The purpose of this figure is to illustrate the following procedure.

Initially, when a CD resupply occurs, the agent will be at the initial state, from where it can take several actions. These actions and the corresponding demand will take it to the next state, from where it will have to take another action, and so on until it reaches the terminal state.

Both figures show the current problem, but they do it with a different perspective. Figure 5.2 gives a more cross-sectional view, while Figure 5.3 shows the agent's perspective when facing the problem.

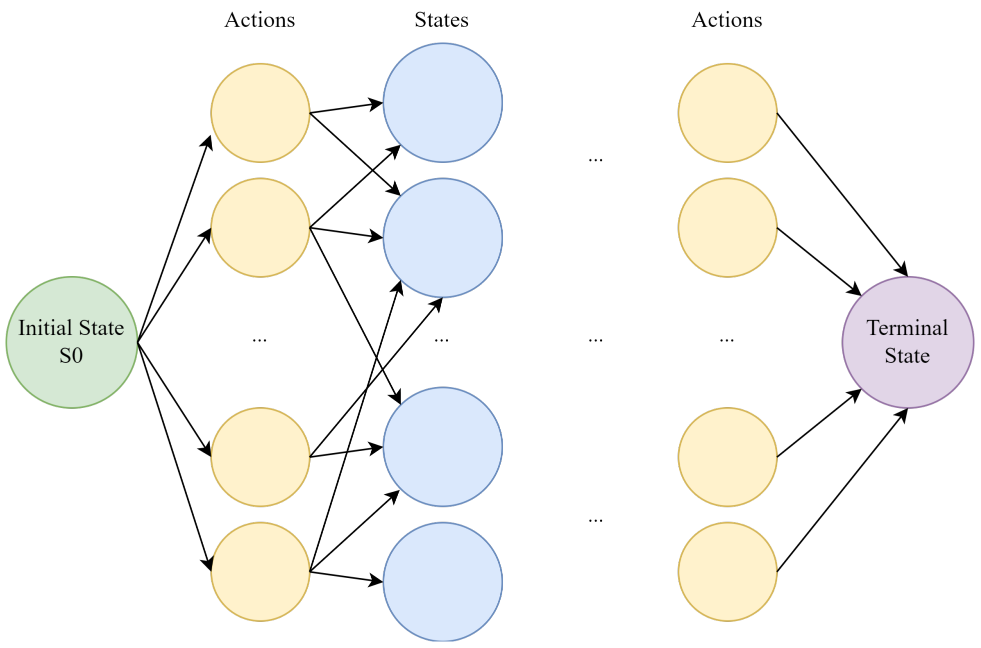

Figure 5.3: State transition diagram from agent's perspective. Source: Own elaboration.

5.3.2 Training and Testing

In most common real-life cases, when we want to do something challenging, like lifting heavy weights or playing a musical instrument, there is an initial phase when we train, and then, a second one, when we test our acquired skills.

In RL something very similar happens. There is a first phase based on a system of rewards and penalties when the agent is trained and a second phase when the trained agent is put into the situation for which it is trained.

The agent's goal is to find the policy, that maximises the expected return, which is called the *optimal policy*. In order to estimate this optimal policy, optimization algorithms as the ones previously mentioned, are applied. The use of these RL algorithms is called *training*.

During the training phase the agent might not be able to find the optimal policy. Therefore, with the aim of being sure whether the learned policy is good enough, it should be evaluated. This is what is known as *testing*.

The testing phase is the estimation of the quality of the learned policy and consequently, how much reward the agent would obtain if it followed that policy.

In this master thesis, as in most other investigation projects, the metric that will be used to assess the quality of the policy is to plot the discounted return G_t, described in Chapter 2 by the Equation 2.1, considering that an algorithm dominates another if its graphic is mostly below the other, as we are considering penalties instead of positive rewards.

To end this subsection, we can conclude that during training, the goal is to find the policy and during testing, to evaluate the quality of the learned policy. Once explained the differences between training and learning, we continue with the corresponding implementation.

5.4 IMPLEMENTATION

Once the problem has been expressed as an MDP, the objective is to translate this approach into code in order to apply the RL algorithms. To do that, first, we will define the problem data and the used hyper-parameters.

Then, the previous mentioned five-tuple $M = \langle S, A, P, R, \gamma \rangle$ will be represented using the Julia Programming Language and especially the interface *POMDPs.jl*.

5.4.1 Problem Data and Hyper-Parameters

Problem data are any characteristic or attribute used to define or classify any system, problem, or situation. It means, that they are helpful for describing, identifying and evaluating.

On the other side, hyper-parameters are the values of the configurations used during the training process. The optimal value of a hyper-parameter cannot be known beforehand, so it is necessary to search for the best option through trial and error.

In the current master thesis, we will first define the problem data and the used hyper-parameters. Then, the results will be evaluated based on the different problem configurations defined by the problem data and the hyper-parameters. We start by defining the problem data.

5.4.1.1 Problem Data

Different problem configurations will be analysed, so as to gain a wider perspective of the current problem. To do that, problem data will have different values. These are as follows:

Number of central depots: Only one CD will be considered. From this depot, all doses will be shipped to the corresponding LDs.

Number of local depots: We will consider two and three LDs.

Central depot stock: It is the storage capacity of the CD. To make the problem more similar to reality, this capacity will be defined as a percentage of the maximum total demand.

Local depots stock: Similar to CD stock, but regarding LDs, it is the maximum storage capacity of each LD.

CD capacity percentage: As its name indicates, this value is a percentage, which takes values from 0 to 1. It will be used to define the central depot stock with respect to the maximum total demand. Different values will be analysed.

Number of periods: They refer to the elapsed time between the availability of two successive product batches. We will analyse different possible number of periods. All of this, assuming periods of identical length.

Demand: The demand rate won't be constant over time. As a consequence, shipment sizes will be different in each period for each LD. Discrete uniform distribution will be used for that purpose.

Fixed cost per shipment: As the objective of this master thesis is to minimise the number of shipments, a fixed cost of 1 per shipment will be considered.

Penalty cost for unsatisfied demand: Regarding clinical trials, it is important to maintain an adequate service level. This means satisfying the highest possible demand. To satisfactorily do that, a penalty cost for each non-served dose of -10 is established.

Penalty cost for illegal actions: As later it will be explained more in detail, there will be certain states from which it will be illegal to take some actions, that is, state-dependent actions. In order to present this to the agent, a penalty cost of -100 is established.

Before doing the problem evaluation, several values for the costs were tested. From these tests it has been concluded that these values have not a great impact on the problem, so these values are fixed. In each section, the corresponding problem data will be indicated.

5.4.1.2 Hyper-Parameters

Hyper-parameters are tunable and have an important impact on the behaviour of the model, that is, they can directly affect how well the model behaves. Thus, it is necessary to define the following hyper-parameters:

Discount factor γ: Varying its values from 0 to 1, we set the preference for sooner rewards rather than later rewards. Several values within this interval have been tested. From these tests it has been concluded that the discount factor $\gamma = 0.9$ gives the best results, so that's why this value is considered throughout this analysis. In order to see two examples of these tests, consult Appendix B.1.

Number of episodes: Number of episodes used for training.

Episode length: Maximum number of steps before the episode is ended if the goal is not reached. The number of periods between two CD resupplies will be considered.

Learning rate α: Is the velocity at which the model learns. It controls the amount of error with which the model model is updated.

Exploration policy: Policy used by the algorithm. ε-greedy policies will be used.

ε: Probability with which the ε-greedy policy selects exploration rather than exploitation.

Number of episodes to evaluate the policy: After learning the policy, we will set a number of episodes to do its testing, in order to estimate how much total reward to expect from that policy. Although several values for this hyper-parameter have been evaluated, it has been concluded that it does not have a great impact on the results, therefore the value that comes with the package by default will be used, that is, 20 episodes.

In each section, the corresponding hyper-parameters will be indicated.

5.4.2 Environment Setting in Julia Language

States, actions and demands are defined as vectors, which contain all the necessary information. In the case of state vectors, they show the stock in the CD, in the LDs and the remaining periods until the next CD resupply. Action vectors indicate the doses to be sent to each LD and demand vectors, the consumption of doses in each LD.

In the next subsections we explain how these vectors and the generative interface are implemented in Julia. It should be noted that an exhaustive explanation of the code is not made here. This is presented in the corresponding code file, which is found in the Appendix C.

5.4.2.1 State-Space

Structs are the most commonly used type in Julia because they allow to define other custom types, similar to classes in an object-oriented programming language. A struct has a name and a set of fields, which can have type restrictions, as it is our case.

We create the struct *State* to define states. It is shown below:

```julia
struct State
    cds::Int #Stock in the CD
    lds::SVector{N_lds, Int} #Stock in the LDs
    nper::Int #Number of periods
end
```

The fields of the struct *State* are defined as follows:

`cds` - Central Depot Stock: Integer that represent the remaining stock in the CD.

`lds` - Local Depots Stock: Vector whose dimension is equal to the number of LDs and it is formed by integers that represent the remaining stock in the LDs.

`nper` - Number of Periods: Integer that represent the remaining periods until the next CD resupply.

Once created the struct *State*, we set up the following states:

`terminal`: States in which the remaining periods until the next CD resupply are equal to zero.

`non_terminal`: States in which there are one or more remaining periods until the next CD resupply.

`init_resupply`: State in which a CD resupply occurs.

`spt`: Dummy state for illegal actions, which will be defined down below.

By concatenating `terminal` and `spt`, this latter is added to the set of terminal states, and by considering these four types of states, we create the whole State-Space, denoted as `SS`. So, terminal states are part of the State-Space `SS`.

5.4.2.2 Action-Space

Similarly as we did with the State-Space, we use a struct called *Action* to define actions. This struct is as follows:

```
struct Action
    ship::SVector{N_lds, Int} #Shipping quantity to the LDs
end
```

In this case, the struct *Action* has only one field:

> **ship - Shipping Quantity to the Local Depots:** Vector whose dimension is equal to the number of LDs and it is formed by integers that represent the doses to be sent to the LDs.

Hence, the Action-Space, which is denoted as `AS`, will be formed by all the possible combinations for actions. Nevertheless, there would be illegal actions, those actions that may not be available at specific states, like for example sending more doses than the remaining stock at the CD or overloading the LDs. In this case, three possible approaches to handle them are proposed:

- Give a bad reward and do not let the environment evolve by moving to a terminal dummy state `spt`.
- As the previous approach, give a bad reward and do not let the environment evolve by remaining in the same state `s`.
- Do not let the agent take illegal actions by using a function that given a state `s`, returns all the possible actions `a` that can be taken from that state.

This third approach seems the most efficient. Nevertheless, due to the limitations imposed by the *POMDPs.jl* package, it is not possible to implement it. Therefore the first and second approaches will be used.

Despite the fact that this package does not allow to use the third approach, it allows to modify the Q-Table of the algorithms. Remember that this table has a row for each state and a column for each action, where it calculates the maximum expected future rewards for the (s, a) pair. So in order to prevent the agent from taking illegal actions, the Q-Table will be pre-populated with large negative values for all impossible combinations. By doing it, the agent does not need to query entire the environment and it can use the Q-Table as-is. As later it will be demonstrated, this will speed-up computations.

To do that, first, the following function which given a state, returns all possible actions will be defined:

```
function A(s::State)
    return  [(Action(ship)) for ship in Iterators.product(ntuple((i ->
    ↪  0:LD_st.-s.lds[i]),N_lds)...) if sum(ship)<=s.cds]
end
```

Next, all the values of the table will be set to zero and then, by calling the previously indicated function `A`, large negative values for all impossible combinations will be set:

```
solver.Q_vals = zeros(length(states(CT_MDP)), length(actions(CT_MDP)))

for s in states(CT_MDP)
    valid_actions = A(s)
    for a in actions(CT_MDP)
        if !(a in valid_actions)
            solver.Q_vals[stateindex(CT_MDP, s), actionindex(CT_MDP, a)] = -9999
        end
    end
end
```

5.4.2.3 Demand

Regarding demand, it is not necessary to create a struct. As we are considering a discrete uniform distribution, we create as many possible configurations until the maximum possible demand for all the LDs. This is done as follows:

```
demand_vect = [((dem)) for dem in [Iterators.product(ntuple(_ -> -dem:0,
↪   N_lds)...)...]]
```

The result, `demand_vect`, is a vector containing as many combinations as possible.

5.4.2.4 Generative Interface with QuickPOMDPs.jl

The quick interface *QuickPOMDPs.jl* will be used to define the MDP.

In order to do it properly, the first argument to the QuickMDP constructor has to be a function which defines the generative model of the MDP. This function, returns a tuple which contains the next state `sp` and the reward `r`, given the current state `s` and action `a`. It is important to keep in mind that the reward `r` will be negative as we are considering costs and penalties, so the agent's goal will be to minimise it.

The steps performed in this function are the following:

1. From a given state `s` and action `a`, it computes `nps`, a fictitious next possible state without considering the demand, in order to check if this next state is feasible. That means, if there is enough stock left in the CD and if the LDs would be overloaded. To do that, it subtracts the sum of the doses to be sent from the CD, it adds them to the corresponding LDs and it reduces by one the remaining periods until the next CD resupply.
2. If the result of this feasibility test is positive, that is, `nps` is within the State-Space `SS`, it considers the demand by subtracting it to the LDs stock and:
    (a) If all the demand is satisfied, it computes the reward `r` only considering the shipment cost if something is sent and it returns the next state `sp`.
    (b) If there is unsatisfied demand, which means that some patients did not receive their doses, it computes the reward `r` as the penalty cost plus the shipment cost, in case something is sent. By doing this, we will consider that missing doses are handled by an external system. Then it sets LD stock to zero in those where there is unsatisfied demand and returns the next state `sp`.

3. If the result of this feasibility test is negative, it gets a bad reward `r` called `penalty` and it sends the agent instantly back to the dummy state `spt` or remains in the same state `s`, depending on the approach followed, which has been defined in Subsection 5.4.2.2.

The next arguments to define the MDP are:

State-Space `SS`: Set of all the possible states.

Action-Space `AS`: Set of all the possible actions.

Initial State `init_resupply`: Initial state of the MDP.

Discount Factor `gamma`: γ wich takes values between 0 and 1.

Terminal State: Terminal states of the MDP.

As a final remark, as we are considering demand with a discrete uniform distribution where each possible demand configuration will have the same probability of occurring, this demand will be chosen randomly.

CHAPTER 6

Experimental Results

In this chapter we analyse the results of the simulations carried out to solve the problem, considering different configurations and values for the previously indicated approaches, problem data and hyper-parameters. To do that, a *Seed*, which is the base value for random number generation, was used.

Regarding the approaches defined in Subsection 5.4.2.2, all the following tests were done following the first two approaches and the corresponding modification of the Q-Table, which will be explained more in detail in following sections. But, as the results were practically the same for both approaches and in order not to saturate the current analysis with too much duplicate information, the initial analysis between the Q-Learning and SARSA algorithms will be shown following the first approach without modifying the Q-Table, that is, using a *Dummy State* `spt` for illegal actions. Subsequently, both approaches and the modification of the Q-Table are considered.

First, we will evaluate the case of 1 CD and 2 LDs, and then 1 CD and 3 LDs.

This evaluation was performed with a laptop MSI Modern 14, Intel Core i7-10510U, 2.30 GHz processor and RAM memory of 16 GB.

6.1 2 LOCAL DEPOTS

To begin this analysis, we will consider 2 LDs and a capacity of serving the 100% of the total maximum demand of the LDs during the whole time horizon. Then, more realistic cases where capacity is 90%, 80%, 70% and 60% will be considered. This is done to get a more lifelike point of view of the problem.

6.1.1 2 LDs with 100% capacity at the CD

Let's start with a simplified version of the problem, where there will be full capacity of the CD to supply all the demand of the LDs.

6.1.1.1 2 LDs with 100% capacity at the CD and 3 periods

There are 2 LDs with sufficient storage capacity for the maximum demand of the entire time horizon between 2 CD resupplies, so our intuition is that the optimal policy will be to fill the LDs to their maximum storage capacity in order to avoid LDs resupply orders.

In this case only 3 periods are considered. Later on we will evaluate a more realistic case, where 10 periods are considered. We follow the first approach defined in Subsection 5.4.2.2.

Initially, we will apply Q-Learning and SARSA algorithms for the problem data shown in Table 6.1, where we can also see the size of the resulting State-Space `SS` and Action-Space `AS`. In order to check the differences between both algorithms, we start considering the values for the hyper-parameters shown in Table 6.2. Plots are shown in Figure 6.1.

Problem data	**Value**
CD capacity percentage	100%
Central depot stock	24
Local depots stock	12
Number of periods	3
Maximum possible demand	4
State-Space	6,593 states
Action-Space	169 actions

Table 6.1: Problem Data for 2 LDs with 100% capacity at the CD and 3 periods.

Algorithm	**Learning Rate α**	ε	**Computation Time (sec)**	**Convergence (episodes)**
Q-Learning	0.10	0.01	59.99	200,000
SARSA	0.10	0.01	64.21	-

Table 6.2: Hyper-Parameters for 2 LDs with 100% capacity at the CD and 3 periods.

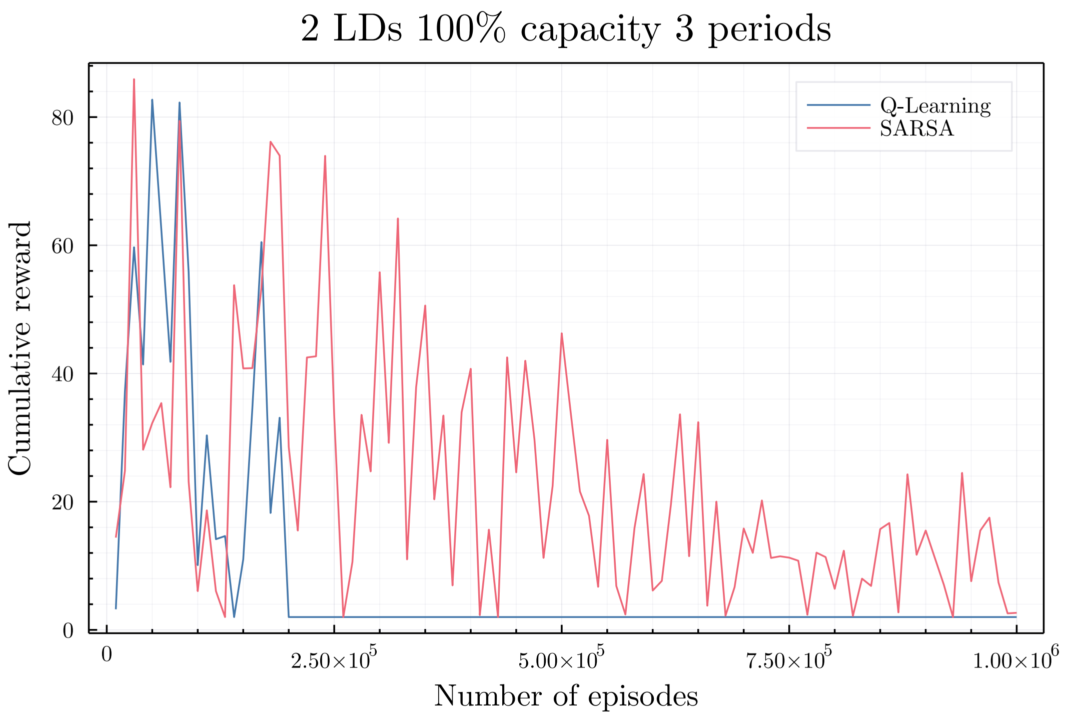

Figure 6.1: Initial plot of Q-Learning and SARSA algorithms for 2 LDs with 100% capacity at the CD and 3 periods. Source: Own elaboration.

