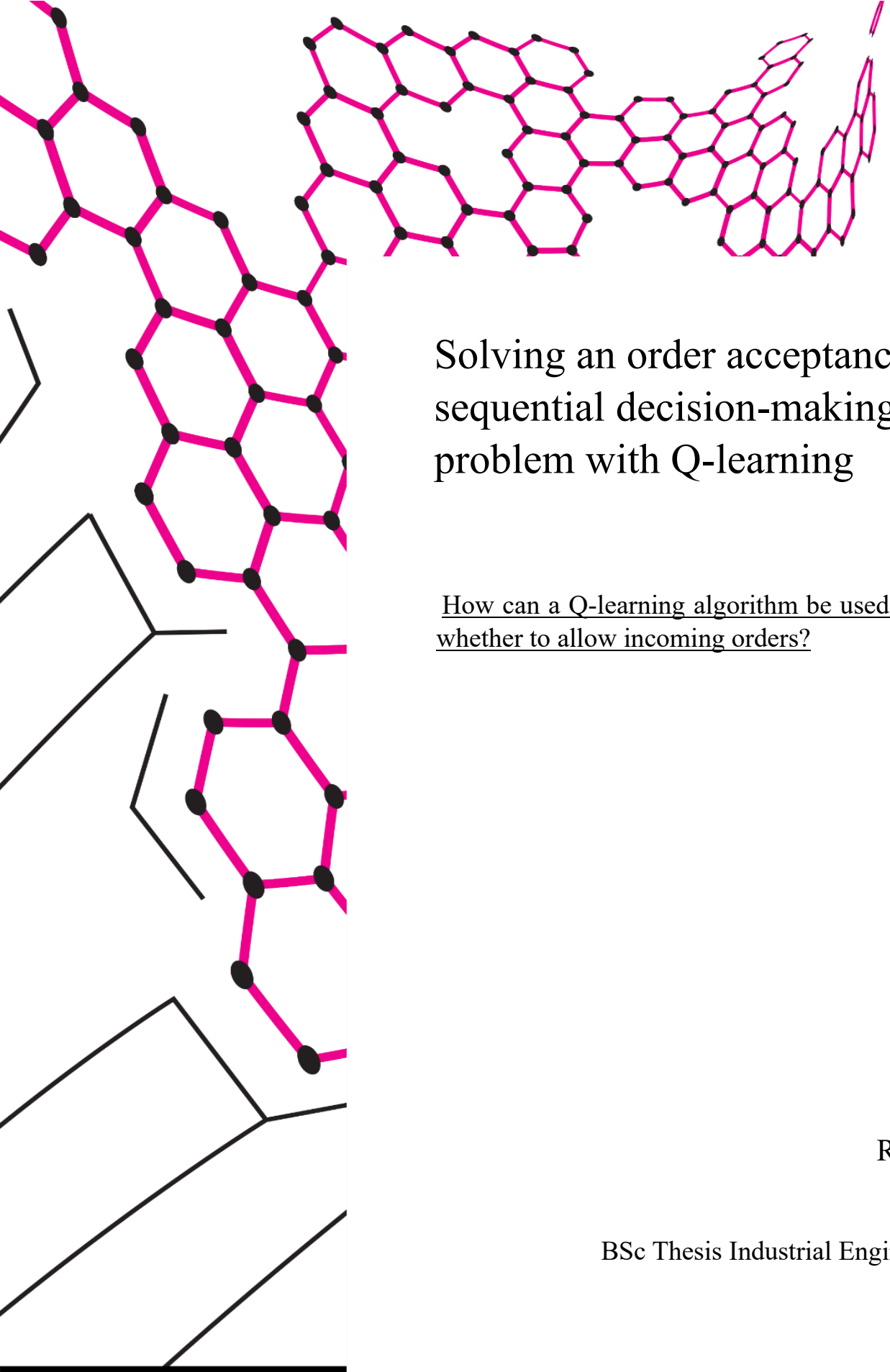

Solving an order acceptance sequential decision-making problem with Q-learning

How can a Q-learning algorithm be used for deciding whether to allow incoming orders?

Raul Calviño Sobrido

July 25, 2024

BSc Thesis Industrial Engineering & Management

UNIVERSITY OF TWENTE.

Colophon

FACULTY

Behavioral, Management, and social sciences

DATE

July 25, 2024

VERSION

Version 1

AUTHOR

Raul Calviño Sobrido, s2609029

SUPERVISORS

Stephan Meisel (first supervisor)

Alessio Trivella (second supervisor)

Martin Wölck (company supervisor)

EMAIL

r.calvinosobrado@student.utwente.nl

POSTAL ADDRESS

P.O. Box217

7500 AE Enschede

WEBSITE

www.utwente.nl

FILE NAME

Solving an order acceptance sequential decision-making problem with Q-learning

Acknowledgements

Dear reader,

In this paper, you will find the culmination of over five months of dedicated effort on what is undoubtedly the most challenging project I have ever undertaken. The completion of this research has been possible not only thanks to my efforts, but also thanks to several people that helped and supported me along the way.

For this, I would like to express my deepest appreciation to Stephan Meisel, my company supervisor. He first introduced me on the topics of dynamic programming and reinforcement learning. His belief in my ability to complete a bachelor's thesis on a topic I initially knew very little about was invaluable. He supported me throughout the entire process, always helping when needed. Additionally, I extend my sincere thanks to Alessio Trivella, my second supervisor, whose feedback greatly improved the quality of my thesis.

I'm also extremely thankful to Flaschenpost, the company providing me with the case, and all the support and resources to finalize my research. Specially, I am very thankful to my company supervisor, Martin Wölck, who offered unwavering guidance and patience, showing profound belief in my abilities.

Lastly, I would like to thank my friends and family, as they were the main source of emotional support during the most difficult periods of the research and accompanied me during the difficult late-night working sessions on the university library.

Raúl Calviño Sobrido

July 25, 2024

Management summary

In this thesis we will focus on the decision-making process for a real-time food delivery company: Flaschenpost. Through a day of operations, they decide whether customers can place orders on their website. The website can then be “open” or “closed” for orders.

Problem description

Deciding whether to open the website is critical for the company's operations. If it is open for too long, it may increase the number of orders arriving late, dropping customer satisfaction. On the contrary, a conservative strategy may lead to leaving potential customers on the table.

Flaschenpost currently uses a series of parameters to make this decision. These parameters involve information about the delivery driver utilization (How much are the drivers being used?) and the order status (How many orders are expected to arrive in time? Or, how many minutes late will an average order arrive?)

Through this research, we will explore this decision-making process utilizing a Q-learning agent, a reinforcement learning algorithm focused on obtaining the optimal decision at each point in time. The agent will learn what the consequences of each action (allowing/not allowing orders) are at a certain point in time, given certain values for the utilization parameters.

Solution method

To train the agent, we had to develop a simulation environment that replicates the behavior of Flaschenpost routing and the city environment. This learning environment replicates orders being placed across a city over a day. Besides, it also utilizes a simple routing algorithm to imitate how the actual routing of Flaschenpost works thus obtaining the decision-making parameters.