Q-Learning is faster than SARSA. It converges after 200,000 episodes, while SARSA does not converge. As it was previously indicated, SARSA is expected to converge to the overall optimal policy as long as all state–action pairs are visited an infinite number of times. So, we should take care of the exploration-exploitation trade-off and use a value of ε ($\varepsilon = 1/t$).

In Figure 6.2, we see the SARSA algorithm for a value of ε ($\varepsilon = 1/t$). That means that at the beginning it explores 100% of the times and it ends up exploiting. It is also considered the impact of one of the most important hyper-parameters, the learning rate α, which indicates the rate of change in the model. By setting a learning rate α that is too high, there might be fluctuations and the model's performance will be oscillating. On the other side, a too small learning rate α might get stuck on a non-optimal solution. We follow the common recommendation and start with a value of 0.01000 and then try exponentially lower values, 0.00100, 0.00010 and 0.00001. The values of 0.10000, 0.01000 and 0.000001 were also tested but results got worse, so they are not plotted.

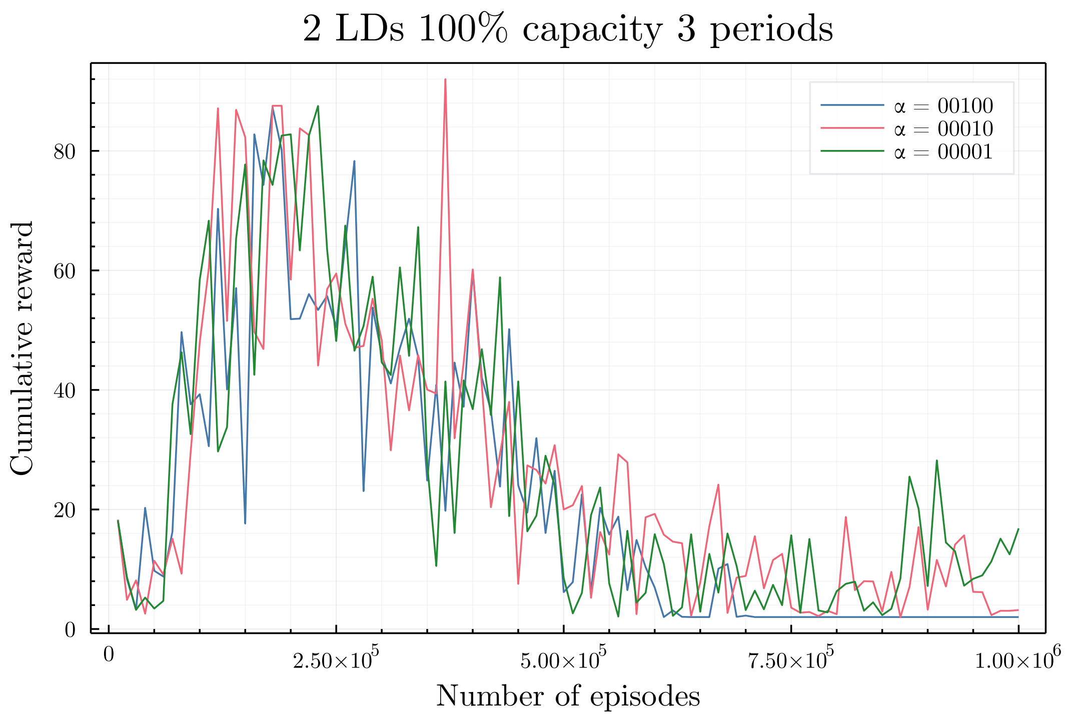

Figure 6.2: SARSA algorithm for 2 LDs with 100% capacity at the CD and 3 periods for different values of α and $\varepsilon = 1/t$. Source: Own elaboration.

In Table 6.3 the computation times and number of episodes required to converge per each α value are shown:

Learning Rate α	**Computation Time (sec)**	**Convergence (episodes)**
0.00100	43.14	710,000
0.00010	43.43	-
0.00001	54.77	-

Table 6.3: Computation times and convergence episodes per each α with SARSA algorithm.

With SARSA, convergence occurs after 710,000 episodes with a learning rate α of 0.00100 and ε decaying ($\varepsilon = 1/t$). Furthermore, in Table 6.3 we can see that there is not a big difference regarding computation times between $\alpha = 0.00100$ and $\alpha = 0.00010$. Nevertheless, if we consider the number of episodes, SARSA with $\alpha = 0.00100$ is the only one to converge.

With Q-Learning, convergence occurs faster and 510,000 episodes before than with SARSA, considering a learning rate α of 0.100 and a value of $\varepsilon = 0.01$. We might now wonder if by setting a different value of the learning rate α for Q-Learning it will converge faster. In Figure 6.3, we see that by considering different values, convergence in Q-Learning does not improve. In fact, convergence is faster for $\alpha = 0.100$. The values of 0.2, 0.0001, 0.00001 and 0.000001 were also tested but results got worse, so they are not plotted.

In Table 6.4, we see that the faster computation is for $\alpha = 0.001$. Nevertheless, if we look at the number of episodes rather than the computation time, we can clearly see that for a value of $\alpha =$

0.100, convergence occurs at 200,000 episodes, while in the other cases convergence takes more episodes or more time.

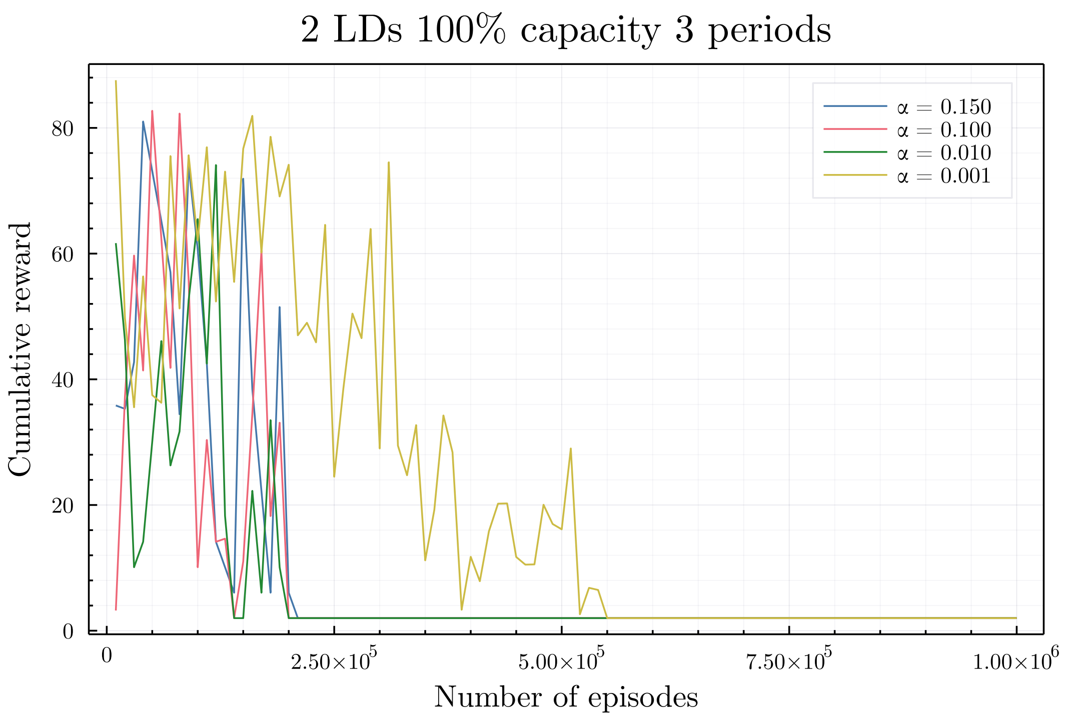

Figure 6.3: Q-Learning algorithm for 2 LDs with 100% capacity at the CD and 3 periods for different values of α. Source: Own elaboration.

Learning Rate α	**Computation Time (sec)**	**Convergence (episodes)**
0.150	65.51	210,000
0.100	59.99	200,000
0.010	66.76	200,000
0.001	57.36	550,000

Table 6.4: Computation times and convergence episodes per each α with Q-Learning algorithm.

To continue this first analysis of the differences between Q-Learning and SARSA algorithms, we now study the impact of ε in Q-Learning, so we compare using ε decaying ($\varepsilon = 1/t$), $\varepsilon = 0.01$ and $\varepsilon = 0.10$, which are commonly used ε values. In all cases we show the results for a learning rate $\alpha = 0.100$. We also tested the previously indicated values of α and the results obtained were worse.

In Table 6.5, the computation times and the number of episodes to converge are shown. We observe that for the value of $\varepsilon = 0.01$, it converges at a smaller number of episodes, 200,000.

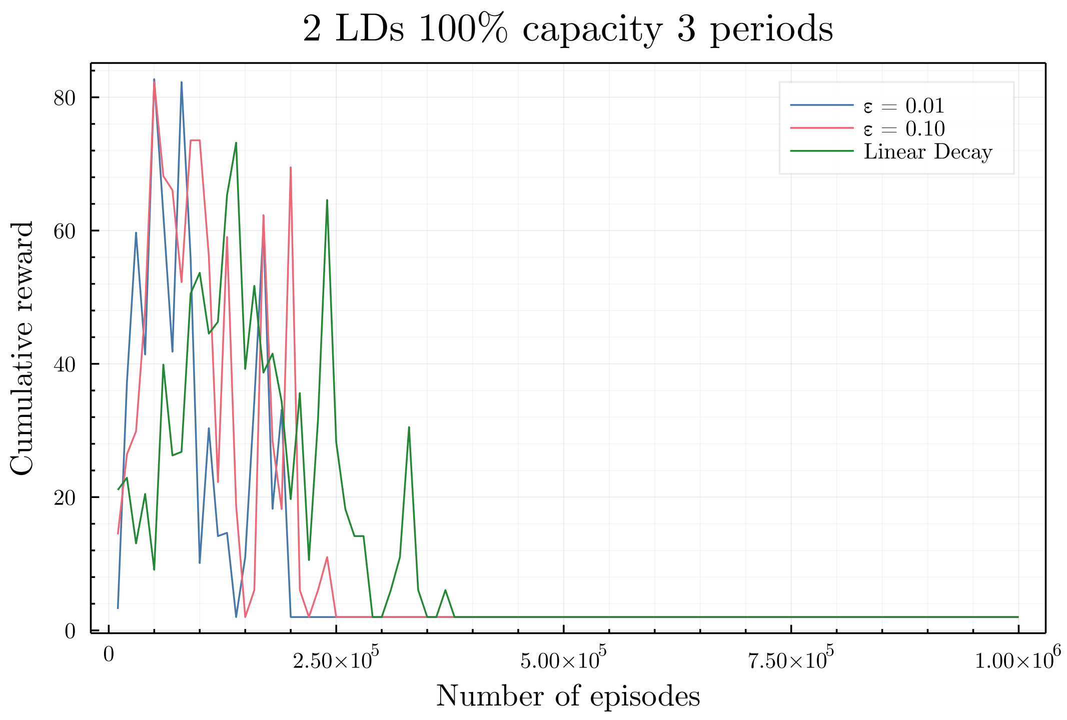

Figure 6.4: Q-Learning algorithm for 2 LDs with 100% capacity at the CD and 3 periods for different values of ε. Source: Own elaboration.

ε	**Computation Time (sec)**	**Convergence (episodes)**
0.01	59.99	200,000
0.10	61.17	250,000
Linear Decay	55.97	380,000

Table 6.5: Computation times and convergence episodes per each ε with Q-Learning algorithm.

In Figure 6.5, the final plot of both algorithms is shown.

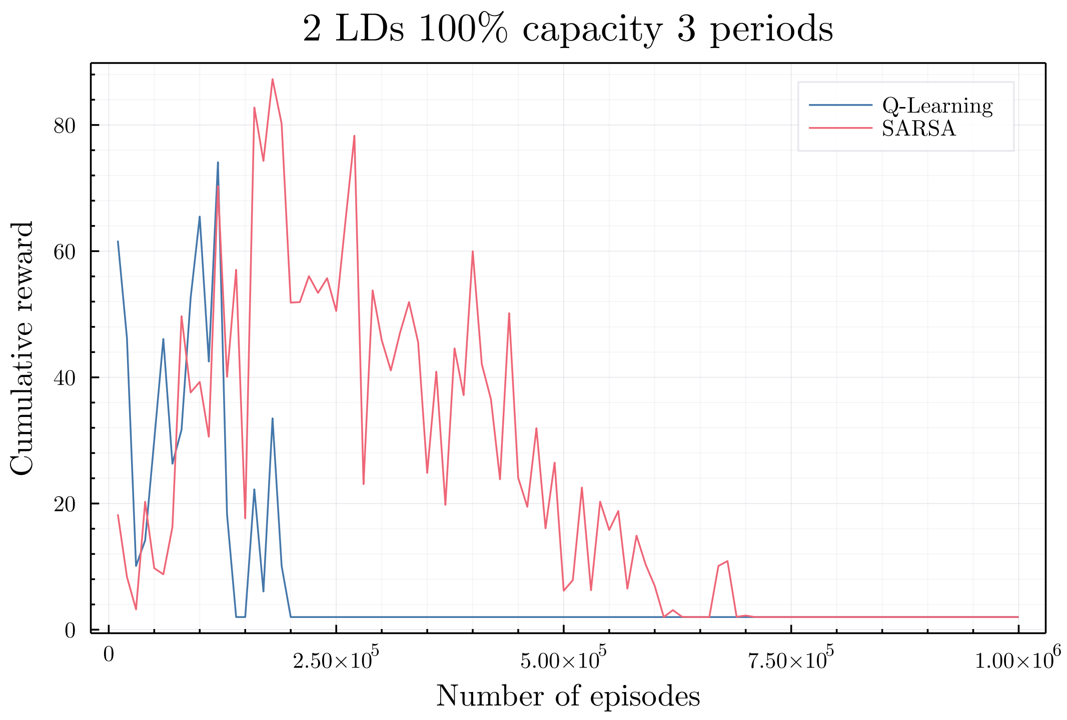

Figure 6.5: Q-Learning and SARSA algorithms using the Dummy State Approach. Source: Own elaboration.

Following the first approach, which is using a *Dummy State* `spt`, Q-Learning is faster than SARSA. In Figure 6.6, we see the results obtained with both approaches and both algorithms.

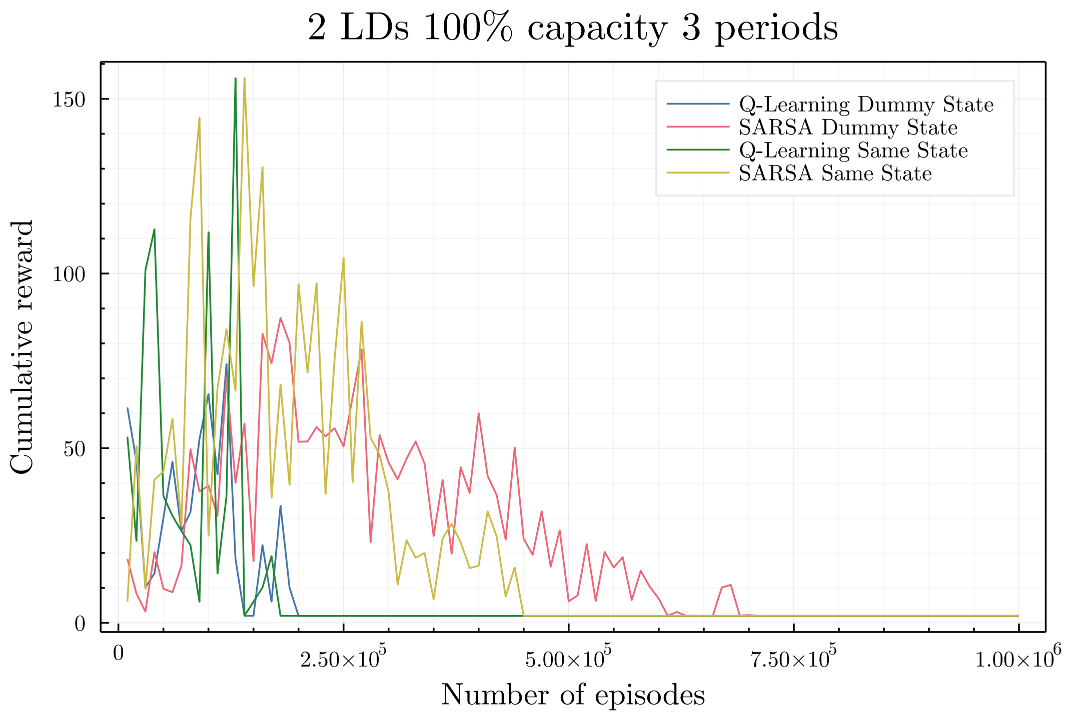

Figure 6.6: Q-Learning and SARSA algorithms using the Dummy State and the Same State Approach. Source: Own elaboration.

From Figure 6.6, it can be concluded that this second approach, denoted as *Same State*, is faster but gets higher cumulative rewards at the beginning. This is because when the agent takes an illegal action, it has to wait until the end of the episode in the same state `s` and the cumulative reward increases. In this example, as the number of periods is small, this second approach gives faster results. Nevertheless, as later it will be shown in Subsection 6.1.1.2, when considering longer periods of time, this approach will give worse results.

In Table 6.6 it is shown that Q-Learning algorithm takes more time than SARSA. However, from Figure 6.6 we can see that convergence occurs considerably before with Q-Learning.

Approach	**Algorithm**	**Learning Rate α**	ε	**Computation Time (sec)**
Dummy State	Q-Learning	0.100	0.01	59.99
Dummy State	SARSA	0.001	Linear Decay	43.14
Same State	Q-Learning	0.100	0.01	43.50
Same State	SARSA	0.001	Linear Decay	38.32

Table 6.6: Comparison of results between algorithms and approaches.

Once the differences between approaches and algorithms have been indicated, the obtained policies are evaluated. They are shown in the Appendix A.1.1.1.

The obtained policies are exactly the same as it was expected. Stock-holding costs are not considered and because demand follows a discrete uniform distribution, the optimal policy is filling the LDs to their maximum storage capacity in order to avoid LDs resupply orders and give a service level of 100%.

We conclude, that despite the fact that both algorithms found the optimal policy, the Q-Learning algorithm obtains results faster than the SARSA algorithm. Therefore, since from now on, more complex configurations will be analysed, Q-Learning will be the only algorithm used.

6.1.1.2 2 LDs with 100% capacity at the CD and 10 periods

Once the simple case of 2 LDs with 100% capacity for 3 periods between CD resupplies has been analysed and it was shown that the Q-Learning algorithm is the most adequate for the current problem, we will evaluate a similar case, but with the difference that now there are 10 periods between CD resupplies and the 2 LDs can only store demand for up to 2 periods.

Q-Learning is the algorithm used. Table 6.7 and Table 6.8 show the problem data and the hyperparameters, respectively. The *Dummy State* and the *Same State* approaches will be considered.

Problem data	**Value**
CD capacity percentage	100%
Central depot stock	80
Local depots stock	8
Number of periods	10
Maximum possible demand	4
State-Space	59,132 states
Action-Space	81 actions

Table 6.7: Problem Data for 2 LDs with 100% capacity at the CD and 10 periods.

Approach	**Algorithm**	**Learning Rate α**	ε	**Computation Time (sec)**
Dummy State	Q-Learning	0.10	0.01	3,154.89
Same State	Q-Learning	0.10	0.01	3,157.55

Table 6.8: Hyper-Parameters for 2 LDs with 100% capacity at the CD and 10 periods.