Once the learning environment was modeled, we trained the Q-learning agent by repeating the same day and adjusting the learning set-up to maximize the daily reward. This reward balances orders being accepted (positive reward) and orders being late or not delivered (negative reward).

Results

There are two main results extracted: the policy derived by the Q-learning agent itself, and how the Q-learning agent’s policy performs with respect to other options. Those results were obtained considering approximately 1000 orders were arriving per day, and we switched between 15 drivers and 10 drivers available.

For the 15-driver set-up, the policy keeps the store open for most of the day unless the drivers are highly utilized. The store is then mostly closed toward the end of the day, where it mostly

closes. For the 10-day set-up, the store is also open for most of the day (≈ 90% of the time), which leads to the orders punctuality dropping through the day,

We compared this policy with three other options: random decision, always open, and company-derived policy. The Q-learning policy performed better on the 15-driver set-up compared with the company-derived policy. This set-up was shown to allow most of the orders to be assigned, as the capacity was never a constraint. For the 10-driver set-up, the Q-policy performed slightly worse than the company-derived one. Despite that, across the testing, there were specific days where the results were higher. This means that, as of right now, the company's simple heuristic is better for more challenging scenarios, but the Q-learning algorithm showed a high learning potential.

Recommendations

Even though more testing could be done, the data shows that Q-learning is a suitable approach for deciding whether to allow/not allow customer orders and shows great learning potential. For that reason, we have the following suggestions on how to proceed:

- Considering the insights of a Q-policy, explore whether the current policy is too greedy and whether the store is closed too much time.
- Increase the complexity and accuracy of the Q-learning model, including more real-life parameters, or a more realistic demand simulator, based on the actual demand experienced.
- With the already developed mathematical model, expand the learning agent towards a more complex algorithm that can give more accurate results.

Table of contents

List of figures

List of tables

Table of abbreviations

Abbreviation	Meaning
ML	Machine Learning
MDP	Markov Decision Process
RL	Reinforcement Learning
FMCGs	Fast Moving Consumer Goods
MPSM	Managerial Problem-Solving Method
TD	Temporal Difference
NNA	Nearest-Neighbor Algorithm
NHPP	Non-Homogeneous Poisson Process

Summary of notation

Notation	Meaning
Markov decision process and reinforcement learning basics	
$s \in \mathcal{S}$	State belonging to the finite set $\mathcal{S}$ of possible states
$a \in \mathcal{A}(s)$	Action taken, belonging to the finite set of actions $\mathcal{A}(s)$ given state s
$T(s', s, a)$	Transition function given current state s, action a and future state s'
$R(s', s, a)$	Reward function given current state s, action a and future state s'
$t \in \{0,1, \dots, T\}$	Discrete time step of the agent-environment interaction
$\pi(a\|s)$	Policy determining probability of choosing action a given state s
$v_\pi(s)$	Value of state s under policy π
$q_\pi(s, \mathrm{a})$	Value of taking action a in state s under policy π
S_t	State value at stage t
R_t	Reward value at sage t
A_t	Action taken at stage t
$V(S_t)$	agent's estimate of $v(s)$ at a given stage t
$Q(S_t, A_t)$	agent's estimate of $\mathrm{q}(s, a)$ at a given stage t
π_b	Behavior policy
π_t	Target policy
R_{epsode}	Cumulative reward of an episode
Problem modeling	
$d \in D_t$	Driver belonging to the finite set of drivers D_t at t
$o \in O_t$	Order belonging to all active orders O_t to be fulfilled at t
$W(a, t)$	Number of orders arriving at t given action a was taken
r_a	Reward received for order being placed
r_l	Reward received for delivering an order late
r_n	Reward received for not delivering an order
$n(t)$	Function that returns the interval of a day a stage t is in
Implementation of the algorithm	
k	Area of the artificial city
$C_{k,t}$	R.V. representing the number of orders arriving at k at stage t
$\lambda_{k,t}$	Order arrival rate for area k at stage t
$\lambda_{k,p}$	Peak arrival rate for area k
$\lambda_{k,b}$	Base arrival rate for area k
b_k	Width of the demand peak for area k
$t_{k,p}$	Time t where peak demand is reached
$W_{k,t}$	Set of orders arriving from area k at stage t
$w_t(a)$	Set of orders arriving given an action a
$\varepsilon_{episode}$	Probability of exploration in an episode
ε_{max}	Maximum agent's exploration

ε_{mii}	Minimum agent's exploration
κ	Exploration decay through a run
α	Q-learning agent learning rate
γ	Q-learning agent discount factor
E	Number of episodes in a training

1 Introduction

In this chapter we will discuss all the relevant information regarding the context of my bachelor's theses, the problem definition and the overall methodology followed for solving the problem. Section 1.1 will contextualize the study, Section 1.2 introduces the problem at hand and Section 1.3 will include the methodology followed on this research.

1.1 Background and context for the study

1.1.1 Company description

Flaschenpost.de is a German real-time delivery company for beverages and food. Its headquarters are in Münster, and currently belongs to the German conglomerate Dr.Oetker group[1]. The company specializes in fast last-mile delivery to the customer, allowing to fulfill over 90% of their orders within 2 hours of being placed.

Currently, they have managed to expand across the entirety of Germany, offering their services in over 190 German cities. The size and speed of their service is obtained thanks to their over 30 warehouses nationwide, and over 20.000 employees across all departments. However, even more advantageous than their manpower is their routing algorithm, which greatly optimizes their last-mile logistics.

Flaschenpost is open for delivery to customers every day from 8:00 until 20:00. During this time, a customer can order beverages and food, deciding between an "immediate" delivery or a "pre-order". If immediate delivery is selected, the order will arrive to the customer within 2 hours of being placed. The customer can also place a "pre-order", where its order will arrive at some point in the following 3 days. The immediate order is not always available, as every 5 minutes the company may decide to block this option if the warehouse/delivery drivers are very busy. This means that no new orders will be allowed and thus the website will be closed.

1.1.2 Company context

Flaschenpost.de competes in the fast-moving consumer good (FMCGs) eCommerce market. Following a study by Statista (2023) the market has been on the rise since 2019, reaching an all-time high in 2023, with over 5.3 million Germans estimated to purchase groceries through the internet.

This fast-growing market has created great opportunities for Flaschenpost.de, positioning itself as the leading online food and beverage company in Germany, being acquired in 2020 by Dr. Oetker for an estimated 1 billion € (Hüfner, 2020). The acquisition also helped the

[1] The information about Flaschenpost comes from the "About us" section of their website https://www.flaschenpost.de/unternehmen/ueber-uns.

company keep growing, reaching a total estimated net sale of over 500 million € in 2023 (Statista, 2023).

Despite the success of the company, the fast-growing market has also created a wildly competitive environment, highlighting both rewe.de and amazon.de (Amazon fresh) as second and third market leaders by net sales (Statista, 2023). This increasing competition made Flaschenpost.de keep cutting-edge innovations and continuously improve its service to the customers.

1.2 Problem description

As it is right now, Flaschenpost is struggling to keep up with the demand, especially fulfilling orders during their peak hours. After a series of meetings with the company supervisors, we specified the core business values of the company regarding delivery. They want to maintain a high fulfillment rate. This means that the maximum proportion of demand is met and most of these orders are within the 2-hour window.

The company believes that the decision-making process for making available immediate orders could be improved. As of right now, this decision relies on the gut feeling of a company expert based on some driver utilization information, but there has never been an analytical exploration of the decision. This makes the company potentially lose sales as they close the online store when orders could still be processed (conservative decision), or don’t close it when they should (risky decision)..

1.2.1 Problem cluster

We will use a problem cluster to further expand on the issues and determine the core problem to be addressed (Heerkens et al., 2017). This tool will allow us to bring order to the problem context, by stablishing cause-effect relationships. We will detect the upstream problems that lead to losing sales and identify a core problem to be addressed. The problem cluster of this research can be seen in Figure 1-1.

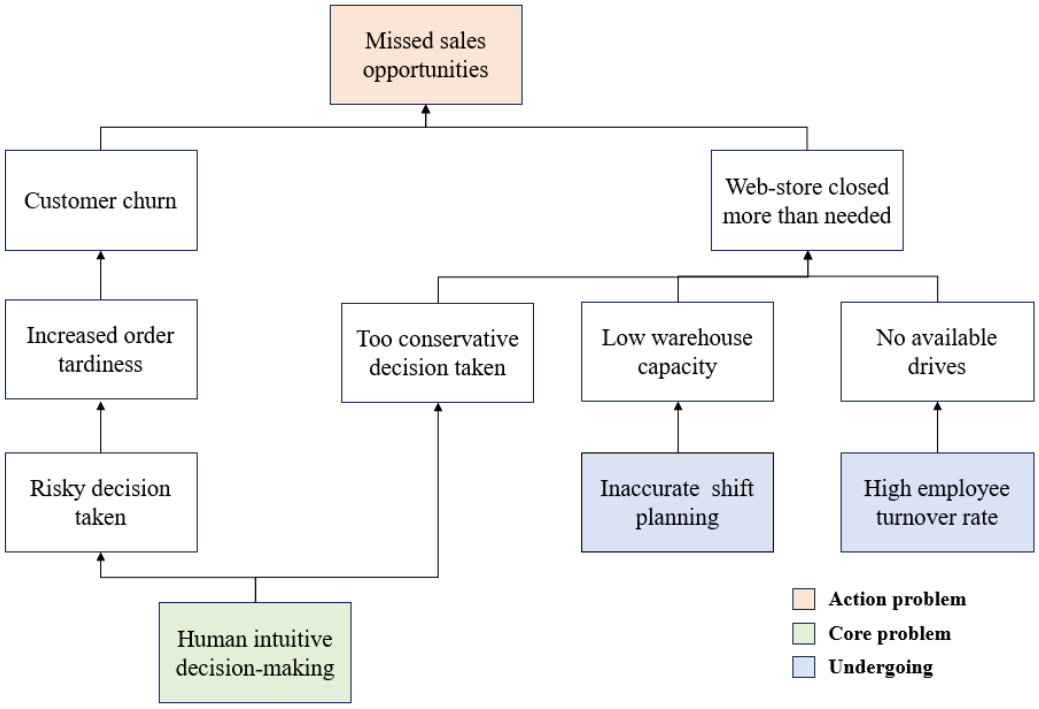

Figure 1-1 - Flaschenpost problem cluster.

We identified the action problem to be "Missed sales opportunities". As introduced during the problem description, as right now, there is a "leakage" of potential sales. This means that, on a given day, the number of fulfilled orders could be higher, which also leads to a more efficient utilization of the company resources.

The main two reasons identified for missing sales opportunities are: website closed and customer churn.

Regarding the closure of the website, every couple of minutes, a decision is made on whether delivering the orders immediately is possible. If the store is closed, the customers will then place an order for a future point in time or cancel the order completely. Thus, closing the store implies not allowing new incoming orders.

The customer churn refers to the percentage of customers that decide not to re-order after bad experiences with the company. A possible low customer churn derives from a lower customer satisfaction due a placed order arriving later than the promised 2-hour delivery period. In general, if the customer did not receive the order in time, it may decide not to place an order in the future, which may impact sales negatively.

We then identified the core problem of this research to be that Flaschenpost **makes decisions on opening-closing the store based on human intuition and gut feeling**. They do not have a systematic decision-making approach or proper assessment on how their current decisions (allowing or not new orders) will affect their future capacity, and thus take either too conservative or too risky decisions. Our task will be then to produce a decision-making policy

that considers how current decisions affect future scenarios, maximizing the number of orders fulfilled through a day.

1.2.2 Company problem as a sequential decision-making problem

On a basic level, this problem is a sequential decision-making process, as every 5 minutes the company takes a decision on whether to open/close the store. From Bertsekas (2007), sequential decision-making is then defined as a situation where the decision maker (in this case the company) makes successive decisions (opening or closing the website) across time (the day) based on some state (in this case the situation of the drivers and warehouse), with the goal of maximizing reward (orders fulfilled through the day).

Since the last couple of years, innovations in the fields of operations research and reinforcement learning (RL) have allowed more decision-making problems, such as the one at hand, to be modeled and solved using RL methods. This increasing trend led to the number of publications on the use of RL for supply chain and logistics has doubled over the last 4 years (Yan et al., 2022).

On a basic level, we then refer to RL as a subset of Machine Learning (ML) that uses a simulation environment to learn about what is the best decision at each point in time (Ławrynowicz & Tresp, 2014). A RL algorithm updates its knowledge of the environment through interaction (exploration of the data) to optimize the performance of a task.

RL has then been proven to be widely used in the fields of logistics and supply chain management. Across the different methodologies of reinforcement learning, we can highlight Q-learning as a widely used method for dealing with these types of problems. As seen in Figure 1-2, from an exploration of 114 publications on RL in supply chain and logistics, around 60% of them used the Q-learning methodology (Yan et al., 2022).