As the State-Space and the Action-Space grow, time required for computations increases. In fact, considering that the amount of experience required for convergence scales according to $O(|\mathcal{S}| \times |\mathcal{A}|)$, 5,000,000 episodes were considered. This computation took 3,154.89 seconds (52 minutes and 35 seconds) when using a *Dummy State* and 3,157.55 seconds (52 minutes and 38 seconds) when remaining at the *Same State*.

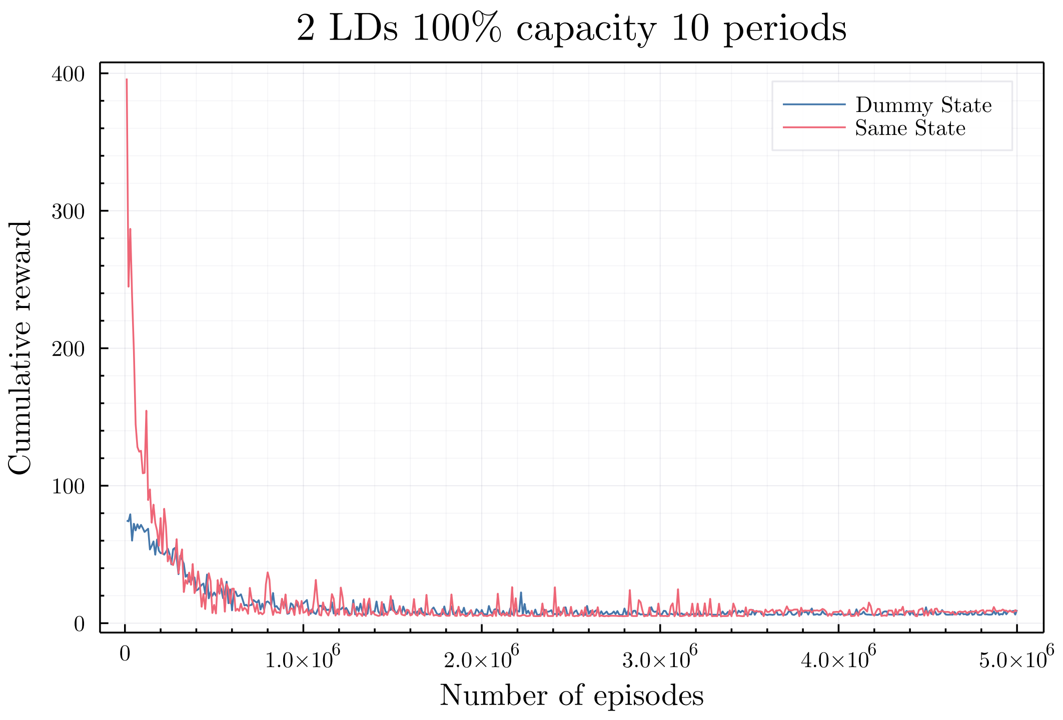

Figure 6.7: Q-Learning for 2 LDs with 100% capacity at the CD and 10 periods using both approaches. Source: Own elaboration.

The resulting policies are shown in the Appendix A.1.1.2. Analysing the policies, it is observed that when using a *Dummy State*, the learned policy follows a similar pattern as the one for the previous case. The agent initially fills the LDs, so there are enough doses for 2 periods. Whenever the stock is bellow the maximum possible demand, it fills again the LDs, except in the last period, when it only sends the doses required to avoid unsatisfied demand.

However, when remaining at the *Same State* the agent decides to make more shipments than necessary, it does not fill the LDs and in the last period it sends more doses than required.

To understand this phenomenon, it is necessary to look at the code of the project, which can be found in the Appendix C.

It is common sense that when remaining in the *Same State*, the agent will have to wait until the end of the episode to finish, while when using a *Dummy State*, whenever the agent takes an illegal action, the episode ends automatically and it has to start a completely new episode.

When considering 3 periods, remaining in the *Same State* gets faster results, but as the number of periods, states and actions increases, it behaves worse and needs to run more episodes to reach a similar solution to the one obtained with the *Dummy State* approach. In addition, `s`, used when following the *Same State* approach, is a local variable, while `spt` used when following the *Dummy State* approach, is a global variable, so that's why computation times are smaller when following the *Same State* approach.

As the main objective of this master thesis is to focus on more realistic cases, where the dimensionality of the problem is bigger, from now on, the *Dummy State* approach will be the only one considered. Furthermore, as the dimensionality of the problem increases, the amount of experience required for convergence also increases, so it is necessary to pre-populate the Q-Table with large negative values for all the illegal actions, as it was previously indicated.

Modifying the Q-Table

In Tables 6.7 and 6.8 from the previous example, we can see that there are 59,132 states and 81 actions, as a consequence, computation times are around 52 minutes. High cumulative rewards in Figure 6.7 indicate that the agent tries to take illegal actions. In order to prevent the agent from taking illegal actions and speed up the computation process, we will pre-populate the Q-Table.

The same algorithm will be used, which is Q-Learning with a learning rate $\alpha = 0.10$ and $\varepsilon = 0.01$.

To better understand the Q-Table modification, Figure 6.8 shows the initial Q-Table whose values are zero (green) and the pre-populate Q-Table, where impossible combinations have a large negative value of -9,999 (red). Several large negative values were tested and results were better with -9,999. On the other side, Figure 6.9 shows the resulting Q-Table if it is not pre-populated. The visited combinations are indicated in yellow. As we can see, there are impossible combinations that the agent visits even though they are not possible. This produces higher and inefficient computations.

The effect of modifying the Q-Table is shown in Figure 6.10 and in Table 6.9. This method takes 1,142.42 seconds (19 minutes and 2 seconds) and 1.48 seconds to create the Q-Table. That is in total, a 63.76% less than the previous ones, due to the fact that at the beginning there are 3,365,361 combinations indicated as impossible. Furthermore, the amount of experience required for convergence is reduced.

The resulting policy is exactly the same as the one obtained when using a *Dummy State*. So, we conclude that by modifying the Q-Table, computation times are reduced, which is of great importance, as the dimensionality of the problem increases.

Approach	**Computation Time (sec)**	**Q-Table Time (sec)**	**Impossible Combinations**
Pre-populate the Q-Table	1,142.42	1.48	3,365,361

Table 6.9: Computation times, Q-Table times and impossible combinations for 2 LDs with 100% capacity for 10 periods.

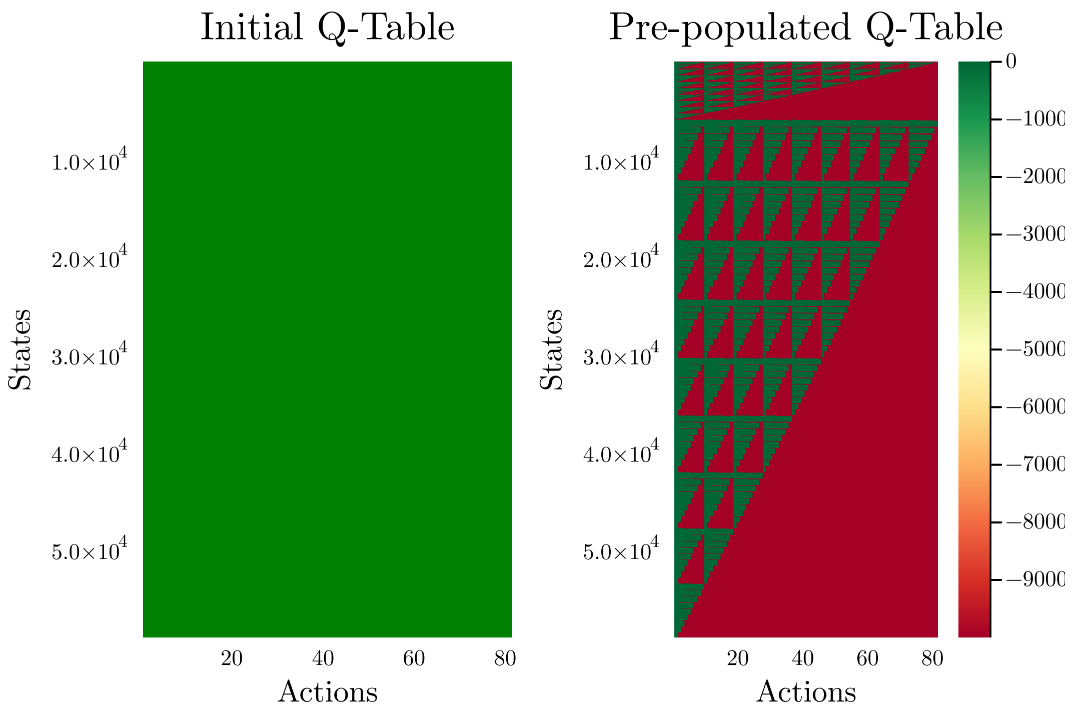

Figure 6.8: Pre-Populated Q-Table for 2 LDs with 100% capacity at the CD and 10 periods. Source: Own elaboration.

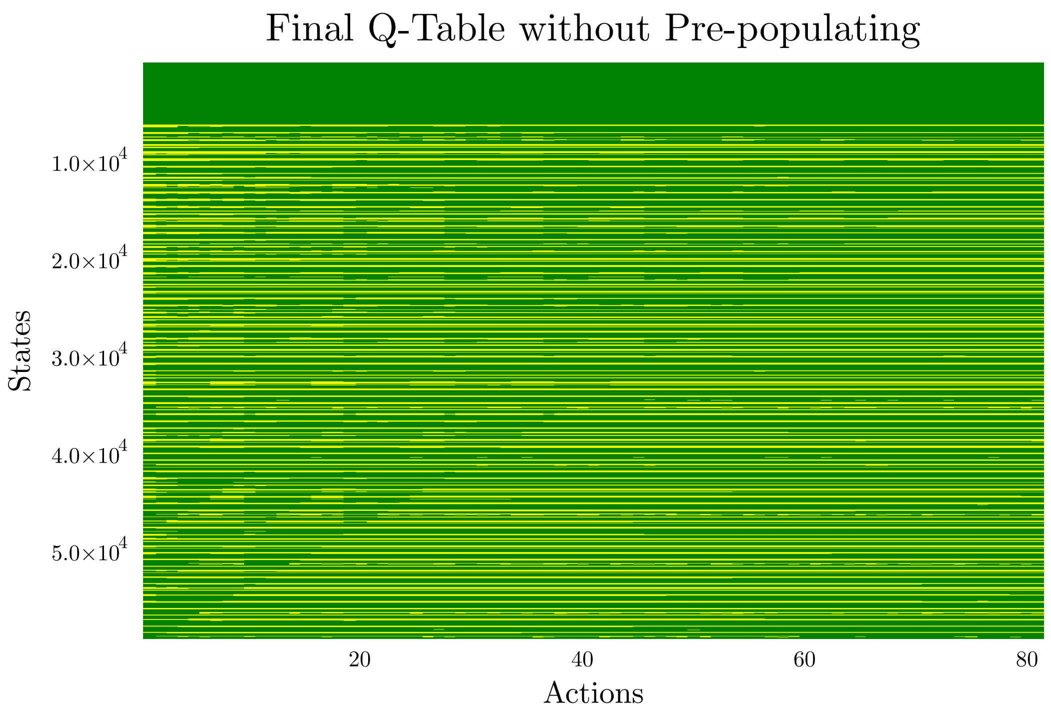

Figure 6.9: Final Q-Table without Pre-Populating for 2 LDs with 100% capacity at the CD and 10 periods. Source: Own elaboration.

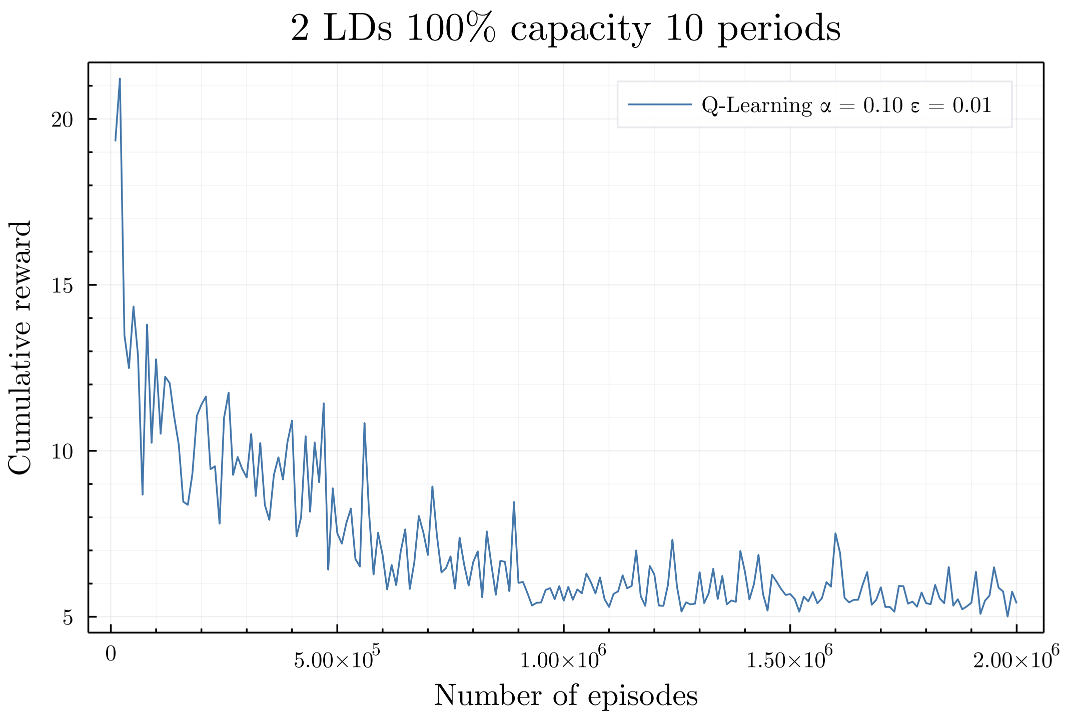

Figure 6.10: Q-Learning algorithm for 2 LDs with 100% capacity at the CD and 10 periods with pre-populated Q-Table. Source: Own elaboration.

This is an ideal situation, where there is enough stock for satisfying the maximum possible demand. In the following subsection we will analyse the present problem but adding capacity restrictions to get a more realistic perspective of the problem.

6.1.2 2 LDs with 90%, 80% 70% and 60% capacity at the CD

In this subsection the capacities of serving the 90%, 80% 70% and 60% of the total maximum demand of the LDs during the whole time horizon will be considered. First, 3 periods will be considered and then, 10 periods.

6.1.2.1 2 LDs with 90%, 80%, 70% and 60% capacity at the CD and 3 periods

As aforementioned, Q-Learning is the used algorithm with a learning rate $\alpha = 0.10$ and $\varepsilon = 0.01$. Table 6.10 and Table 6.11 show the problem data and the hyper-parameters, respectively. As the dimensions of the problems are not that big, Q-Tables won't be pre-populated.

In Table 6.11 it is shown the different number of episodes required for each CD capacity percentage. The lower the capacity, the more episodes are required. Computation times are not directly proportional to the number of episodes.

Problem data	**Values**			
CD capacity percentage	90%	80%	70%	60%
Central depot stock	21	19	16	15
Local depots stock	12	12	12	12
Number of periods	3	3	3	3
Maximum possible demand	4	4	4	4
State-Space	5,084 states	4,118 states	2,789 states	2,018 states
Action-Space	169 actions	169 actions	169 actions	169 actions

Table 6.10: Problem Data for 2 LDs with several capacities at the CD and 3 periods.

CD Capacity %	**Computation Time (sec)**	**No. of episodes**
90%	67.05	2,000,000
80%	61.76	2,000,000
70%	75.38	3,000,000
60%	89.43	4,000,000

Table 6.11: Computation times and convergence episodes per each CD capacity.

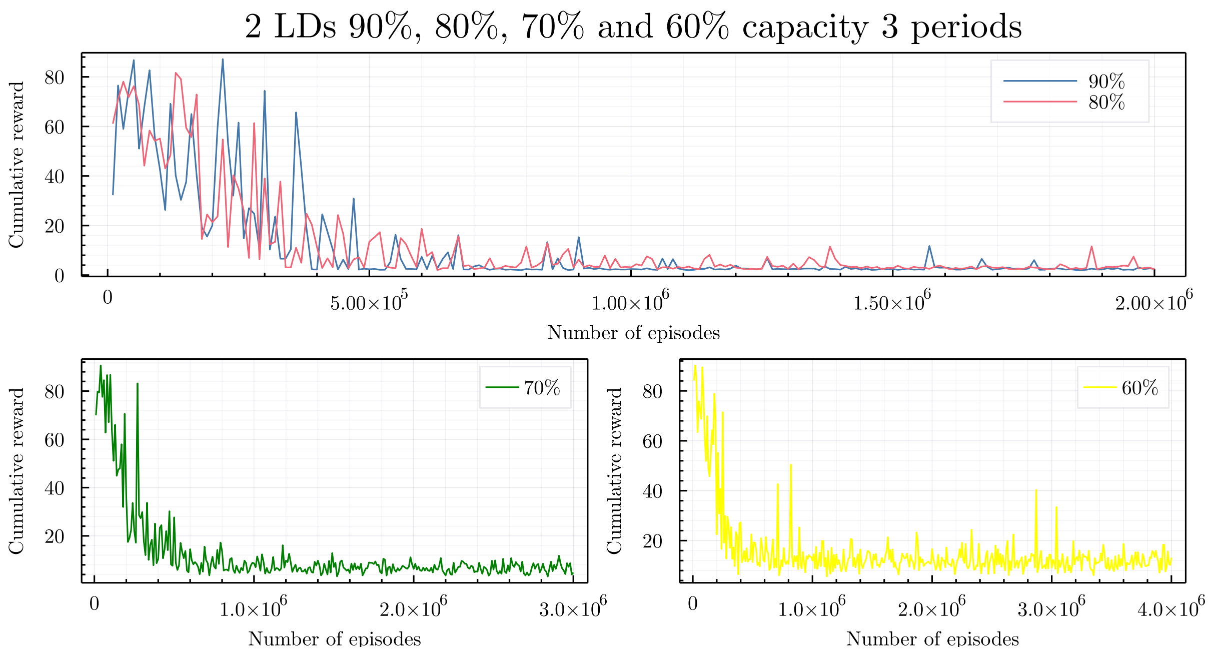

Figure 6.11: Q-Learning algorithm for 2 LDs with several capacities at the CD and 3 periods. Source: Own elaboration.

In Figure 6.11 we show the expected cumulative reward as a function of the number of episodes for the capacities of 90%, 80%, 70% and 60%. Compared to the situation when there is 100% capacity, as the capacity of the CD decreases, the plot becomes more wavy because it gets more difficult to satisfy all the demand, so the cumulative reward increases. Remember that in this problem, cumulative reward means the cost derived from the number of shipments.

The resulting policies for each capacity (90%, 80%, 70% and 60%) are shown in the Appendices A.1.2.1, A.1.2.2, A.1.2.3 and A.1.2.4 respectively.

Regarding 90% capacity, the 3 simulations show different possible situations. The initial shipment is of 10 doses per LD. In Simulation 1 all the demand is satisfied with only 1 shipment. In Simulation 2 all the demand is satisfied with 2 shipments. Nevertheless, in Simulation 3 not all the demand is satisfied because demand in the LD 1 is 4 during the whole time horizon and doses cannot be sent from the other LD.

When analysing the case of 80% capacity, the initial shipment is of 9 doses per LD and the 3 simulations are similar to the case of 90% capacity. In Simulations 3 and 2 all the demand is satisfied with 1 and 2 shipments respectively. In Simulation 1 not all the demand is satisfied. There are 2 missing doses.