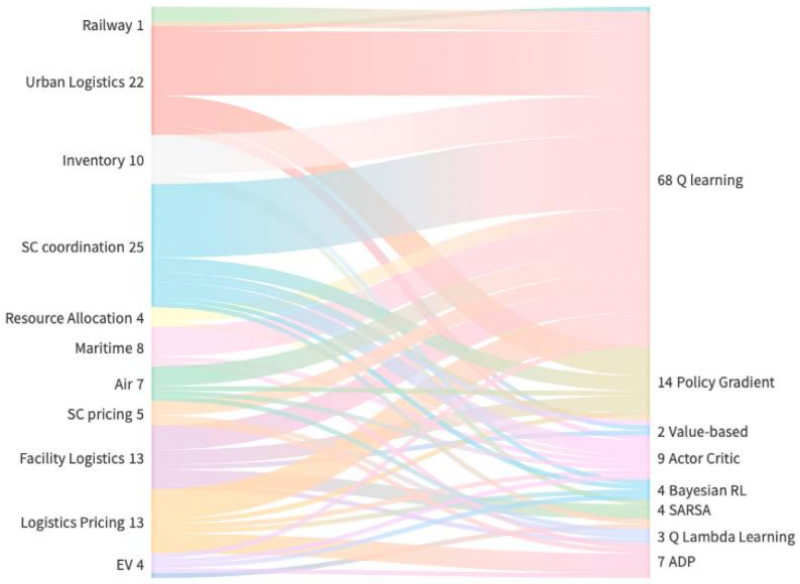

Figure 1-2 - Mapping of applications to RL training methods (Yan et al., 2022) .

Q-learning (Bellman, 1957) is a RL algorithm that focuses on understanding the value of each state the environment can be in across the different actions that can be taken. During Chapter 3 we will explore more in detail what sequential decision-making entails, how can it be modeled as a Markov Decision Process, and how specifically RL can be used for solving this kind of problems.

1.3 Methodology

Research approach.

To approaching this problem, we decided to use the managerial problem-solving method (MPSM), first introduced by Heerkens et al. (2017). This problem-solving method allows us to discern the real problem, and develop solutions in a sistematically, making sure that different perspectives are explored, and the effectiveness of the solution is evaluated. The MPSM is divided into the following phases:1.Defining the problem, 2.Formulating the approach, 3. Analyzing the problem, 4. Formulating (alternative) solutions, 5.Choosing a solution, 6. Implementing the solution, 8.Evaluation of the solution.

To implement MPSM, we will formulate different knowledge problems for each of its phases. Note that the phases of "defining the problem" and "formulating the approach" correspond to this proposal's problem identification and plan of approach. The knowledge problems will address the remaining phases of the MPSM.

Research question.

After analyzing the problem, the source, and the research approach, we can formulate the research question as follows:

How can a Q-learning algorithm be used for deciding whether to allow incoming orders?

We can also approach the research question as a norm vs reality problem, where the current reality of the shop opening/closing system does not align with the desired norm. Heerkens et al. (2017) define norm to be how a given process should work is perceived by most people or, in this case, the company. Reality is how actually things happen. MPSM aims to close then the gap between the reality of the company and its perceived norm.

For our specific case, the norm should be that the decision to allow new orders comes from a systematic exploration. On the other hand, reality is the current approach, where there is no predictive decision-making process, but the decision is taken based on human gut feeling.

Knowledge problems

The following includes a list of the sub-research knowledge questions that will target to resolve the main research question. Each of the sub-research questions targets a different knowledge problem, and each aims to target one (or multiple) sections of the MPSM approach.

- **What is the current open-close strategy of Flaschenpost website?**

We will look at how the current decision-making system works, as well as what exactly does the company consider when making decisions. We then can concretize all the problem parameters, as well as which KPI's are relevant to determine a policy improvement. This question will also aim to answer the third phase of MPSM, as will give a better insight into the problem, and later help on generating solutions.

- **What is the state-of-the-art for reinforcement learning and Q-learning, and how can it be applied to Flaschenpost's case?**

We will do an exploration on reinforcement learning, Q-learning, its methodologies and applications. The deep understanding of these concepts will help on determining how will they later be implemented into Flaschenpost, as well as how to extract conclusions from them. The exploration of the literature will help generating potential solutions (phase 4 of MPSM).

- **How can the company's sequential decision-making process be modeled mathematically?**

This is, without a doubt, the most challenging part of the project. Once we have explored all the related literature, we will develop and formalize a mathematical model that takes all necessary parameters and decisions into account. Later, using this modeling notation, we will formalize an algorithm that can later be used for policy making. This directly targets step 5 of the MPSM framework, as will help us develop a policy (solution).

- **How can a Q-learning algorithm be implemented to the decision-making problem?**

This knowledge problem will include all the necessary steps to implementing the reinforcement a simulation environment with a simple routing algorithm and obtain a decision-making policy. This policy will indicate to Flaschenpost what decision (either open or close) should be made at each point in time. The algorithm then will be adjusted with the company's current operations, and we will detail the simplifications that were taken. This corresponds to the 6th phase of the MPSM. This section will also answer to which experimental set-ups are best fitted for addressing our problem.

- **What is the policy's performance based on the KPIs?**

Once the algorithm is completed and we have generated a policy, we will jump into evaluating the policy utilizing the KPIs. We can also compare the performance of our policy with a "mark-up" heuristic that replicates their current policy. On top of that, we will discuss the simulation model with the company, to make sure that the assumptions taken are correct, and the results are still representative. Overall, this will help us evaluate the performance of our solution, corresponding to the 7th (and last) phase of the MPSM.

Scope of the research

We can also define the general scope of the research, including its main goals.

- We will focus on modelling the decision-making process of Flaschenpost mathematically, according to the Markov Decision Process (MDP) framework. This formulation will be used later to implement Q-learning. Some simplifications will be done to make the problems possible to model.
- As seen before, Q-learning is the most common methodology for solving sequential decision problems using RL. For that reason, we will only focus on Q-learning, and how a Q-learning agent can be used for improving the decision-making process. The company is also interested in exploring the implementation of this RL methodology for their decision-making process. Further reasoning of the selection of Q-learning will follow in Chapter 3.
- We will focus on developing a decision-making policy that will only address if new orders are allowed at each point in time for a particular warehouse in a specific day.

Besides the scope of the research, other problem-simplifications will also be done later to help develop and implement a Q-learning algorithm. Despite that, we will make sure that they do not affect the relevance of the conclusions.

Validity and reliability

Cooper & Schindler (2014) define two main parameters to look for assessing a research measurement and quality: Validity and reliability. On one hand, validity refers to the methods used for measuring, analysis of the results and generalizability of the findings, ensuring that they measure and asses what they are intended to. On the other hand, reliability refers to the replication and consistency of the experiment.

For assessing validity, it will be crucial to include the company perspective and evaluation into interplay, maintaining them in the loop of assumptions made when modeling and simulating. This will make sure that the assumptions made do not invalidate the possible conclusions, or the quality of the policy generated. Following this reasoning, the main threads for the validity of the experiment are when creating the simulation environment for the reinforcement learning application. Some assumptions and simplifications will need to be made, but the environment behavior will need to still be significantly close to reality.

For assessing reliability of the results, we need to look at the replication of the policy across different scenarios. When implementing the policy to reality, some more factors may need to be considered. For that reason, during the implementation discussion of the project, we will look at possible changes that the policy would inquire if it is implemented into Flaschenpost operations.

1.4 Conclusions

This research aims to explore how a Q-learning algorithm can be used for determining a decision-making policy. This policy will indicate when should Flaschenpost allow/not allow new orders to be placed over the course of a day. To reach this goal, we divided the main research question into different knowledge problems, following the MPSM framework.

At the end of the project, we will then have mathematically modeled a simplified version of their sequential decision-making process, as well as used this mathematical model to train a Q-learning agent into taking optimal decisions. The policy that the Q-learning agent comes up with will provide some insight into how the actual Flaschenpost policy should look like, as well as what is the potential of this methodology for the decision-making.

The report will then be divided into 6 chapters (excluding the introduction). Chapter 2 will further describe the problem, as well as introduce the KPIs for evaluating the solution. Chapter 3 will explore the related literature of RL and Q-learning needed for the research. Chapter 4 will model the problem mathematically and formally describe the Q-learning algorithm. Chapter 5 will discuss the implementation of the algorithm. Chapter 6 will discuss the numerical experiments done to extract conclusions. Chapter 7 will conclude the research, summarizing the results, and discussing future improvements and recommendations to the company.

2 Detailed description of the problem

This chapter will discuss the current decision-making process and the KPIs needed for evaluating a policy. In Section 2.1, we will explore Flaschenpost operations, as well as which parameters does the company use for taking decisions. Section 2.2 will discuss which problem assumptions and simplifications will be done. Finally, section 2.3 will discuss the KPIs we will be using for measuring the policy's performance.

2.1 Flaschenpost current situation

As mentioned in the previous chapter, Flaschenpost specializes in last-mile delivery of fast-moving consumer goods. On their website, at any point in time, customers can place an order for beverages or food. This order can be placed to arrive immediately ("immediate order") or place an order that will arrive at some point in the following 3 days ("pre-order").

The option of placing an "immediate order" is not always available, as it depends on how much are the drivers being utilized, as well as what is the status of the already placed orders. The decision-making process of whether to allow customer to place an "immediate order" is the target of this research.

2.1.1 Exploration of the daily operations

Flaschenpost's daily operations can be generalized as daily operations for a real-time delivery service provider. For a better understanding of the problem, Figure 2-1**Error! Reference source not found.** summarizes the relevant business process occurring through a day of operations, including the decision point.

The company opens every day from 8:00 to 20:00. Every 5 minutes, all the information on the status of the drivers and the orders to be processed is gathered. The algorithm then generates routes and computes a series of parameters. These parameters give the company an insight into the status of the drivers, as well as when will the orders be delivered. Based on the parameters, the company will then decide to allow or not to allow immediate orders for the following 5 minutes. If open, new immediate orders will arrive while, if closed, only pre-orders will be placed.

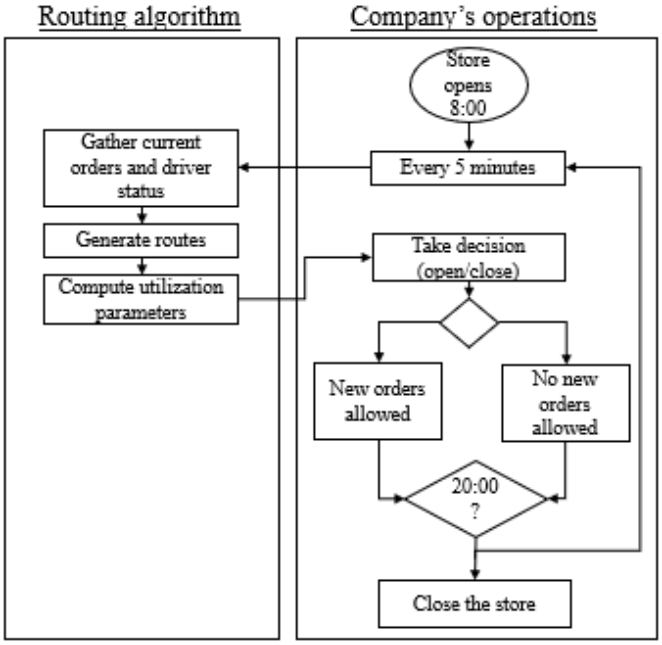

Figure 2-1 - Relevant business process for decision-making.

There are also three important concepts to specify: “loading units”, “orders” and “drivers”. Flaschenpost groups all the possible items a customer can order into “loading units”. Each loading unit has a fixed dimension (40x30 cm) as the size of an average beverage crate. For non-beverage items, the company places them in a box with equal dimensions. This unit will be used, among others, when determining the “size” (or “load”) of an order or when measuring the capacity of a driver.

With “orders” we refer to the items that a customer has requested to be delivered. Each order has different characteristics, including the customer’s location, the “size” of the order, and the time it has been placed. Flaschenpost also specifies that, for the order to be delivered “in time”, it must arrive within 2 hours of being placed. The order can still be delivered within the next 2 hours, but a penalty will be inquired, and the order will be considered to arrive “late”. After 4 hours have passed, the order won’t be delivered anymore.

“Driver” refers to all the information combining the vans that the company uses for delivery and the specific employee in charge of the delivery. Every time the routing algorithm runs, all the active drivers get assigned a route including all the orders they need to deliver. As with orders, each driver has some associated characteristics. The driver has a maximum time they can drive before needing to go back to the depot, a maximum load they can carry, maximum number of different orders they deliver in one route, etc. These characteristics may be either unique or shared across all drivers.

Besides all the general information given about Flaschenpost daily operations, there are some other factor/parameters that could also be considered for addressing the problem: driver limitations due illness, dealing with sudden peaks in demands, time taken to prepare an order on the warehouse, etc.

2.1.2 Decision-making parameters

In the previous section, we specified when an opening/closing decision is made. This decision is made by looking at a series of parameters including driver utilization and status of active orders (all the orders placed by customers that need to be delivered at a given point in time). After a series of talks with the company employees, the following list has been drafted:

- Average loading utilization of the drivers. It indicates what is the average utilization of the loading capacity of the drivers. I.e., if a driver’s van is planned to carry 25 loading units, and his capacity is 50, then it has a loading utilization of 50%.
- Trip duration utilization. Indicates the average ratio between a driver’s assigned route duration and the maximum possible duration of the route. I.e. if a driver is assigned a 30 min. route, out of a possible 2-hour maximum duration, then it has a trip duration utilization of 25%.
- Percentage of orders assigned. At each point in time, a company has a set of orders that have been placed and need to be delivered: the active orders. This parameter indicates the percentage of orders that have a driver that delivers them across active orders.

- Estimated percentage of orders arriving in time, or “order punctuality”. The routing algorithm not only generates the routes but also calculates the expected delivery time of each order based on the routes generated. Using that information, order punctuality computes which percentage of the active orders will arrive in time (2-hour window)
- Average order tardiness. Using the same delivery estimation, the parameter indicates how late (in minutes) would an average order arrive.

As of right now, the company looks like these parameters (plus the point in time of the day) and takes a decision based on gut feeling. I.e. if less than 80% of the orders are predicted to arrive in time, then close the store. The drivers then execute the routes, and after 5 minutes have passed, their routing algorithm runs again with the active orders, computes again the parameters and a new decision is taken.