For 70% capacity, as there is less capacity at the CD, less demand is satisfied. The initial shipment is of 8 doses per LD. In Simulation 1 all the demand is satisfied with 1 shipment. In Simulations 2 and 3 not all the demand is satisfied, there is 1 missing dose in Simulation 2 and 3 missing doses in Simulation 3.

Finally, for 60% capacity the initial shipment is of 6 doses per LD. As for the case of 70% capacity, when capacity decreases it gets difficult to satisfy all the demand. In fact, in Simulation 1, 3 shipments are required, while in Simulations 2 and 3 there are 2 and 4 missing doses respectively.

In Table 6.12 we can see the relationship between the CD capacity and the number of doses of the initial shipment. The amount of doses to be sent at the start is directly proportional to the CD capacity and the LD capacity as follows:

$$\lfloor CD_{capacity}\% \cdot LD_{capacity} \rfloor$$

Nevertheless, this is not true when the CD capacity is 60%. The agent half fills the LDs, because it is more difficult to satisfy all the demand with less doses at the CD.

Analysing the number of doses left after initial shipment, we see that the agent leaves 1 dose when the CD capacity is 90% and 80%, 0 doses when it is 70% and 2 doses when it is 60%. So, when the CD capacity is lower, the agent decides to keep inventory at the CD to avoid wasting doses by sending them to a LD that does not need them.

CD Capacity %	**Initial Shipment (doses)**	**Doses left after initial shipment**
90%	10	1
80%	9	1
70%	8	0
60%	6	2

Table 6.12: Number of doses of the initial shipment based on the CD capacity and doses left after initial shipment.

6.1.2.2 2 LDs with 90%, 80%, 70% and 60% capacity at the CD and 10 periods

We will evaluate a similar case to the one shown in Subsection 6.1.1.2. However, after doing several tests and obtaining similar results, it has not been necessary to run more episodes for 70% and 60% capacity. This is due to the fact that the capacity of the LDs is limited to 2 periods.

Problem data is shown in Table 6.13. As the State-Space and Action-Space increase, Q-Tables were pre-populated. In Table 6.14 computation, Q-Tables times and impossible combinations are shown.

Problem data	**Values**			
CD capacity percentage	90%	80%	70%	60%
Central depot stock	72	64	56	48
Local depots stock	8	8	8	8
Number of periods	10	10	10	10
Maximum possible demand	4	4	4	4
State-Space	52,562 states	46,173 states	39,692 states	33,212 states
Action-Space	81 actions	81 actions	81 actions	81 actions

Table 6.13: Problem Data for 2 LDs with several CD capacities and 10 periods.

CD Capacity %	Computation Time (sec)	Q-Table Time (sec)	Impossible Combinations
90%	1,143.82	1.53	3,002,481
80%	923.88	1.19	2,639,601
70%	878.51	1.16	2,276,721
60%	685.99	0.83	1,913,841

Table 6.14: Computation times, Q-Table times and impossible combinations for 2 LDs with several CD capacities and 10 periods.

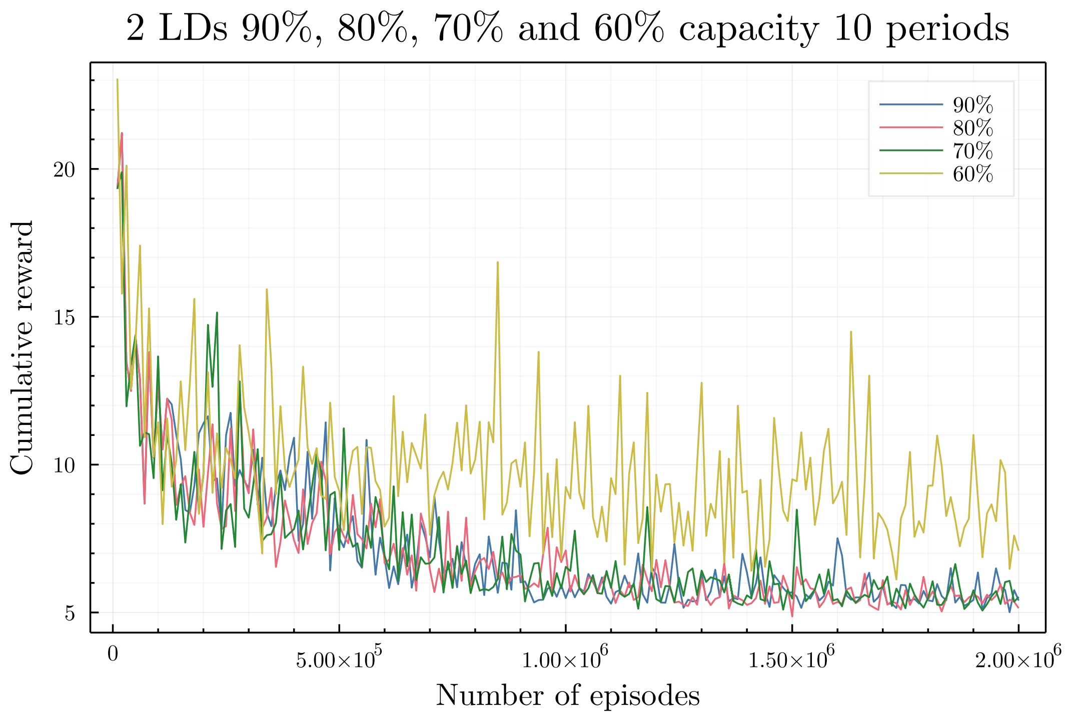

Figure 6.12: Q-Learning algorithm for 2 LDs with several CD capacities and 10 periods. Source: Own elaboration.

From Figure 6.12 we can see a similar behaviour to the one from Figure 6.11. As the capacity of the CD decreases, it is more challenging to properly satisfy all the demand, and as a consequence, the number of shipments increases, especially for the case of 60%. This means that when the CD capacity is lower, by withholding more doses at the CD and doing more shipments, unused doses are avoided in order to limit the number of missing doses.

More episodes were tested for the cases of 60% and 70%, but results were exactly the same. One of the resulting policies for each capacity (90%, 80%, 70% and 60%) are shown in the Appendices A.1.2.5, A.1.2.6, A.1.2.7 and A.1.2.8 respectively.

When there is a capacity of 90%, 80% and 70% the agent decides to fill the LDs as for the case of 100% capacity and 10 periods. However, when capacity is 60%, the agent fills the LDs at the beginning and in the next periods does more shipments than required and does not send the same number of doses to each LD. As a result of the lower capacity of the CD and this policy, all the demand of the LDs cannot be satisfied.

6.2 3 LOCAL DEPOTS

To continue this analysis, we now evaluate the case of 1 CD which has to provide doses to 3 LDs.

Following a similar structure as in Section 6.1, first the case of 100% capacity at the CD will be studied. Later, lower CD capacities and more periods will be considered. Nevertheless, in order to get a wider and more realistic point of view, the simulations performed will be different from those performed in Section 6.1.

6.2.1 3 LDs with 100% capacity at the CD and 3 periods

When the number of LDs increases, the number of states and actions also increases. Consequently, the problem is closer to reality but more computations are needed.

Q-Learning is the used algorithm with a learning rate $\alpha = 0.10$ and $\varepsilon = 0.01$. Initially, the Q-Table was not pre-populated, but since the State-Space and the Action-Space increase, it took a lot of time and episodes to reach to the solution. Consequently, after this test, the Q-Table was pre-populated.

Tables 6.15 and 6.16 show the problem data and the effects of pre-populating the Q-Table. When the Q-Table is pre-populated, the number of necessary episodes to converge is reduced by 92.81% (28,400,000 episodes less) and the time is reduced by 74.93% (7 hours 37 minutes and 12 seconds less). In Figure 6.13 both plots are shown.

Resulting policy can be consulted in Appendix A.2.1. This policy is very similar to the one obtained in Subsection 6.1.1.1, the agent decides to fill the LDs to their maximum storage capacity at the beginning in order to avoid LDs resupplies.

As the resulting policy is the same and by pre-populating the Q-Table computations are faster, from now on, Q-Table will be pre-populated.

Problem data	Value
CD capacity percentage	100%
Central depot stock	36
Local depots stock	12
Number of periods	3
Maximum possible demand	4
State-Space	125,231 states
Action-Space	2,197 actions

Table 6.15: Problem Data for 3 LDs with 100% capacity at the CD and 3 periods.

Approach	**No. of Episodes**	**Computation Time (sec)**	**Q-Table Time (sec)**	**Imp. Combinations**
Non-Pre-pop. Q-Table	30,600,000	36,612.25	0	0
Pre-pop. Q-Table	2,200,000	9,180.36	314.67	245,741,041

Table 6.16: Effects of pre-populating the Q-Table for 3 LDs with 100% capacity at the CD and 3 periods.

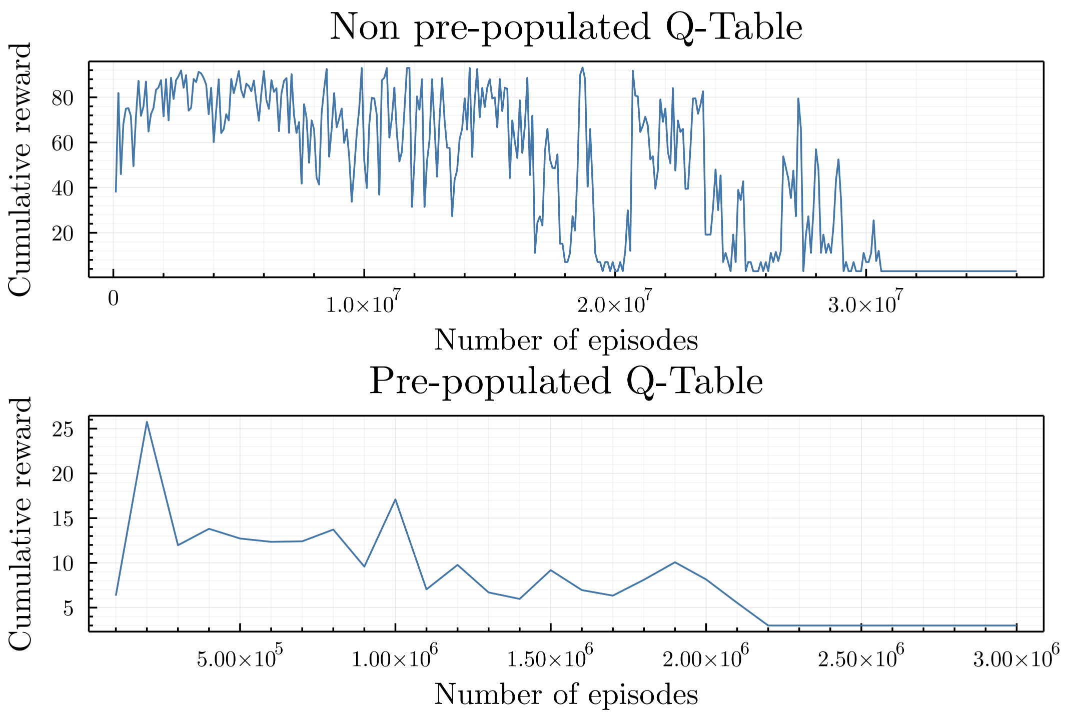

Figure 6.13: Q-Learning algorithm for 3 LDs with 100% capacity at the CD and 3 periods. Source: Own elaboration.

6.2.2 3 LDs with 80% and 70% capacity at the CD and 5 periods

We continue studying a more realistic case where CD capacities are 80% and 70%. There are 3 LDs, with a maximum possible demand of 3 doses and each LD can store doses for 3 periods. Problem data is shown in Table 6.17. Table 6.18 shows the time required for pre-populating the Q-Table, the computation time and the number of impossible combinations.

Problem data	**Values**	
CD capacity percentage	80%	70%
Central depot stock	36	31
Local depots stock	9	9
Number of periods	5	5
Maximum possible demand	3	3
State-Space	117,502 states	92,502 states
Action-Space	1,000 actions	1,000 actions

Table 6.17: Problem Data for 3 LDs with several CD capacities and 5 periods.

CD Capacity %	**Computation Time (sec)**	**Q-Table Time (sec)**	**Impossible Combinations**
80%	32,208.72	57.68	101,695,375
70%	17,255.66	55.74	80,854,750

Table 6.18: Computation times, Q-Table times and impossible combinations for 3 LDs with several CD capacities and 5 periods.

Resulting policies are in the Appendices A.2.2.1 and A.2.2.2 respectively. Figure 6.14, shows the cumulative reward as function of the number of episodes. When capacity is 80%, cumulative reward is lower than when capacity is 70%. To understand why this happens, it is necessary to look at the resulting policies. When capacity is 80%, the system has more maneuverability to properly send doses to the LDs where they are needed. Nevertheless, when capacity is 70%, there is more uncertainty and it gets more difficult to properly satisfy all the demand, and that's why there are missing doses. As we can see, this situation follows a similar pattern to the previous cases where CD capacity was low.

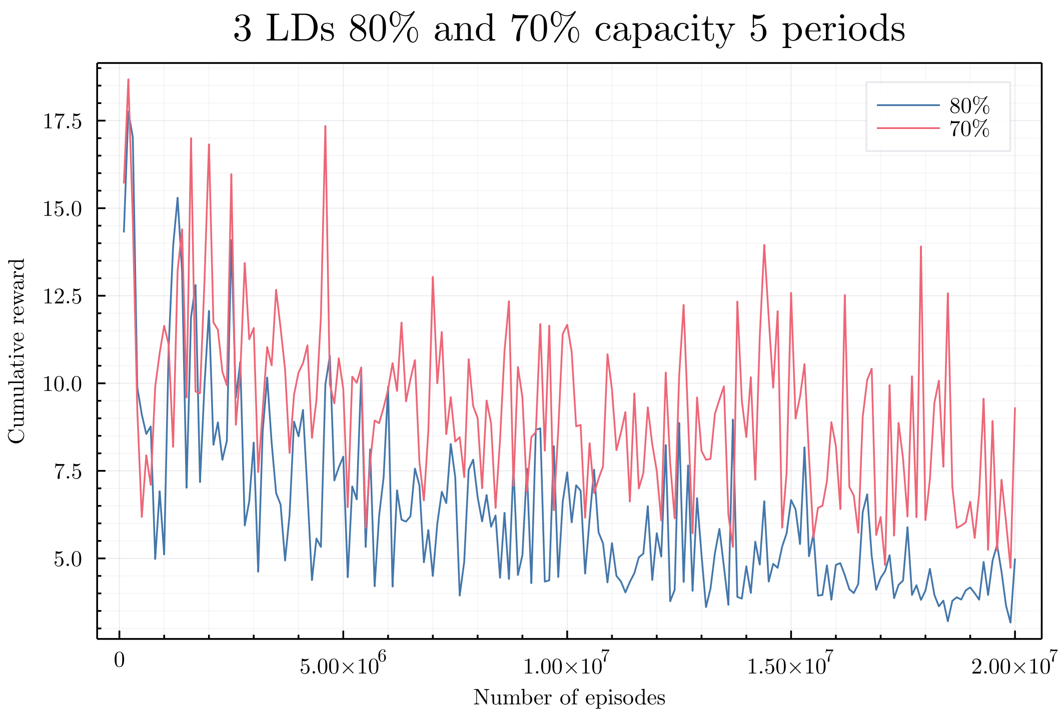

Figure 6.14: Q-Learning algorithm for 3 LDs with several CD capacities and 5 periods. Source: Own elaboration.

6.2.3 3 LDs with 80% and 70% capacity at the CD and 7 periods

Once the effectiveness of pre-populating the Q-Table has been proved, we now analyse an extended version of the problem described in the Subsection 6.2.2. The problem is described in Table 6.19. Table 6.20 shows the time required for pre-populating the Q-Table, the computation time and the number of impossible combinations.

One simulation for the resulting policies can be found in Appendices A.2.3.1 and A.2.3.2. These policies follow the same pattern as the ones obtained in previous sections. The agent fills the LDs to their maximum capacity, except in the last period, when it only sends the necessary doses to satisfy all the possible demand. When CD capacity is 70%, there is a missing dose in the last period because there are less resources to satisfy demand.

Problem data	**Values**	
CD capacity percentage	80%	70%
Central depot stock	67	58
Local depots stock	8	8
Number of periods	7	7
Maximum possible demand	4	4
State-Space	285,770 states	239,843 states
Action-Space	729 actions	729 actions

Table 6.19: Problem Data for 3 LDs with several CD capacities and 7 periods.

CD Capacity %	Computation Time (sec)	Q-Table Time (sec)	Impossible Combinations
80%	41,899.06	98.59	175,156,101
70%	38,674.49	75.83	147,416,193

Table 6.20: Computation times, Q-Table times and impossible combinations for 3 LDs with several CD capacities and 7 periods.

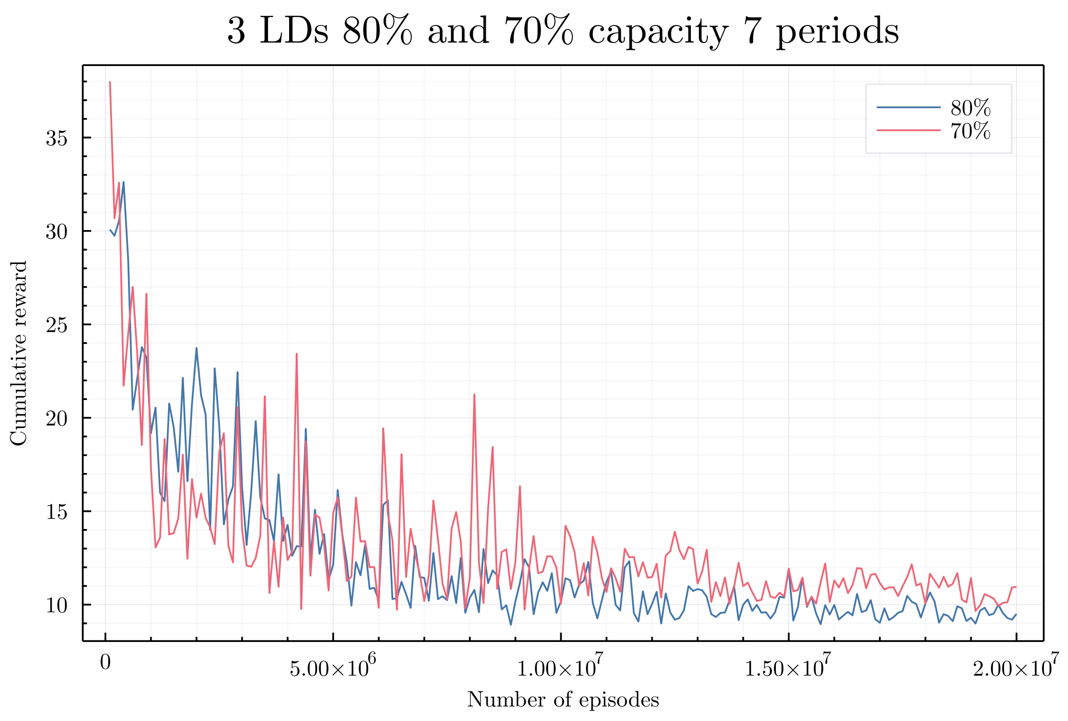

Figure 6.15: Q-Learning algorithm for 3 LDs with several CD capacities and 7 periods. Source: Own elaboration.

6.3 QUANTITATIVE ANALYSIS

We present quantitatively the results obtained from the previous computational experiences.

First, we will focus on the β Service Level (Type 2) for each configuration. Then, their respective computational performances will be evaluated.

6.3.1 Service Level

As aforementioned, we will use the β Service Level (Type 2), which is a quantity-oriented performance measure, which indicates the expected proportion of demand that is satisfied.

Table 6.21 indicates the mean discounted reward and the β Service Level for each of the previous configurations. 50 simulations were done for each configuration.

We begin by analysing the case of 2 and 3 LDs with 100% CD capacity, which is shown in Figure 6.16. It shows the demand of doses in blue, the number of doses served in grey and the mean discounted reward. These configurations give a service level of 100.00%, which means that all the demand is satisfied.

Configuration	Mean Discounted Reward	Service Level
2 LDs / 100% / 3 Periods	2.00	100.00%
2 LDs / 90% / 3 Periods	2.57	99.26%
2 LDs / 80% / 3 Periods	3.65	96.04%
2 LDs / 70% / 3 Periods	7.52	93.03%
2 LDs / 60% / 3 Periods	12.96	86.08%
2 LDs / 100% / 10 Periods	5.23	100.00%
2 LDs / 90% / 10 Periods	5.40	99.03%
2 LDs / 80% / 10 Periods	5.49	95.31%
2 LDs / 70% / 10 Periods	5.62	90.19%
2 LDs / 60% / 10 Periods	7.61	85.76%
3 LDs / 100% / 3 Periods	3.00	100.00%
3 LDs / 80% / 7 Periods	9.63	95.74%
3 LDs / 70% / 7 Periods	10.95	89.15%
3 LDs / 80% / 5 Periods	5.61	96.35%
3 LDs / 70% / 5 Periods	7.56	91.71%

Table 6.21: Mean discounted reward and service level per each configuration.

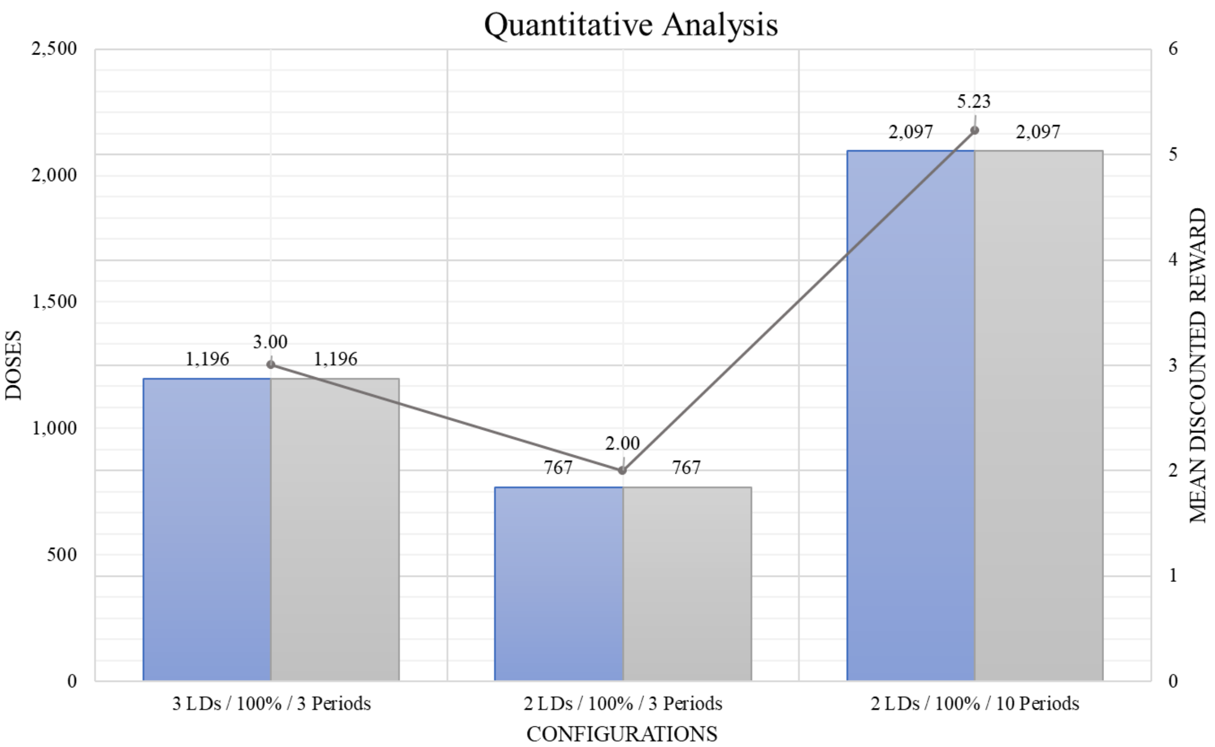

Figure 6.16: Quantitative analysis for 2 and 3 LDs with 100% capacity at the CD. Source: Own elaboration.

As we can see, when LDs have enough storage capacity for the whole time horizon, the more LDs, the higher the mean discounted reward. Logically, this increase is proportional to the number of LDs. Moreover, when there are limits in the LDs storage capacity, it is necessary to do more shipments to properly satisfy the demand.

Figure 6.17 shows the cases of 2 LDs considering several CD capacities. It is trivial that when considering 10 periods, the total demand of doses is bigger than when considering 3 periods. When the CD capacity is 90% and 80%, the more periods considered, the higher the mean discounted reward. Nevertheless, when the CD capacity is 70% and 60%, the mean discounted reward decreases as the number of periods increase.

To understand why this happens, we should look at the resulting policies shown in the Appendix A.1.2 and Equation 2.1 from Chapter 2. When there are 3 periods and the CD capacity is 90% and 80%, there is more stock available, so the agent decides to send more doses at the beginning than when CD capacity is 70% and 60%. By doing this, there is less probability that LDs resupplies are required and demand is not satisfied.

However, when there are 10 periods, the agent can send doses only for the maximum possible demand during 2 periods, so it will have to do more LDs resupplies. The difference between CD capacities 90%-80% and 70%-60%, is that in both cases the agent fills the LDs at the beginning, but as capacity is lower, more demand is not satisfied at the end of the episode and its cost is multiplied by the discount factor raised to a higher power. In fact, mean discounted reward is higher for the capacities of 70% and 60% than for 90% and 80% because there are more missing doses.

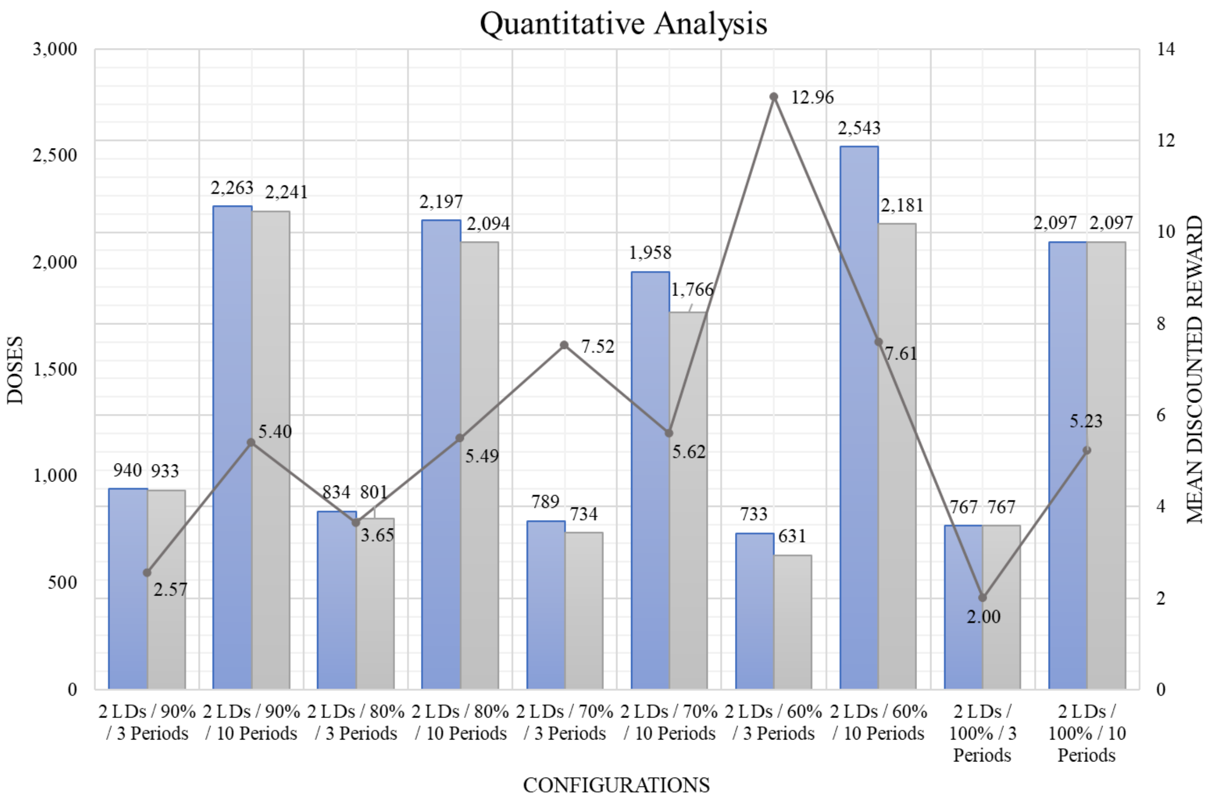

Figure 6.17: Quantitative analysis for 2 LDs with several CD capacities. Source: Own elaboration.

As aforementioned, when the CD capacity is lower, the agent decides to keep inventory at the

CD to avoid wasting doses by sending them to a LD that does not need them.

As it was previously indicated γ must be within the interval of 0 and 1, to adjust the importance of rewards over time. The value of $\gamma = 1$ was also tested, but results were basically the same as the main problem is the lack of capacity at the CD.

In Figure 6.18 we see the mean number of missing doses for 2 LDs and the mean discounted reward, considering several CD capacities. As we can see, the agent tries to limit the number of missing doses and in order to do it, it decides to increase regularly the number of shipments when the CD capacity decreases.

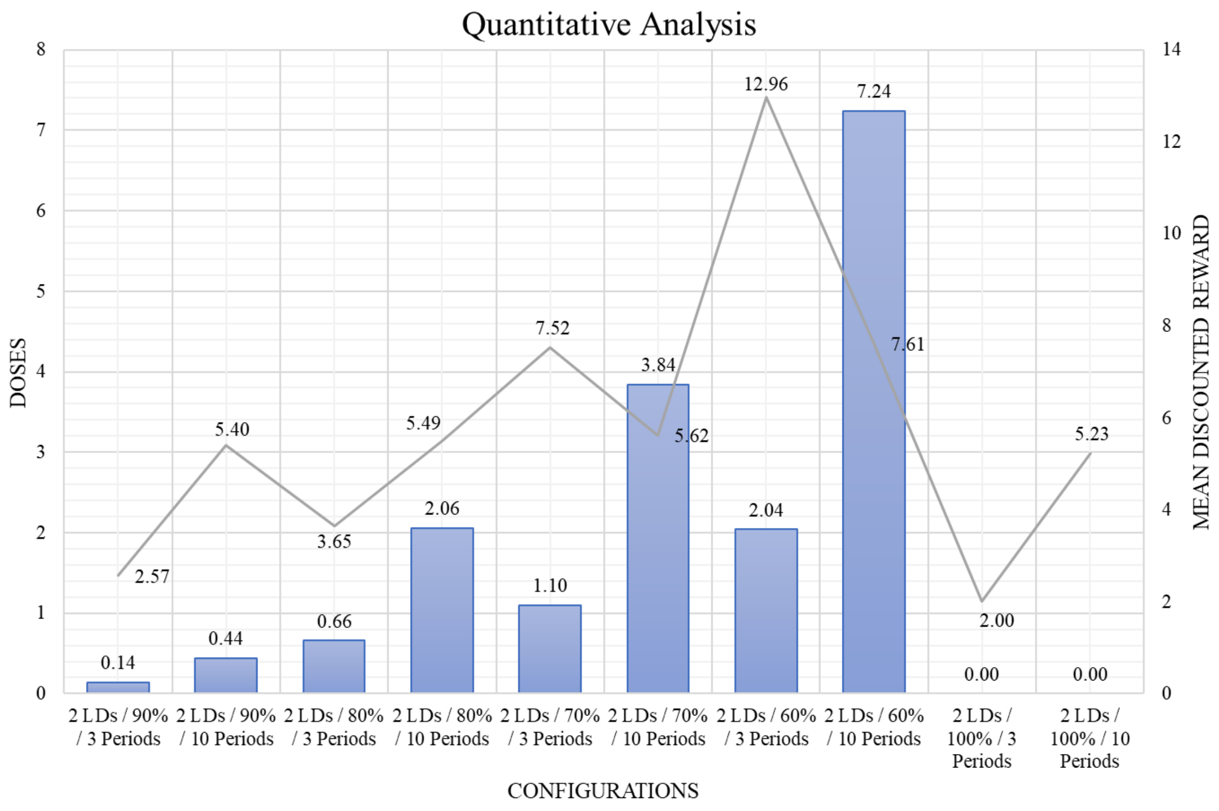

Figure 6.18: Mean number of missing doses and mean discounted reward for 2 LDs with several CD capacities. Source: Own elaboration.

With respect to 3 LDs in Figure 6.19 we can see the demand of doses, the number of doses served and the mean discounted reward. Results are similar to when we consider 2 LDs. As the number of periods increases, the number of doses and the mean discounted reward increase. Furthermore, as the CD capacity decreases, the mean discounted reward increases. Nevertheless, it is important to take into account that the maximum possible demand per period is 3 in the configurations 3 LDs/ 80%/ 5 Periods and 3 LDs/ 70%/ 5 Periods, and 4 for 3 LDs/ 80%/ 7 Periods and 3 LDs/ 70%/ 7 Periods.

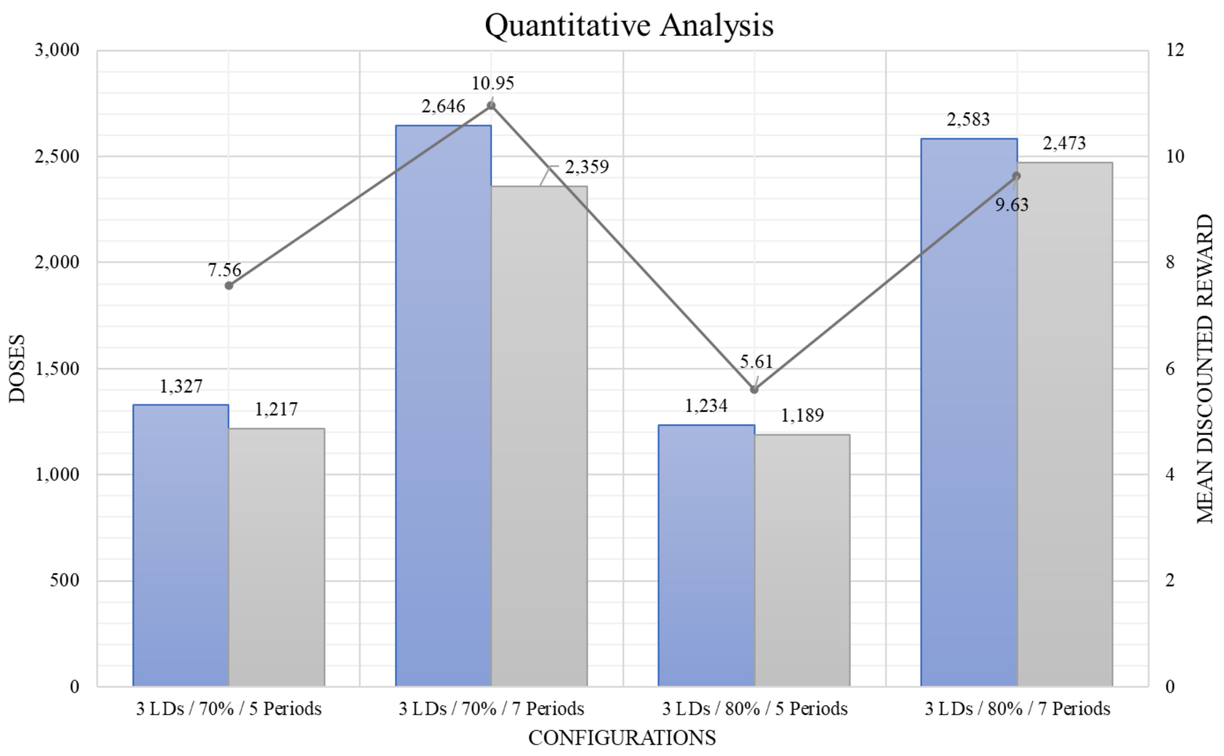

Figure 6.19: Quantitative analysis for 3 LDs with several CD capacities. Source: Own elaboration.

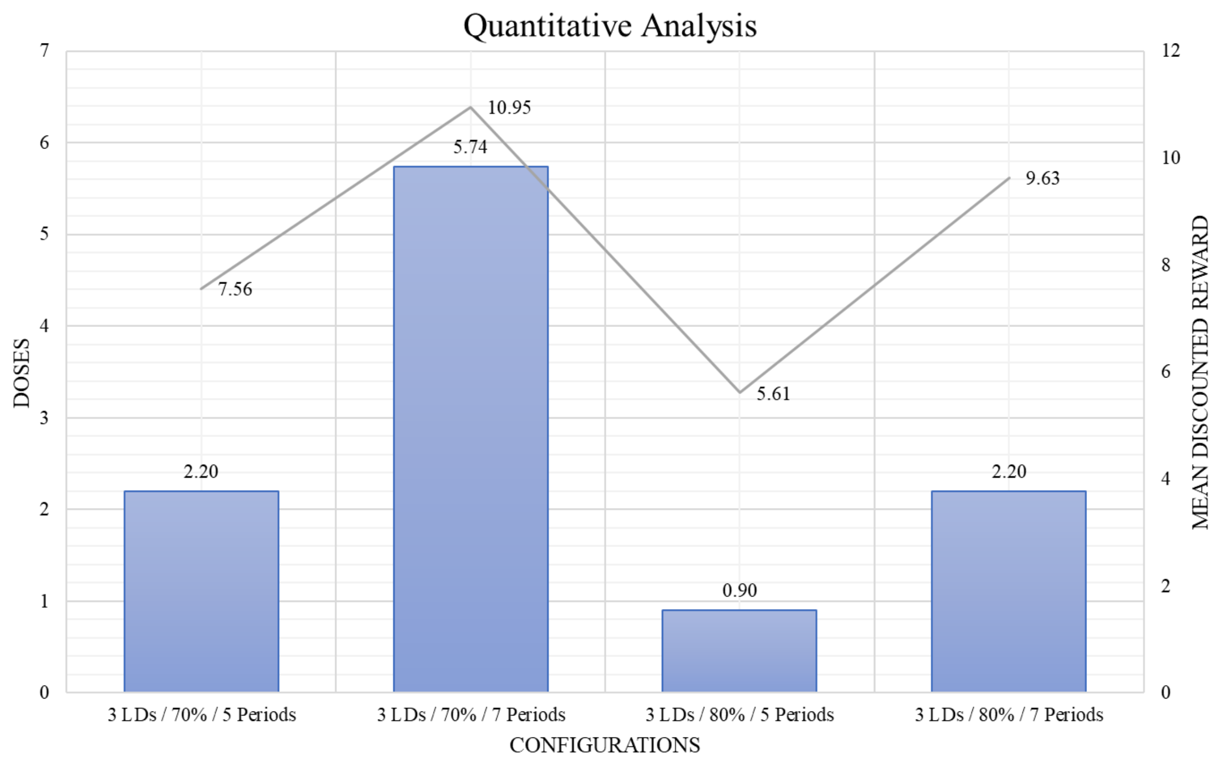

Figure 6.20: Mean number of missing doses and mean discounted reward for 3 LDs with several CD capacities. Source: Own elaboration.