2.2 Problem assumptions and simplifications.

Once we have described the main problem, we need to mention the main simplifications and problem assumptions done for modeling the problem. Mainly, we generalized and simplified the problem with respect to the actual. Among other things, we can highlight some generic assumptions:

- An order cannot be delivered later than 4 hours from it being placed. Thus, if this is the case, the order will be lost, and an associated penalty will be inquired.
- Company performance will not affect the demand, which implies that orders arrival will not be affected by the company not delivering previous orders.
- Each day behaves independently; thus, the problem can be modeled only considering the different stages of 1 day, without considering how that day will affect the following.

Besides the assumptions, we also included some simplifications to reduce the problem complexity.

- Only the KPIs given by the company will be considered for evaluating a given state. We will not explore the possibility of changing them, but just how they can be used for Q-learning. This KPIs will be the ones looked for making decisions at each point in the day.
- We remove the possibility of pre-orders, that means only immediate orders can be placed by customers.
- Orders will only start arriving the moment the shop opens. This implies that there will be no orders to be fulfilled before 8:00.

There are also some fundamental problem characteristics we will keep. This includes orders containing a specific load, or delivery drivers with specific characteristics and constrains. Keeping these concepts is fundamental to ensure that the validity of the research is not affected, and that relevant conclusions can still be drawn.

2.3 KPI selection

To evaluate the effectiveness of the policy, we will need a complete list of KPIs, as well as determining an operationalization strategy. We can divide the KPI section into two. The first KPIs come from the parameters the company uses for taking the decision (explored in Section 2.1.2). The second KPIs will help measure the performance of the policy through a day.

- From the company parameters:
    - Average loading utilization of the drivers
    - Average trip duration utilization per day
    - Average percentage of orders assigned per day.
    - Average order punctuality per day.
    - Average tardiness per day.
- From the output
    - Orders fulfilled on a day.
    - Orders delivered late.
    - Orders not delivered on a day

The first set of KPIS are derived from the parameters the company currently uses, averaged through the day. This will give an insight on weather their utilization has changed, increased, or decreased with the policy found.

The second are daily performance outputs of the policy, tailored to determine if the “sale opportunity” has been improved. We will define an improvement in the “sale opportunity” as a balance between more orders fulfilled on a day, orders being more punctual and less orders not being delivered. We will later merge this 3 KPIs into the “daily reward”, which weights the orders accepted, orders delivered late, and orders not delivered.

The KPI operationalization will be done alongside the numerical experiments in Chapter 6.

2.4 Conclusions

Flaschenpost takes a decision of whether to allow orders every 5 minutes based on different utilization parameters. These parameters include information of the driver utilization, and the status of the daily orders. Flaschenpost current operations are quite complex, so some problem simplifications are done to make sure that is still possible to model and solve the problem. The simplifications mainly regard the behavior of the demand, as well as the exclusion of pre-orders, as it would make the modeling more complicated and difficult to solve in the given time window.

From the exploration of the company decision-making, we have defined the main KPIs over which the performance of the solution will be addressed. The KPIs are a mixture of the parameters used by the company, as well as extra information on the performance of the policy based on its output.

3 Related Literature

As explained through the previous chapters, Flaschenpost faces a sequential decision-making problem for allowing/not allowing orders. Yan et al. (2022) explores the rising trend in research for approaching these problems using reinforcement learning. During this section we will explain sequential decision problems in the context of MDP, and how reinforcement learning (especially Q-learning) can be used to develop an optimal solution policy.

Section 3.1 will discuss the Markov decision process framework. Section 3.2 will introduce the concept of reinforcement learning on the context of a Markov decision process. Section 3.3 will focus on reinforcement learning methodologies, with a special emphasis on Q-learning. Finally, Section 3.4 explores how our specific problem fits within the literature, and what current methodologies are used to solve sequential decision-making problems.

3.1 Markov decision process, policies and value functions

For understanding reinforcement learning, first we need to understand a Markov Decision Process (MDP), as a mathematical modeling approach for building a reinforcement learning (RL) framework. Through this first section of the chapter, we will formalize the definition of an MDP. This will serve as methodological background for later understanding of reinforcement learning methodologies during follow-up sections.

3.1.1 Markov decision processes (MDP)

In short, an MDP is meant to be used as framework in problems where the objective is to achieve a goal by learning through interacting with its environment (Bellman, 1957). MDPs is then a classical formalization of sequential decision-making under uncertainty, where actions determine (up to a certain extent) immediate reward and subsequent situations (states) (R. S. Sutton & Barto, 2018). MDPs could be used for formulating both problems with a finite and infinite number of possible states. On this theoretical framework (and for the remainder of the thesis) we will only be working with finite MDPs, where the number of possible states is finite.

Elements of a Markov decision process

To start, we need to define the different elements that compose a Markov decision process: states, actions, transition function and reward function (Watkins, 1989; van Otterlo Martijnand Wiering, 2012).

States – We denote environmental state $\mathcal{S}$ as the finite set of variables $\{s^1, s^2, \dots, s^N\}$ of the system that compose a specific characterization of all the relevant information needed to take a decision, where each s represents a possible state of the system.

Actions – A set of actions $\mathcal{A}$ refers to a finite set all possible decisions that can be taken $\{a^1, a^2, \dots a^K\}$, where the size of the action space is K. The possible set of actions may be dependent on the state s, thus, we denote this set of actions as $\mathcal{A}(s) \in \mathcal{A}$.

Transition Function – In an MDP, when an action $a \in \mathcal{A}$ is taken in each state $s \in \mathcal{S}$, the system makes a transition to a following state $s' \in \mathcal{S}$. The transition function is a probability distribution that determines the probability of landing in state s' from current state s and given action a. The transition function can be written as follows (van Otterlo Martijnand Wiering, 2012):

$$T(s', s, a) = P(s'|s, a) \tag{3.1}$$

$P(s'|s, a)$ represents the transition probability of going from state s to state s' given an action a.

In an MDP, the system being controlled follows Markovian dynamics (R. S. Sutton & Barto, 2018). The idea behind this property, is that the current state s gives enough information to make an optimal decision.