Similarly as what we did for 2 LDs, in Figure 6.20 we see the mean number of missing doses and mean discounted reward for 3 LDs with several CD capacities. Again, as the CD capacity decreases, the agent limits the number of missing doses by increasing the number of shipments.

6.3.2 Performance

In Chapter 6 we have pre-populated the Q-Table when the State-Space and the Action-Space grow, because there will be more impossible combinations and in order to speed up computations it is a very useful procedure. Now, we will quantitatively analyse its impact.

In Table 6.22 we can see the total time of computation, the number of total combinations and the percentage of impossible combinations over the total combinations for each of the considered configurations.

Configuration	Combinations	% Imp. Combinations	Total Time (sec)
2 LDs / 100% / 10 Periods	4,789,692	70.26%	1,143.90
2 LDs / 90% / 10 Periods	4,257,522	70.52%	1,145.35
2 LDs / 80% / 10 Periods	3,740,013	70.58%	925.07
2 LDs / 70% / 10 Periods	3,215,052	70.81%	879.67
2 LDs / 60% / 10 Periods	2,690,172	71.14%	686.82
3 LDs / 100% / 3 Periods	275,132,507	89.32%	9,495.03
3 LDs / 80% / 7 Periods	208,326,330	84.08%	41,997.65
3 LDs / 70% / 7 Periods	174,845,547	84.31%	38,750.32
3 LDs / 80% / 5 Periods	117,502,000	86.55%	21,266.50
3 LDs / 70% / 5 Periods	92,502,000	87.41%	17,311.40

Table 6.22: Computation times, total combinations and percentage of impossible combinations per each configuration.

Undoubtedly, the proportion of impossible combinations is quite high. To get a better idea of the magnitude of this, lets consider the following graphics.

As we did in previous sections, we first attend to the case of 2 LDs. In Figure 6.21 we can see the number of total combinations in blue, the number of impossible combinations in grey and the total time of computation.

Time of computation is proportional to the number of combinations, as the number of combinations increases, the computation time also increases. Nevertheless, for the configuration of 2 LDs/ 100%/ 10 Periods, computation time is lower than for 2 LDs/ 90%/ 10 Periods. So the CD capacity is also relevant. As there are more doses, it is easier for the agent to satisfy all the demand and so, computations are faster.

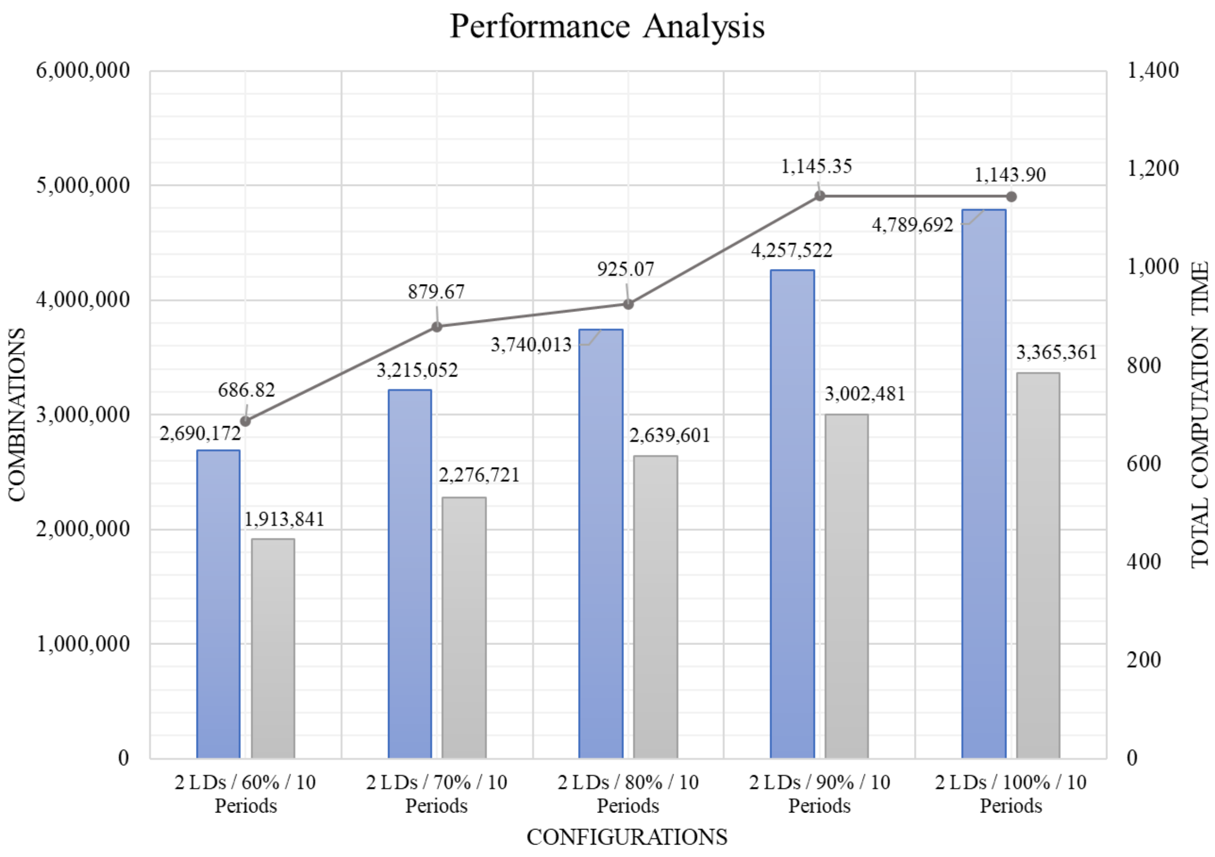

Figure 6.21: Performance analysis for 2 LDs with several CD capacities. Source: Own elaboration.

In fact, if we now consider 3 LDs, in Figure 6.22 we can see a similar sequence. The computation time increases as the number of combinations increases. From this figure, it is important to pay attention to the configuration 3 LDs/ 100%/ 3 Periods. This configuration shows the smaller time of computation, 9,495.3 seconds (2 hours 38 minutes and 15 seconds). Although it has the highest number of combinations, from previous sections we have seen that the resulting policy was trivial. So, dimensionality of the problem is not the only factor which influences on the time of computation, the complexity of the problem also influences.

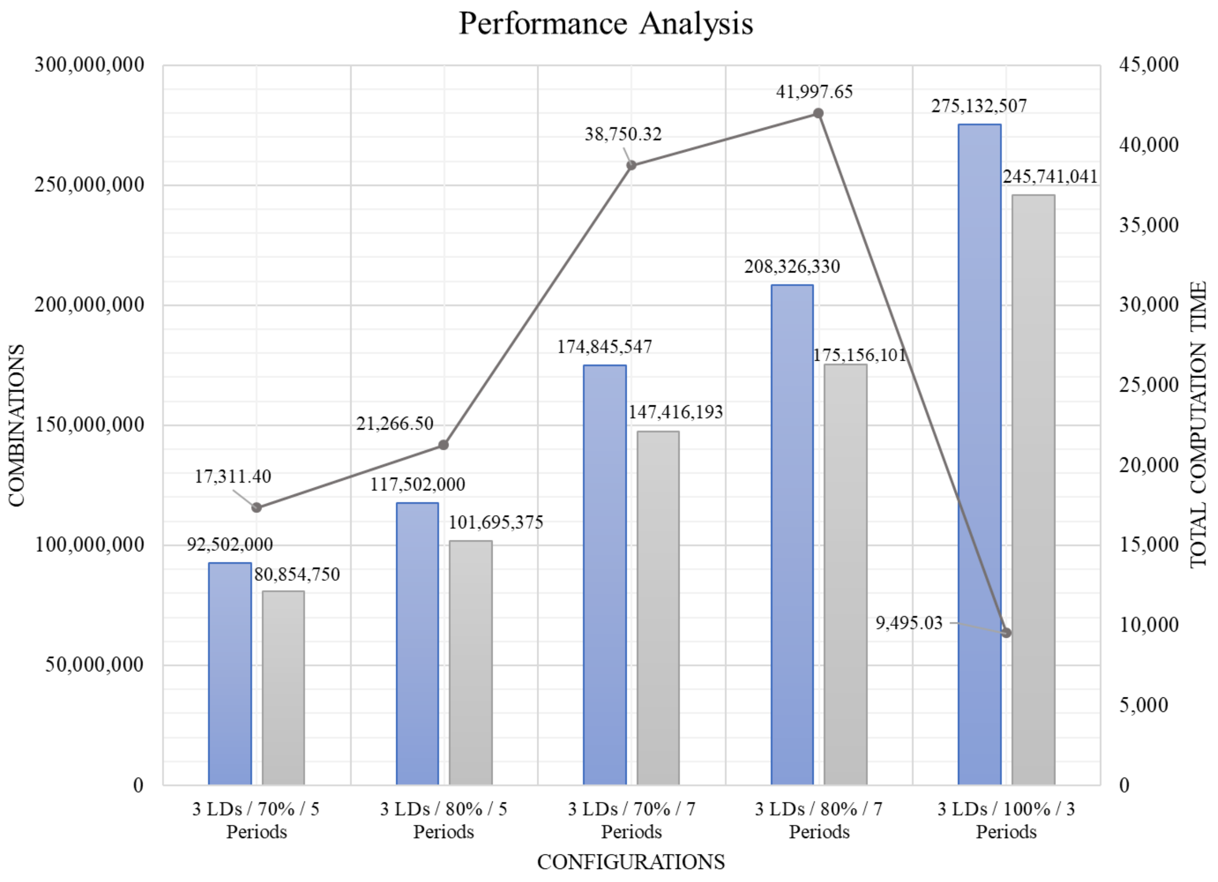

Figure 6.22: Performance analysis for 3 LDs with several CD capacities. Source: Own elaboration.

In Figure 6.23, we can see the number of possible combinations computed per second in each configuration. As it was already stated, as the dimensionality and complexity of the problem increase, computation times increase, and so less combinations are computed per unit of time. Nevertheless, as we can see for the case of 3 LDs/ 100%/ 3 Periods, the speed of computations does not only depend on the number of combinations, the ease with which the agent finds the solution is of great importance.

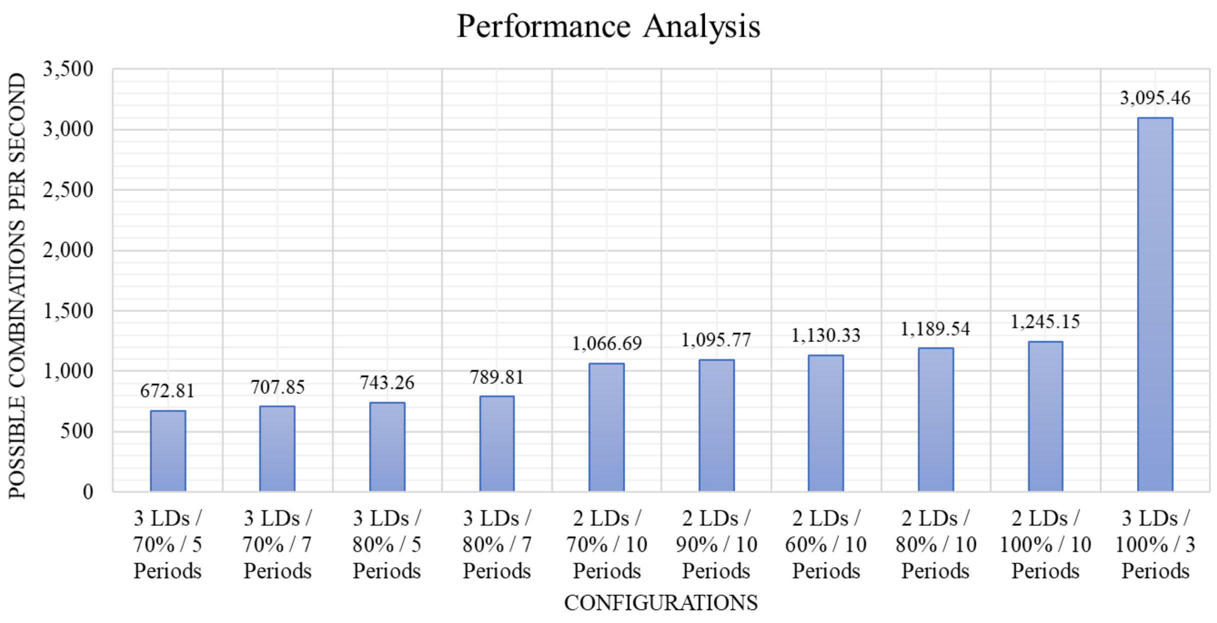

Figure 6.23: Number of possible combinations computed per second. Source: Own elaboration.

In this chapter, we have carried out the quantitative analysis distinguishing between service level and performance.

We found that problem data and hyper-parameters have an important high impact on the service level and performance. In the next chapter, conclusions and further work will be addressed.

CHAPTER 7

Conclusions and further work

Once the quantitative analysis is completed, it is interesting to propose new ways to continue this project in order to give it continuity in the future.

In this chapter we summarise the conclusions of the research found during the development of this master thesis, and we also propose possible extensions.

7.1 CONCLUSIONS

At the beginning of this master thesis we have introduced all the necessary background regarding RL, AI and MDPs.

Then, the problem to be solved has been presented and we also have introduced the interface *POMDPs.jl*, which is an open-source framework that allows implementing POMDPs and MDPs in Julia language. In fact, this interface enabled us to implement and solve our own MDP.

To solve the problem we have used the tabular methods Q-Learning and SARSA, provided by *TabularTDLearning.jl*. When the State-Space and the Action-Space are small enough to be represented as tables, tabular RL methods show a good convergence property. However, they suffer from the curse of dimensionality. This means that, as the magnitude of the problem increases, more computations are necessary and therefore, it takes more time to reach the solution.

Illegal actions play an important role. Ideally we would not allow the agent to take illegal actions in a current state because if there are several illegal actions, it will cost the agent a lot of time to explore them and learning not to take them. This is commonly defined as *state-dependent actions*, but the current *TabularTDLearning.jl* does not support them.

To overcome this problem, we have pre-populated the Q-Table with large negative values for all the impossible combinations or illegal actions. We have chosen a large negative value to avoid any interference from the rest of the environment's reward structure. Thanks to this, the agent does not need to query the full environment, and as a consequence, computation times are significantly reduced.

In Chapter 6 we have quantitatively shown and analysed the impact of pre-populating the Q-Table. The proposed solution works well for problems with limited State-Spaces and Action-Spaces because computation times are reduced, which allows us to solve problems faster than without pre-populating the Q-Table. However, the problems that have been solved are not really close to reality.

7.2 FURTHER WORK

As evidenced, tabular methods must compute a value for each state, which becomes unfeasible with large state spaces.

Despite the fact that in this master thesis the *pre-populating the Q-Table approach* enabled us to increase the dimensionality of the problem, it is necessary to look for improvements for this method.

We have focused on tabular RL methods, particularly on Q-Learning algorithm. This could be further improved by using Q-Learning with Neural Networks, also known as the *Deep Q-Network (DQN) algorithm* [44].

DQN is a combination of Q-Learning and Deep Neural Networkss (DNNs), which are good approximators for non-linear functions. Therefore, it is not necessary to store the Q-Values in the Q-Table, as the DNNs approximate the Q-Function.

DQN uses two neural networks:

The main neural network represented by the weight vector θ and which estimates the Q-Values for the current state s and action a, $\mathrm{Q(s, a;}\theta)$.

The target neural network represented by the weight vector θ' and which estimates the Q-Values for the next state s' and action a'.

Both networks have the same structures but different weights. The learning process occurs in the main network while the target network‘s parameters remain unchanged for a fixed number of iterations and later the main neural network‘s weights are copied into the target neural network. By doing this, the target neural network‘s estimations are less inexact.

There is also the loss function, which can be computed as the squared difference between the target and predicted value. The goal will be to minimise the loss by updating the weight. This is computed as follows:

$$Loss = \left(r + \gamma \max_{a'} Q\left(s', a'; \theta'\right) - Q(s, a; \theta)\right)^2 \tag{7.1}$$

To understand why we propose to use DQN, it is important to consider that when using Q-

Learning only one Q-Value is produced at each time step. Nevertheless, neural networks produce a Q-Value for each of the possible actions as it is shown in Figure 7.1.

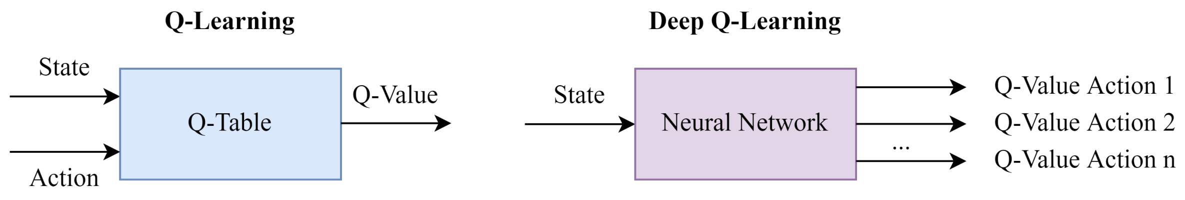

Figure 7.1: Schematic comparison of Q-Learning and a Deep Q-Learning method. Source: Own elaboration.

As a consequence, this approach increases the learning speed because neural networks might independently learn patterns that characterise the state.

To conclude this master thesis, we can state that the described research establishes a solid framework for further development of the problem and opens up many possibilities for future work. However, in order to solve more realistic problems, this must be improved.

We propose:

1. The use of state-dependent actions to prevent the agent from taking illegal actions.
2. If real information about the demand could be obtained, we would create a model more adjusted to reality instead of considering discrete uniform distribution.
3. The application of Deep RL. Nevertheless, it is commonly known that this algorithms might require a big quantity of data to reach a reasonable performance but even then, it is not guaranteed that they converge on the optimal value function. So, the next step to continue with this project would be to properly apply the DQN algorithm for its resolution.

Appendices

APPENDIX A

Resulting Policies

Simulations are done using the package *POMDPSimulators.jl*, which contains simulators for *POMDPs.jl*.

A.1 2 LDS

A.1.1 2 LDs with 100% capacity at the CD

A.1.1.1 2 LDs with 100% capacity at the CD 3 periods

The resulting policies are:

Q-Learning

In State (24, [0, 0], 3) takes Action ([12, 12]) ends up in State (0, [10, 8], 2) with a cost of 2.

In State (0, [10, 8], 2) takes Action ([0, 0]) ends up in State (0, [8, 6], 1) with a cost of 0.

In State (0, [8, 6], 1) takes Action ([0, 0]) ends up in State (0, [7, 6], 0) with a cost of 0.

SARSA

In State (24, [0, 0], 3) takes Action ([12, 12]) ends up in State (0, [9, 12], 2) with a cost of 2.

In State (0, [9, 12], 2) takes Action ([0, 0]) ends up in State (0, [6, 10], 1) with a cost of 0.

In State (0, [6, 10], 1) takes Action ([0, 0]) ends up in State (0, [6, 8], 0) with a cost of 0.

A.1.1.2 2 LDs with 100% capacity at the CD 10 periods

Dummy State Approach

In State (80, [0, 0], 10) takes Action ([8, 8]) ends up in State (64, [6, 4], 9) with a cost of 2.

In State (64, [6, 4], 9) takes Action ([0, 0]) ends up in State (64, [3, 0], 8) with a cost of 0.

In State (64, [3, 0], 8) takes Action ([5, 8]) ends up in State (51, [7, 5], 7) with a cost of 2.

In State (51, [7, 5], 7) takes Action ([0, 0]) ends up in State (51, [5, 1], 6) with a cost of 0.

In State (51, [5, 1], 6) takes Action ([0, 7]) ends up in State (44, [2, 8], 5) with a cost of 1.

In State (44, [2, 8], 5) takes Action ([6, 0]) ends up in State (38, [7, 7], 4) with a cost of 1.

In State (38, [7, 7], 4) takes Action ([0, 0]) ends up in State (38, [7, 5], 3) with a cost of 0.

In State (38, [7, 5], 3) takes Action ([0, 0]) ends up in State (38, [6, 5], 2) with a cost of 0.

In State (38, [6, 5], 2) takes Action ([0, 0]) ends up in State (38, [3, 4], 1) with a cost of 0.

In State (38, [3, 4], 1) takes Action ([1, 0]) ends up in State (37, [3, 2], 0) with a cost of 1.

Same State Approach

In State (80, [0, 0], 10) takes Action ([8, 8]) ends up in State (64, [5, 4], 9) with a cost of 2.

In State (64, [5, 4], 9) takes Action ([0, 0]) ends up in State (64, [1, 0], 8) with a cost of 0.

In State (64, [1, 0], 8) takes Action ([7, 8]) ends up in State (49, [6, 4], 7) with a cost of 2.

In State (49, [6, 4], 7) takes Action ([2, 0]) ends up in State (47, [8, 3], 6) with a cost of 1.

In State (47, [8, 3], 6) takes Action ([0, 1]) ends up in State (46, [6, 1], 5) with a cost of 1.

In State (46, [6, 1], 5) takes Action ([0, 7]) ends up in State (39, [5, 6], 4) with a cost of 1.

In State (39, [5, 6], 4) takes Action ([0, 0]) ends up in State (39, [3, 3], 3) with a cost of 0.

In State (39, [3, 3], 3) takes Action ([5, 5]) ends up in State (29, [4, 5], 2) with a cost of 2.

In State (29, [4, 5], 2) takes Action ([0, 0]) ends up in State (29, [2, 3], 1) with a cost of 0.

In State (29, [2, 3], 1) takes Action ([5, 1]) ends up in State (23, [5, 3], 0) with a cost of 2.

A.1.2 2 LDs with 90%, 80%, 70% an 60% capacity at the CD

A.1.2.1 2 LDs with 90% capacity at the CD 3 periods

Simulation 1

In State (21, [0, 0], 3) takes Action ([10, 10]) ends up in State (1, [9, 7], 2) with a cost of 2.

In State (1, [9, 7], 2) takes Action ([0, 0]) ends up in State (1, [7, 6], 1) with a cost of 0.

In State (1, [7, 6], 1) takes Action ([0, 0]) ends up in State (1, [3, 5], 0) with a cost of 0.

Simulation 2

In State (21, [0, 0], 3) takes Action ([10, 10]) ends up in State (1, [6, 9], 2) with a cost of 2.

In State (1, [6, 9], 2) takes Action ([0, 0]) ends up in State (1, [3, 6], 1) with a cost of 0.

In State (1, [3, 6], 1) takes Action ([1, 0]) ends up in State (0, [1, 6], 0) with a cost of 1.

Simulation 3

In State (21, [0, 0], 3) takes Action ([10, 10]) ends up in State (1, [6, 8], 2) with a cost of 2.

In State (1, [6, 8], 2) takes Action ([0, 0]) ends up in State (1, [2, 4], 1) with a cost of 0.

In State (1, [2, 4], 1) takes Action ([1, 0]) ends up in State (0, [0, 2], 0) with a cost of 11.

A.1.2.2 2 LDs with 80% capacity at the CD 3 periods

Simulation 1

In State (19, [0, 0], 3) takes Action ([9, 9]) ends up in State (1, [5, 7], 2) with a cost of 2.

In State (1, [5, 7], 2) takes Action ([1, 0]) ends up in State (0, [2, 3], 1) with a cost of 1.

In State (0, [2, 3], 1) takes Action ([0, 0]) ends up in State (0, [0, 0], 0) with a cost of 20.

Simulation 2

In State (19, [0, 0], 3) takes Action ([9, 9]) ends up in State (1, [7, 7], 2) with a cost of 2.

In State (1, [7, 7], 2) takes Action ([0, 0]) ends up in State (1, [3, 4], 1) with a cost of 0.

In State (1, [3, 4], 1) takes Action ([1, 0]) ends up in State (0, [2, 1], 0) with a cost of 1.

Simulation 3

In State (19, [0, 0], 3) takes Action ([9, 9]) ends up in State (1, [7, 6], 2) with a cost of 2.

In State (1, [7, 6], 2) takes Action ([0, 0]) ends up in State (1, [4, 6], 1) with a cost of 0.

In State (1, [4, 6], 1) takes Action ([0, 0]) ends up in State (1, [4, 2], 0) with a cost of 0.

A.1.2.3 2 LDs with 70% capacity at the CD 3 periods

Simulation 1

In State (16, [0, 0], 3) takes Action ([8, 8]) ends up in State (0, [6, 7], 2) with a cost of 2.

In State (0, [6, 7], 2) takes Action ([0, 0]) ends up in State (0, [6, 6], 1) with a cost of 0.

In State (0, [6, 6], 1) takes Action ([0, 0]) ends up in State (0, [2, 2], 0) with a cost of 0.

Simulation 2

In State (16, [0, 0], 3) takes Action ([8, 8]) ends up in State (0, [4, 5], 2) with a cost of 2.

In State (0, [4, 5], 2) takes Action ([0, 0]) ends up in State (0, [2, 5], 1) with a cost of 0.

In State (0, [2, 5], 1) takes Action ([0, 0]) ends up in State (0, [0, 4], 0) with a cost of 10.

Simulation 3

In State (16, [0, 0], 3) takes Action ([8, 8]) ends up in State (0, [6, 4], 2) with a cost of 2.

In State (0, [6, 4], 2) takes Action ([0, 0]) ends up in State (0, [3, 1], 1) with a cost of 0.

In State (0, [3, 1], 1) takes Action ([0, 0]) ends up in State (0, [0, 0], 0) with a cost of 30.

A.1.2.4 2 LDs with 60% capacity at the CD 3 periods

Simulation 1

In State (14, [0, 0], 3) takes Action ([6, 6]) ends up in State (2, [4, 5], 2) with a cost of 2.

In State (2, [4, 5], 2) takes Action ([1, 0]) ends up in State (1, [3, 3], 1) with a cost of 1.

In State (1, [3, 3], 1) takes Action ([0, 0]) ends up in State (0, [1, 1], 0) with a cost of 1.

Simulation 2

In State (14, [0, 0], 3) takes Action ([6, 6]) ends up in State (2, [5, 2], 2) with a cost of 2.

In State (2, [5, 2], 2) takes Action ([0, 2]) ends up in State (0, [1, 2], 1) with a cost of 1.

In State (0, [1, 2], 1) takes Action ([0, 0]) ends up in State (0, [1, 0], 0) with a cost of 20.

Simulation 3

In State (14, [0, 0], 3) takes Action ([6, 6]) ends up in State (2, [3, 2], 2) with a cost of 2.

In State (2, [3, 2], 2) takes Action ([0, 2]) ends up in State (0, [0, 1], 1) with a cost of 0.

In State (0, [0, 1], 1) takes Action ([0, 0]) ends up in State (0, [0, 0], 0) with a cost of 40.

A.1.2.5 2 LDs with 90% capacity at the CD 10 periods

In State (72, [0, 0], 10) takes Action ([8, 8]) ends up in State (56, [4, 4], 9) with a cost of 2.

In State (56, [4, 4], 9) takes Action ([0, 0]) ends up in State (56, [3, 3], 8) with a cost of 0.

In State (56, [3, 3], 8) takes Action ([5, 5]) ends up in State (46, [6, 5], 7) with a cost of 2.

In State (46, [6, 5], 7) takes Action ([0, 0]) ends up in State (46, [4, 3], 6) with a cost of 0.

In State (46, [4, 3], 6) takes Action ([4, 5]) ends up in State (37, [5, 7], 5) with a cost of 2.

In State (37, [5, 7], 5) takes Action ([0, 0]) ends up in State (37, [4, 6], 4) with a cost of 0.

In State (37, [4, 6], 4) takes Action ([0, 0]) ends up in State (37, [1, 2], 3) with a cost of 0.

In State (37, [1, 2], 3) takes Action ([7, 6]) ends up in State (24, [8, 5], 2) with a cost of 2.

In State (24, [8, 5], 2) takes Action ([0, 0]) ends up in State (24, [5, 1], 1) with a cost of 0.

In State (24, [5, 1], 1) takes Action ([0, 3]) ends up in State (21, [3, 2], 0) with a cost of 1.

A.1.2.6 2 LDs with 80% capacity at the CD 10 periods

In State (64, [0, 0], 10) takes Action ([8, 8]) ends up in State (48, [8, 6], 9) with a cost of 2.

In State (48, [8, 6], 9) takes Action ([0, 0]) ends up in State (48, [6, 2], 8) with a cost of 0.

In State (48, [6, 2], 8) takes Action ([0, 6]) ends up in State (42, [4, 6], 7) with a cost of 1.

In State (42, [4, 6], 7) takes Action ([0, 0]) ends up in State (42, [1, 6], 6) with a cost of 0.

In State (42, [1, 6], 6) takes Action ([7, 0]) ends up in State (35, [7, 4], 5) with a cost of 1.

In State (35, [7, 4], 5) takes Action ([0, 0]) ends up in State (35, [4, 3], 4) with a cost of 0.

In State (35, [4, 3], 4) takes Action ([0, 5]) ends up in State (30, [2, 6], 3) with a cost of 1.

In State (30, [2, 6], 3) takes Action ([6, 0]) ends up in State (24, [6, 5], 2) with a cost of 1.

In State (24, [6, 5], 2) takes Action ([0, 0]) ends up in State (24, [4, 1], 1) with a cost of 0.

In State (24, [4, 1], 1) takes Action ([0, 3]) ends up in State (21, [0, 3], 0) with a cost of 1.

A.1.2.7 2 LDs with 70% capacity at the CD 10 periods

In State (56, [0, 0], 10) takes Action ([8, 8]) ends up in State (40, [7, 4], 9) with a cost of 2.

In State (40, [7, 4], 9) takes Action ([0, 0]) ends up in State (40, [5, 0], 8) with a cost of 0.

In State (40, [5, 0], 8) takes Action ([0, 8]) ends up in State (32, [2, 7], 7) with a cost of 1.

In State (32, [2, 7], 7) takes Action ([6, 0]) ends up in State (26, [7, 6], 6) with a cost of 1.

In State (26, [7, 6], 6) takes Action ([0, 0]) ends up in State (26, [7, 5], 5) with a cost of 0.

In State (26, [7, 5], 5) takes Action ([0, 0]) ends up in State (26, [5, 2], 4) with a cost of 0.

In State (26, [5, 2], 4) takes Action ([0, 6]) ends up in State (20, [1, 4], 3) with a cost of 1.

In State (20, [1, 4], 3) takes Action ([7, 0]) ends up in State (13, [8, 2], 2) with a cost of 1.

In State (13, [8, 2], 2) takes Action ([0, 6]) ends up in State (7, [6, 8], 1) with a cost of 1.

In State (7, [6, 8], 1) takes Action ([0, 0]) ends up in State (7, [4, 6], 0) with a cost of 0.

A.1.2.8 2 LDs with 60% capacity at the CD 10 periods

In State (48, [0, 0], 10) takes Action ([8, 8]) ends up in State (32, [6, 4], 9) with a cost of 2.

In State (32, [6, 4], 9) takes Action ([0, 0]) ends up in State (32, [2, 2], 8) with a cost of 0.

In State (32, [2, 2], 8) takes Action ([5, 6]) ends up in State (21, [3, 4], 7) with a cost of 2.

In State (21, [3, 4], 7) takes Action ([2, 4]) ends up in State (15, [4, 4], 6) with a cost of 2.

In State (15, [4, 4], 6) takes Action ([3, 4]) ends up in State (8, [5, 5], 5) with a cost of 2.

In State (8, [5, 5], 5) takes Action ([2, 1]) ends up in State (5, [4, 4], 4) with a cost of 2.

In State (5, [4, 4], 4) takes Action ([2, 3]) ends up in State (0, [6, 7], 3) with a cost of 2.

In State (0, [6, 7], 3) takes Action ([0, 0]) ends up in State (0, [2, 4], 2) with a cost of 0.

In State (0, [2, 4], 2) takes Action ([0, 0]) ends up in State (0, [0, 2], 1) with a cost of 20.

In State (0, [0, 2], 1) takes Action ([0, 0]) ends up in State (0, [0, 1], 0) with a cost of 30.

A.2 3 LDS

A.2.1 3 LDs with 100% capacity at the CD 3 periods

In State (36, [0, 0, 0], 3) takes Action ([12, 12, 12]) ends up in State (0, [8, 10, 11], 2) with a cost of 3.

In State (0, [8, 10, 11], 2) takes Action ([0, 0, 0]) ends up in State (0, [6, 7, 8], 1) with a cost of 0.

In State (0, [6, 7, 8], 1) takes Action ([0, 0, 0]) ends up in State (0, [3, 3, 4], 0) with a cost of 0.

A.2.2 3 LDs with 80% and 70% capacity at the CD 5 periods

A.2.2.1 3 LDs with 80% capacity at the CD 5 periods

In State (36, [0, 0, 0], 5) takes Action ([7, 9, 8]) ends up in State (12, [7, 9, 5], 4) with a cost of 3.

In State (12, [7, 9, 5], 4) takes Action ([0, 0, 0]) ends up in State (12, [4, 8, 5], 3) with a cost of 0.

In State (12, [4, 8, 5], 3) takes Action ([0, 0, 0]) ends up in State (12, [2, 6, 5], 2) with a cost of 0.

In State (12, [2, 6, 5], 2) takes Action ([4, 0, 0]) ends up in State (8, [4, 3, 2], 1) with a cost of 1.

In State (8, [4, 3, 2], 1) takes Action ([0, 0, 1]) ends up in State (7, [2, 0, 3], 0) with a cost of 1.

A.2.2.2 3 LDs with 70% capacity at the CD 5 periods

In State (31, [0, 0, 0], 5) takes Action ([5, 4, 8]) ends up in State (14, [5, 2, 7], 4) with a cost of 3.

In State (14, [5, 2, 7], 4) takes Action ([0, 7, 0]) ends up in State (7, [4, 7, 5], 3) with a cost of 1.

In State (7, [4, 7, 5], 3) takes Action ([0, 0, 0]) ends up in State (7, [1, 5, 2], 2) with a cost of 0.

In State (7, [1, 5, 2], 2) takes Action ([2, 0, 3]) ends up in State (2, [0, 5, 2], 1) with a cost of 2.

In State (2, [0, 5, 2], 1) takes Action ([2, 0, 0]) ends up in State (0, [0, 2, 2], 0) with a cost of 11.

A.2.3 3 LDs with 80% and 70% capacity at the CD 7 periods

A.2.3.1 3 LDs with 80% capacity at the CD 7 periods

In State (67, [0, 0, 0], 7) takes Action ([8, 8, 8]) ends up in State (43, [7, 5, 6], 6) with a cost of 3.

In State (43, [7, 5, 6], 6) takes Action ([0, 0, 0]) ends up in State (43, [3, 2, 4], 5) with a cost of 0.

In State (43, [3, 2, 4], 5) takes Action ([5, 6, 0]) ends up in State (32, [6, 7, 1], 4) with a cost of 2.

In State (32, [6, 7, 1], 4) takes Action ([0, 0, 7]) ends up in State (25, [5, 4, 7], 3) with a cost of 1.

In State (25, [5, 4, 7], 3) takes Action ([0, 0, 0]) ends up in State (25, [1, 2, 4], 2) with a cost of 0.

In State (25, [1, 2, 4], 2) takes Action ([7, 6, 0]) ends up in State (12, [6, 5, 2], 1) with a cost of 2.

In State (12, [6, 5, 2], 1) takes Action ([0, 0, 2]) ends up in State (10, [2, 3, 2], 0) with a cost of 1.

A.2.3.2 3 LDs with 70% capacity at the CD 7 periods

In State (58, [0, 0, 0], 7) takes Action ([8, 8, 8]) ends up in State (34, [6, 6, 7], 6) with a cost of 3.

In State (34, [6, 6, 7], 6) takes Action ([0, 0, 0]) ends up in State (34, [3, 3, 3], 5) with a cost of 0.

In State (34, [3, 3, 3], 5) takes Action ([5, 5, 5]) ends up in State (19, [7, 5, 6], 4) with a cost of 3.

In State (19, [7, 5, 6], 4) takes Action ([0, 0, 0]) ends up in State (19, [4, 3, 3], 3) with a cost of 0.

In State (19, [4, 3, 3], 3) takes Action ([0, 5, 5]) ends up in State (9, [1, 6, 5], 2) with a cost of 2.

In State (9, [1, 6, 5], 2) takes Action ([7, 0, 0]) ends up in State (2, [6, 3, 2], 1) with a cost of 1.

In State (2, [6, 3, 2], 1) takes Action ([0, 0, 2]) ends up in State (0, [4, 0, 2], 0) with a cost of 11.

APPENDIX B

INITIAL TEST

B.1 DISCOUNT FACTOR

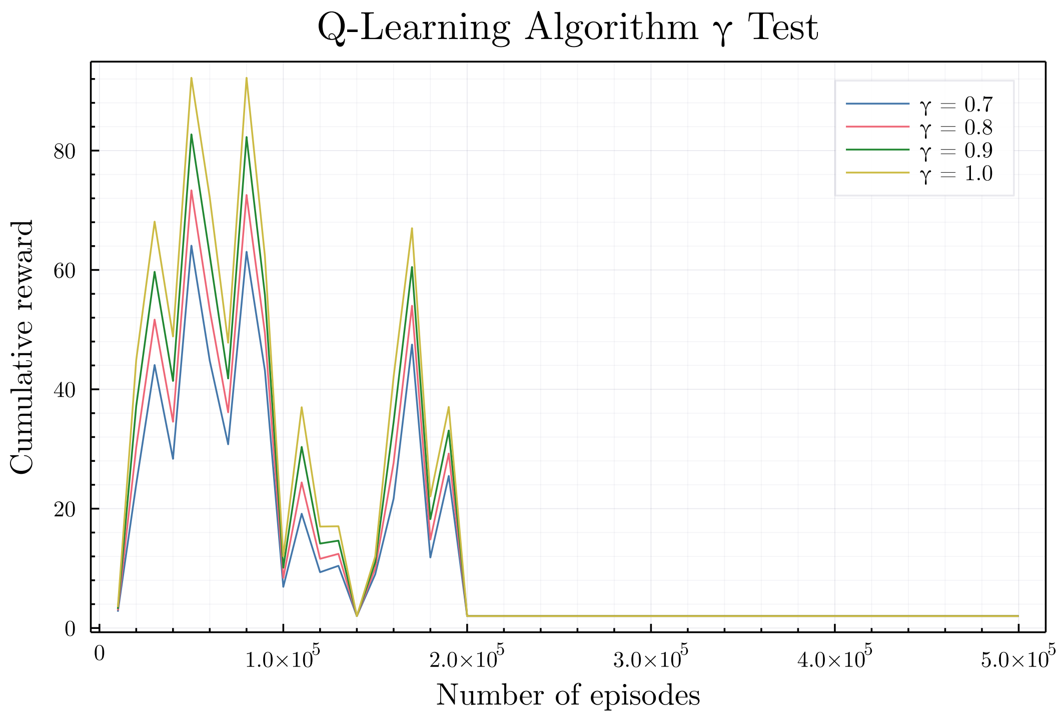

Figure B.1: Defining the value of γ with Q-Learning algorithm for 2 LDs with 100% capacity at the CD and 3 periods. Source: Own elaboration.