A MDP can be both deterministic and stochastic. In a deterministic MDP, each action-state pair will always result in the same follow-up state s'. On the other hand, in a stochastic scenario, there is a probability distribution indicating the likelihood of each outcome. For both scenarios, we get:

$$\sum_{s' \in \mathcal{S}} T(s', s, a) = 1 \tag{3.2}$$

Reward Function – Taking an action a at a given state s, results in a numerical reward $r \in \mathcal{R}$, where $\mathcal{R}$ represent the set of all possible rewards. The reward function R returns the reward (either deterministic or expected) given the action, the state and, depending on the MPD, the follow up state. The reward function can be then defined as $R: \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$, and computed as follows (R. S. Sutton & Barto, 2018):

$$R(s', s, a) = \mathbb{E}[\mathrm{r}|s', s, a] = \sum_{r \in \mathcal{R}} \sum_{s' \in \mathcal{S}} r * P(r|s', s, a) \tag{3.3}$$

The Agent-Environment Framework: Finite vs Infinite Horizon MDP

We refer as the *agent* to the learner and decision maker, while we refer as the *environment* to everything outside the *agent* itself (R. S. Sutton & Barto, 2018). the interaction between *agent* and *environment* occurs through a series of stages $t = 0,1,2,3, \ldots$

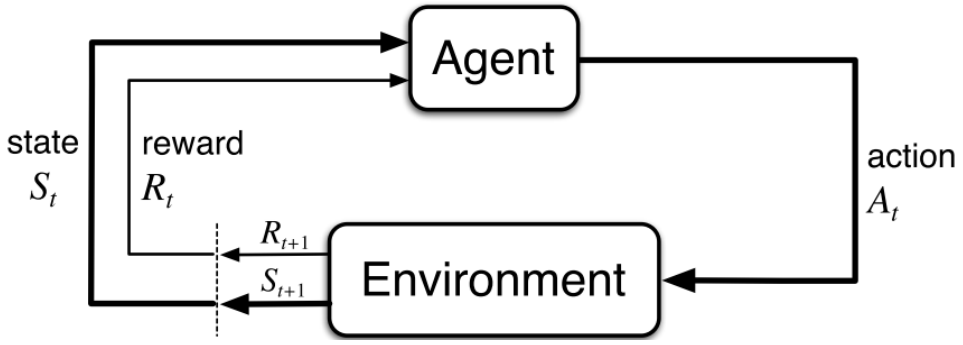

Figure 3-1 - Agent-Environment interaction as a MDP (R. S. Sutton & Barto, 2018).

Thus, over stages t, the *agent* will observe a state S_t and a reward R_t, and then make an action A_t, which will result on a new state and reward (S_{t+1}, A_{t+1}).

Usually, a MDP can be either an infinite or a finite horizon problem. In an infinite horizon problem, there is not an initial state or end state, but the interaction could start at any state and will go on for an infinite number of stages. On the other hand, for finite horizon problems the stages are divided into $t = 1,2,3, \dots, T$, where T indicates the terminal state of the interaction. Finite horizon MDPs also have an initial state S_0 and terminal state S_T.

In the case of Flaschenpost's decision-making problem we also have starting stage and end stage (beginning and end of the day), thus the problem at hand needs to be modeled as a finite horizon MDP.

3.1.2 Policies and value functions

Now that the basic MDP framework has been defined, we can look at the evaluation of each state, as well as the decision-making process of the agent. With this in mind, we can define the concept of *policy* and *value function.*

Policies

We understand policy as a rule for deciding what to do given knowledge of the current state. A policy should be defined over the entire state space (Watkins, 1989).

A policy (π), is a computable function that determines the probability of selecting each possible action from each given state. Thus, $\pi(a|s)$ gives the probability that $A_t = a$ if $S_t = s$ at a time t (R. S. Sutton & Barto, 2018). Policies can be divided into both stochastic and deterministic. On a stochastic policy, the action taken given a certain state comes from a probability distribution, while in a deterministic policy, from a given state *s*, the agent always takes action *a*.

MDPs also make use of *Markovian dynamics*. These dynamics ensure that the next state only depends on the current state and the *agent*'s decision; the current stage does not have an impact on the follow up state, nor the reward (R. S. Sutton & Barto, 2018). For that reason, the stage *t* is not considered on the policy.

Value function

We denote with $v_\pi(s)$ as the value of a state s under a policy π. This value represents the expected return if policy π is followed, starting at s (van Otterlo Martijnand Wiering, 2012). We can formally define $v_\pi(s)$ as follows:

$$v_\pi(s) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k \, \mathrm{R}_{t+k+1}(s', s, a) \mid \mathrm{S}_t = \mathrm{s}\right], for\ all\ s \in \mathcal{S} \tag{3.4}$$

In the value function, γ is a parameter ($0 \leq \gamma \leq 1$) that represents the *discount rate*, which determines the present value of future rewards. The closer the *discount rate* is to 1, the stronger future rewards are considered, thus the agent will become more farsighted (R. S. Sutton & Barto, 2018).

Similarly, we can also compute the *state-action* value function $q_\pi(s, a)$, as the expected return from state s, selecting action a and then following π (van Otterlo Martijnand Wiering, 2012).

$$q_\pi(s, \mathrm{a}) = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k \, \mathrm{R}_{t+k+1} \mid \mathrm{S}_t = \mathrm{s}, \mathrm{A}_t = \mathrm{a}\right] \tag{3.5}$$

3.1.3 Solving Markov decision process

The goal of a MDP is to find the best/optimal policy π^*. Thus, solving the Flaschenpost sequential-decision problem will involve finding, at each point in time of the day (stage), what decision (action) maximizes the reward at the end of the day. For solving MDPs we need to define both the Bellman equation, and the environment modeling.

Bellmann equation and Optimal policies

One fundamental property of value functions is that they satisfy a certain recursive property, in which a value function can be formulated recursively given policy π and state s. This formulation is called *Bellman Equation* (Bellman, 1957).

$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[\mathrm{R}_{t+1} + \gamma v_\pi(s') \mid S_t = s] \\ &= \sum_a \pi(a|s) \sum_{s',r} P(s', r \mid s, a)\,[r + \gamma v_\pi(s')]\ for\ all\ s \in \mathcal{S} \end{aligned} \tag{3.6}$$

This formulation shapes the relationship between the value function of the current state and the value function of follow-up states. Thus, the value at a given state *s* comes from adding the discounted values of all possible follow-up states s'.

An *optimal policy* π_*, is that policy for which its state value function is greater or equal than any other possible policy's (R. S. Sutton & Barto, 2018), thus $v_{\pi_*}(s) \geq v_\pi(s)$ for any $\pi \neq$

π_* and for all $s \in \mathcal{S}$. We refer to v_* as the *optimal state-value function,* and q_* as the *optimal state-action function*. Bellman (1958) introduces the *Bellman optimality equation* for both v_* and q_*.

$$v_* (s) = \max_a \sum_{s\prime} P(s', r|s, a)(r + \gamma v_*(s')) \tag{3.7}$$

$$q_* (s, \mathrm{a}) = \sum_{s',\ r} P(s', r|s, a) \left[r + \gamma \max_{a\prime} q_* (s', \mathrm{a}')\right] \tag{3.8}$$

The argument associated to every state s of *optimal value function* will then be the optimal policy π^*.

Model-based vs Model-free.

To find the both $v_*(s)$ and π_*, the first important distinction is between *model-based* and *model-free* algorithms. Model-based algorithms rely on the complete knowledge of the environment behavior: state transition function (and thus transition probability distribution). This knowledge can be used for computing value functions and policies using the Bellman equation. These algorithms receive the general name of dynamic programming (Kaelbling et al., 1996; van Otterlo Martijnand Wiering, 2012).