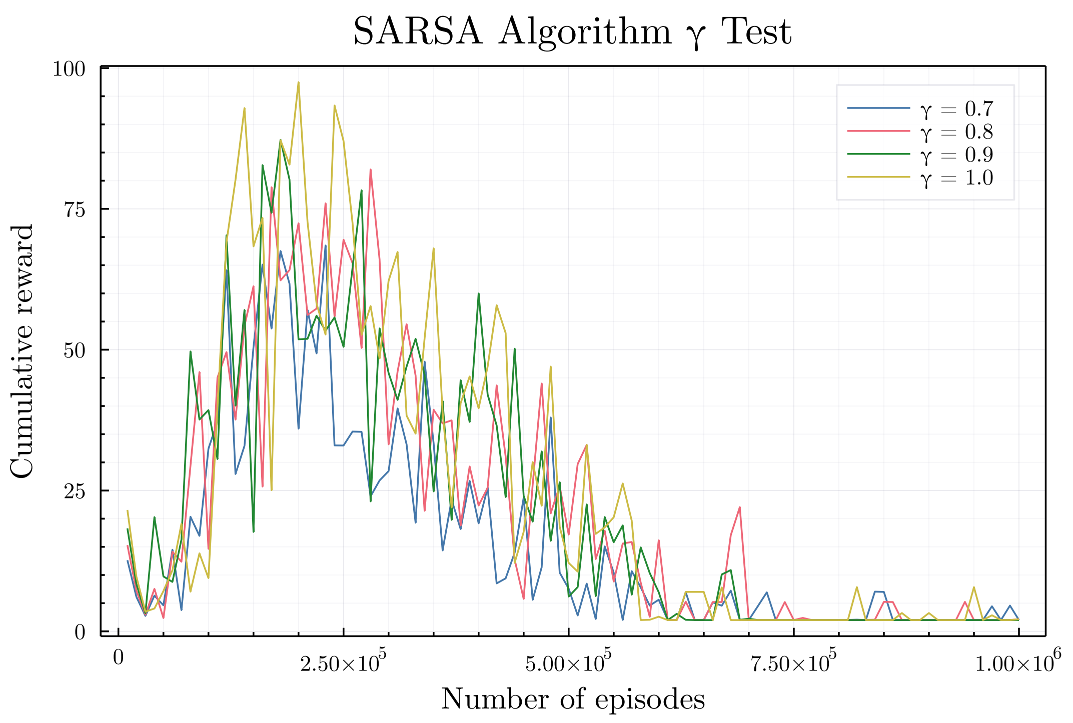

Figure B.2: Defining the value of γ with SARSA algorithm for 2 LDs with 100% capacity at the CD and 3 periods. Source: Own elaboration.

Algorithm	**Discount Factor γ**	**Computation Time (sec)**	**No. Episodes**
Q-Learning	0.7	27.23	200,000
Q-Learning	0.8	26.87	200,000
Q-Learning	0.9	25.65	200,000
Q-Learning	1.0	27.97	200,000
SARSA	0.7	48.59	-
SARSA	0.8	44.57	-
SARSA	0.9	43.14	710,000
SARSA	1.0	53.46	-

Table B.1: Results for Q-Learning and SARSA algorithms in the initial test for defining the value of the discount factor.

From Figures B.1 and B.2 and Table B.1, we see that the best results are those obtained for a value of $\gamma = 0.9$.

When using the Q-Learning algorithm ($\alpha = 0.10$ and $\varepsilon = 0.01$), Figure B.1 shows similar results and number of convergence episodes for all the γ values. Nevertheless, for $\gamma = 0.9$ computation times are smaller, which is of great importance when the dimensionality of the problem increases.

When using the SARSA algorithm ($\alpha = 0.001$ and $\varepsilon = \mathit{LinearDecay}$), from Figure B.2 we see that only when $\gamma = 0.9$ it converges.

APPENDIX C

CODE

C.1 MDP

```
#Load libraries, packages, solvers and policies
begin
    using StaticArrays, Random, Distributions
    using POMDPs, QuickPOMDPs, POMDPPolicies, POMDPModelTools
    using TabularTDLearning
end

#Parameters
cap = 1 #Percentage of total demand that the CD can serve (%)
N_lds = 3 #Number of LDs
N_period = 3 #Number of periods
gamma = 0.9 #Discount factor: preference for sooner rewards rather than later
dem = 4 #Maximum possible demand
CD_st = convert(Int,trunc(cap*dem*N_lds*N_period)) #Central depot stock, round down
LD_st = convert(Int,trunc(dem*N_period))#Maximum storage capacity of the LDs
CF = 1 #Fixed cost per shipment
penalty = 10 #For unsatisfied demand
impossible = -100 #For impossible states

#Define state data type
struct State
    cds::Int #Stock in the CD
    lds::SVector{N_lds, Int} #Stock in the LDs
    nper::Int #Number of periods
end

#Terminal states
terminal = [State(cds, lds, 0)
        for lds in [Iterators.product(ntuple(_ -> 0:LD_st, N_lds)...)...]
        for cds in 0:(CD_st - sum(lds))]
```

```
#Non terminal states
non_terminal = [State(cds, lds, nper)
        for lds in [Iterators.product(ntuple(_ -> 0:LD_st, N_lds)...)...]
        for cds in 0:(CD_st - sum(lds))
        for nper in (N_period-1):-1:1]


#Initial state
init_resupply  = [State(CD_st,lds,N_period)
        for lds in [Iterators.product(ntuple(_ -> 0:0, N_lds)...)...]]

#Dummy state for impossible actions
spt  = [State(-1,lds,-1)
        for lds in [Iterators.product(ntuple(_ -> -1:-1, N_lds)...)...]]

#Concatenate terminal and dummy states as terminal
terminal = vcat(terminal,spt)

#Define State space = concatenate terminal, non_terminal and init_resupply
SS = vcat(terminal, non_terminal, init_resupply)

#Define Action data type
struct Action
    ship::SVector{N_lds, Int} #Shipping quantity to the LDs
end

#Define Actions space
AS = [(Action(ship)) for ship in [Iterators.product(ntuple(_ -> 0:LD_st, N_lds)...)...]]

#Demand: Discrete uniform distribution
demand_vect = [((dem)) for dem in [Iterators.product(ntuple(_ -> -dem:0, N_lds)...)...]]

#Termination condition: Dummy or terminal state
termination(s::State) =  s in terminal

#Seed
Random.seed!(1)

#Define MDP using QuickPOMDPs.jl
CT_MDP = QuickMDP(
    gen = function (s, a, rnd)
        nps = State(s.cds - sum(a.ship), s.lds .+ a.ship , s.nper - 1) #Fictitious next
        ↪  possible state not considering demand
        if nps in SS #Feasibility test
            demand = rand(demand_vect) #Select a random demand because they are equally
            ↪  likely to happen
                sp = State(nps.cds , nps.lds .+ demand , nps.nper ) #Next possible state
                ↪  considering demand
                if sp in SS   #Check if there is no unsatisfied demand
                    r = -(CF*count(i->(i>0), a.ship)) #Cost per shipment with satisfied
                    ↪  demand
                    sp = sp #Next state
                else
                    udv = State(nps.cds, clamp.(sp.lds, ntuple(_ -> 0, N_lds), ntuple(_ ->
                    ↪  LD_st, N_lds)), nps.nper ) #Unsatisfied demand vector, non negative
                    ↪  elements
                    r = (-sum(udv.lds .-sp.lds))*penalty -(CF*count(i->(i>0), a.ship)) #Cost
                    ↪  per shipment + unsatisfied demand
                    sp = udv  #Next state
                end
        else
```

```
            r = impossible #Gets bad reward for impossible action
            sp = spt[1] #Send the agent to the dummy state
        end
        return (sp=sp, r=r) #Tuple containing the next state and reward
    end,
    states = SS, #State-space
    actions = AS, #Action-space
    initialstate = init_resupply, #Initial state
    discount = gamma, #Discount factor
    isterminal = termination) #Termination: Dummy or terminal state
```

C.2 PRE-POPULATING THE Q-TABLE

```
#Function that given a state returns the possible actions the agent can take from there
function A(s::State)
    return  [(Action(ship)) for ship in Iterators.product(ntuple((i ->
    ↪  0:LD_st.-s.lds[i]),N_lds)...) if sum(ship)<=s.cds]
end

#Set all zeros at the Q-Table
q_learning_solver.Q_vals = zeros(length(states(CT_MDP)), length(actions(CT_MDP)))

#Set bad rewards for impossible actions at the Q-Table
@time for s in states(CT_MDP)
    valid_actions = A(s)
    for a in actions(CT_MDP)
        if !(a in valid_actions)
            q_learning_solver.Q_vals[stateindex(CT_MDP, s), actionindex(CT_MDP, a)] = -9999
        end
    end
end

println(count(i->(i== -9999), q_learning_solver.Q_vals)) #Count the number of impossible
↪  combinations
```

C.3 SOLVERS

C.3.1 Q-Learning

```
#Using Q-Learning
q_learning_solver = QLearningSolver(n_episodes = 500000, #Number of episodes to train the
↪  Q-Table
                        max_episode_length = N_period, #Maximum number of steps before the
                        ↪  episode is ended
                        learning_rate = 0.1, #Learning rate
                        exploration_policy = EpsGreedyPolicy(CT_MDP,0.01), #Exploration
                        ↪  policy
                        eval_every = 10000, #Frequency at which to evaluate the trained
                        ↪  policy
                        n_eval_traj = 20, #Number of episodes to evaluate the policy
                        verbose = true) #print information during training

@time q_learning_policy = solve(q_learning_solver, CT_MDP)
```

C.3.2 SARSA

```
#Using SARSA
sarsa_solver = SARSASolver(n_episodes = 500000, #Number of episodes to train the Q-Table
                        max_episode_length =  N_period, #Maximum number of steps before the
                        ↪  episode is ended
                        learning_rate = 0.001,  #Learning rate
                        exploration_policy= EpsGreedyPolicy(CT_MDP,0.01), #Exploration policy
                        eval_every =  10000, #Frequency at which to evaluate the trained
                        ↪  policy
                        n_eval_traj = 20, #Number of episodes to evaluate the policy
                            verbose = true) #print information during training

@time sarsa_policy = solve(sarsa_solver, CT_MDP)
```

C.4 SIMULATORS

```
using POMDPSimulators

#Rollout Simulator to find the accumulated discounted reward from a single simulated
↪  trajectory
rs = RolloutSimulator()
r = simulate(rs, CT_MDP, q_learning_policy)
r = simulate(rs, CT_MDP, sarsa_policy)


#HistoryRecorder runs a simulation and records the trajectory
hr = HistoryRecorder()
h = simulate(hr, CT_MDP, q_learning_policy)
h = simulate(hr, CT_MDP, sarsa_policy)


#Steppingthrough
for (s, a,  r) in stepthrough(CT_MDP,  q_learning_policy, "s,a,r")
    println("In  $s")
    println("took  $a")
    println("and received a reward of $r.")
end
```

Bibliography

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

[2] David J Spiegelhalter. Probabilistic prediction in patient management and clinical trials. *Statistics in medicine*, 5(5):421–433, 1986.

[3] Barbara Farrell, Sara Kenyon, and Haleema Shakur. Managing clinical trials. *Trials*, 11(1):1–6, 2010.

[4] Arindrima Datta. Markov chains and markov decision theory, 2014.

[5] Fathima Anjila PK. What is artificial intelligence? *"Success is no accident. It is hard work, perseverance, learning, studying, sacrifice and most of all, love of what you are doing or learning to do"*., page 65, 1984.

[6] Rusul Abduljabbar, Hussein Dia, Sohani Liyanage, and Saeed Asadi Bagloee. Applications of artificial intelligence in transport: An overview. *Sustainability*, 11(1):189, 2019.

[7] Mohammed Yousef Shaheen. Applications of artificial intelligence (ai) in healthcare: A review. *ScienceOpen Preprints*, 2021.

[8] Xieling Chen, Di Zou, Haoran Xie, Gary Cheng, and Caixia Liu. Two decades of artificial intelligence in education. *Educational Technology & Society*, 25(1):28–47, 2022.

[9] Michel Tokic. Adaptive ε-greedy exploration in reinforcement learning based on value differences. In *Annual Conference on Artificial Intelligence*, pages 203–210. Springer, 2010.

[10] Chitu Okoli and Kira Schabram. A guide to conducting a systematic literature review of information systems research. *Research Methods & Methodology in Accounting eJournal*, 2010.

[11] Catherine De Angelis, Jeffrey M Drazen, Frank A Frizelle, Charlotte Haug, John Hoey, Richard Horton, Sheldon Kotzin, Christine Laine, Ana Marusic, A John PM Overbeke,

Torben V Schroeder, Hal C Sox, and Martin B Van Der Weyden. Clinical trial registration: a statement from the international committee of medical journal editors. *The Lancet*, 364(9438):911–912, 2004.

[12] Mark P Lythgoe and Paul Middleton. Ongoing clinical trials for the management of the covid-19 pandemic. *Trends in pharmacological sciences*, 41(6):363–382, 2020.

[13] Mark P. Lythgoe, Christopher J. Rhodes, Pavandeep Ghataorhe, Mark Attard, John Wharton, and Martin R. Wilkins. Why drugs fail in clinical trials in pulmonary arterial hypertension, and strategies to succeed in the future. *Pharmacology & Therapeutics*, 164:195–203, 2016.

[14] David Francis, Ian Roberts, Diana R Elbourne, Haleema Shakur, Rosemary C Knight, Jo Garcia, Claire Snowdon, Vikki A Entwistle, Alison M McDonald, Adrian M Grant, et al. Marketing and clinical trials: a case study. *Trials*, 8(1):1–7, 2007.

[15] Jennifer M Tetzlaff, An-Wen Chan, Jessica Kitchen, Margaret Sampson, Andrea C Tricco, and David Moher. Guidelines for randomized clinical trial protocol content: a systematic review. *Systematic reviews*, 1(1):1–11, 2012.

[16] An-Wen Chan, Jennifer M Tetzlaff, Douglas G Altman, Andreas Laupacis, Peter C Gøtzsche, Karmela Krleža-Jerić, Asbjørn Hróbjartsson, Howard Mann, Kay Dickersin, Jesse A Berlin, et al. Spirit 2013 statement: defining standard protocol items for clinical trials. *Annals of internal medicine*, 158(3):200–207, 2013.

[17] René Haijema, Jan van der Wal, and Nico M van Dijk. Blood platelet production: Optimization by dynamic programming and simulation. *Computers & Operations Research*, 34(3):760–779, 2007.

[18] Ana R Vila-Parrish, Julie Simmons Ivy, and Russell E King. A simulation-based approach for inventory modeling of perishable pharmaceuticals. In *2008 winter simulation conference*, pages 1532–1538. IEEE, 2008.

[19] Vera Hemmelmayr, Karl F Doerner, Richard F Hartl, and Martin WP Savelsbergh. Vendor managed inventory for environments with stochastic product usage. *European Journal of Operational Research*, 202(3):686–695, 2010.

[20] Robert G Gallager. *Stochastic processes: theory for applications.* Cambridge University Press, 2013.

[21] Yuanguang Zhong, Zhichao Zheng, Mabel C Chou, and Chung-Piaw Teo. Resource pooling and allocation policies to deliver differentiated service. *Management Science*, 64(4):1555–1573, 2018.

[22] Novie Novie and Haryadi Sarjono. The determination of optimal inventory using markov chain. *Jurnal Manajemen Teori dan Terapan*, 6(1), 2013.

[23] Atefeh Baghalian, Shabnam Rezapour, and Reza Zanjirani Farahani. Robust supply chain network design with service level against disruptions and demand uncertainties: A real-life case. *European journal of operational research*, 227(1):199–215, 2013.

[24] Marcel A Sieke, Ralf W Seifert, and Ulrich W Thonemann. Designing service level contracts for supply chain coordination. *Production and Operations Management*, 21(4):698–714, 2012.

[25] Amanda J Schmitt. Strategies for customer service level protection under multi-echelon supply chain disruption risk. *Transportation Research Part B: Methodological*, 45(8):1266–1283, 2011.

[26] Guodong Lyu, Mabel C Chou, Chung-Piaw Teo, Zhichao Zheng, and Yuanguang Zhong. Stochastic knapsack revisited: The service level perspective. *Operations Research*, 2021.

[27] William D Smart and L Pack Kaelbling. Effective reinforcement learning for mobile robots. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No. 02CH37292)*, volume 4, pages 3404–3410. IEEE, 2002.

[28] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[29] Kun Shao, Zhentao Tang, Yuanheng Zhu, Nannan Li, and Dongbin Zhao. A survey of deep reinforcement learning in video games. *arXiv preprint arXiv:1912.10944*, 2019.

[30] Ahmad EL Sallab, Mohammed Abdou, Etienne Perot, and Senthil Yogamani. Deep reinforcement learning framework for autonomous driving. *Electronic Imaging*, 2017(19):70–76, 2017.

[31] Ian Rogers. The google pagerank algorithm and how it works. *http://ianrogers.uk/google-page-rank/*, 2002.

[32] A Agogino and Joydeep Ghosh. Increasing pagerank through reinforcement learning. *Proceedings of Intelligent Engineering Systems Through Artificial Neural Networks*, 12:27–32, 2002.

[33] Yuxi Li. Reinforcement learning applications. *arXiv preprint arXiv:1908.06973*, 2019.

[34] Zhongheng Zhang et al. Reinforcement learning in clinical medicine: a method to optimize dynamic treatment regime over time. *Annals of translational medicine*, 7(14), 2019.

[35] Chengzhi Jiang and Zhaohan Sheng. Case-based reinforcement learning for dynamic inventory control in a multi-agent supply-chain system. *Expert Systems with Applications*, 36(3):6520–6526, 2009.

[36] Ahmet Kara and Ibrahim Dogan. Reinforcement learning approaches for specifying ordering policies of perishable inventory systems. *Expert Systems with Applications*, 91:150–158, 2018.

[37] Ick-Hyun Kwon, Chang Ouk Kim, Jin Jun, and Jung Hoon Lee. Case-based myopic reinforcement learning for satisfying target service level in supply chain. *Expert Systems with Applications*, 35(1-2):389–397, 2008.

[38] Pierpaolo Pontrandolfo, Abhijit Gosavi, O Geoffrey Okogbaa, and Tapas K Das. Global supply chain management: a reinforcement learning approach. *International Journal of Production Research*, 40(6):1299–1317, 2002.

[39] Kaifeng Gao, Gang Mei, Francesco Piccialli, Salvatore Cuomo, Jingzhi Tu, and Zenan Huo. Julia language in machine learning: Algorithms, applications, and open issues. *Computer Science Review*, 37:100254, 2020.

[40] Zacharias Voulgaris. *Julia for Data Science.* Technics Publications, 2016.

[41] Maxim Egorov, Zachary N. Sunberg, Edward Balaban, Tim A. Wheeler, Jayesh K. Gupta, and Mykel J. Kochenderfer. POMDPs.jl: A framework for sequential decision making under uncertainty. *Journal of Machine Learning Research*, 18(26):1–5, 2017.

[42] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. *Thesis submitted for Ph.D*, 1989.

[43] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. Citeseer, 1994.

[44] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533, 2015.