On the other hand, on a *model-free* MDP, the transition or/and the reward function are not known. For that reason, the agent must interact with the environment to obtain information, and ultimately obtain the optimal policy. For addressing these types of problems, it is used Reinforcement Learning (Kaelbling et al., 1996).

For Flaschenpost case, if the transition probabilities were known, we could use a *model-based* algorithm (dynamic programming). Despite that, in Flaschenpost sequential-decision problems we do not have knowledge of the transition probabilities as multiple uncertain factors may come into play: new orders arriving for the following stage, order processing times, etc. For that reason, we cannot use a model-based approach (dynamic programming) for determining the policy, but we will use Reinforcement Learning

3.2 Introduction to reinforcement learning

As mentioned before, because Flaschenpost problem transition probabilities cannot be known, we will be using Reinforcement Learning (RL). This section elaborates in this concept, as the group of algorithms and methods for solving *model-free* MDP problems. We first will define RL within the MDP framework, and then we will explore solution methods and the Q-learning algorithms.

3.2.1 Formalization of reinforcement learning

For understanding reinforcement learning (RL), we can look at the following definition by R. S. Sutton & Barto (2018).

"Learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. …, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards." (Sutton & Barto, 2018, p. 1).

Most times, it is not possible to obtain precise information on the transition probabilities, so there is a requirement for an exploration of the environment and registering rewards associated with each decision. The results of the interaction between the environment and the agent will be later used for optimizing the general policy and, eventually, arrive to an approximation of the optimal policy (R. S. Sutton & Barto, 2018). The agent learns which actions are better by maximizing the cumulative reward over the whole process.

3.2.2 Reinforcement learning dimensions.

Reinforcement learning algorithms and methods are divided following a series of dimensions/characteristics, depending on the context of the problem. Theses dimensions will shape how the agent learns, and how the RL behaves. We will take into account the different dimensions when selecting the reinforcement learning model on following sections (van Otterlo Martijnand Wiering, 2012).

On-Policy vs Off-Policy methods

The first main dimension for reinforcement learning is the difference between On- and Off-Policy learning. In *On-policy* methods, the agent constantly evaluates the policy used to make decisions, updates state values accordingly, and tries to improve the policy used. *Off-policy*, on the other hand, the agent explores using a *behavior* policy π_b (to understand and evaluate the environment) and uses this knowledge for evaluating (and improving) a deterministic policy: the *target* policy π_t.

Online vs Offline methods

Other characteristic that divides reinforcement learning methods is whether the learning occurs *Online* or *Offline*. Online learning, the agent utilizes real-world data for learning π_*. This may be very hard due to uncomplete or biased learning data. Offline methods create an environment that replicates the real world. The agent then interacts with this simulator, as a safer (and more complete) way of exploring and making mistakes (van Otterlo Martijnand Wiering, 2012).

Flaschenpost problem will be addressed with an offline method, as creating an environment for our reinforcement learning agent to learning will allow us to explore different extreme scenarios that the company data may not include. For this reason, besides the reinforcement learning algorithm, our problem will require us to create a learning environment that simulates the company's operations.

Exploration vs exploitation

Through the interaction with the environment, the agent improves the estimate value for each action at each state. Among the set of possible actions, the *greedy action*, is the action with the highest estimated value (Yahyaa, 2015). At any stage t the agent can then decide between

to strategies: *Exploitation* and *Exploration*. *Exploitation* consists of choosing the *greedy action* (despite maybe not being the optimal one), while *exploration* refers to choosing a non-greedy action to improve the estimate of that state-action reward.

There exist different selection policies that balance between *exploration* and *exploitation*: SoftMax, Pursuit, etc. But the most common used basic reinforcement learning applications is the ε-greedy exploration (Yahyaa, 2015).

In ε-greedy exploration, at every stage, the agent selects the greedy action with probability $1-\varepsilon$, while selects uniformly at random any of the other actions with probability $\varepsilon \in [0,1]$ (Yahyaa, 2015).

In general terms, the *exploration* is more important when the agent just started interacting with the environment, as it does not possess that much knowledge, while towards the end of the interaction (last *episodes*), *exploitation* gains importance. For that reason, ε should decay through the life of the agent (R. S. Sutton & Barto, 2018).

3.2.3 Reinforcement learning challenges.

Reinforcement learning presents a series of challenges, which may need to be considered when implementing reinforcement learning.

Credit assignment

Reinforcement learning (as seen during the previous sections) is heavily influenced by the rewards received. It may be that the decision made at the beginning of one episode may heavily affect the reward of future episodes. This is referred as the *temporal credit assignment* problem (van Otterlo Martijnand Wiering, 2012).

Curse of Dimensionality

Powell (2011) introduces the *three curses of dimensionality*, when the MDP is sufficiently big that cannot be solved. These three curses refer to having too large $\mathcal{S}$ so no $V(s)$ can be calculated, too large $\mathcal{A}(s)$ so no optimal action can be found, and the future reward can be too hard to compute.

We will refer as *tabular reinforcement learning* to all those MDP problems from which its state and action space is small enough to approximate value functions as arrays (R. S. Sutton & Barto, 2018). Other methodologies need to be used if the problem suffers of one (or more) of the curses of dimensionality.

3.3 Reinforcement learning algorithms and solution methods.

During this section, we will discuss the main type of reinforcement learning that will be used for the research as part of temporal difference learning: Q-learning, as a part of the temporal difference learning methods.

3.3.1 Temporal difference learning methods

Temporal difference (TD) learning methods use experience to compute and adjust its estimate of $v(s)$, and thus determine π_*. In TD methos, for every new visit at a given state, the agent makes the following update (R. Sutton, 1988):

$$V(S_t) \leftarrow V(S_t) + \alpha\,[R_{t+1} + \gamma V(S_{t+1}) - V(S_t)\ ] \tag{3.9}$$

$V(S_t)$ represents the agent's estimate of $v(s)$ at a given stage t. This estimate is updated by the difference between the most recent estimation $R_{t+1} + \gamma V(S_{t+1})$ and the current estimate $V(S_t)$. $\alpha \in [0,1]$ represents the *step size* parameter, which determines how much the difference between the new information and previous estimate affects the new estimate. If $\alpha = 1$, the new estimate its just the new observation $(R_{t+1} + \gamma V(S_{t+1}))$.

3.3.2 Basics of Q-learning

Q-learning was firstly introduced by (Watkins, 1989), as a temporal difference reinforcement learning algorithm for approximating the state-action value function $q_*(s, \mathrm{a})$. The Q-learning update rule is defined as follows:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left[R_{t+1} + \gamma \max_{a \in \mathcal{A}(S_{t+1})} Q(S_{t+1}, a) - Q(S_{t+1}, A_t)\ \right] \tag{3.10}$$

The learned value function Q is then an approximation of q_*, which is independent on the policy being followed (R. S. Sutton & Barto, 2018). The Q-learning algorithm belongs then on the tabular solution methods, as Q approximates state-action values for each pair and stores it in a table (Q-table).

$$Q(S_t, A_t)$$

Q-learning is also an off-policy method, where a behavior policy is used for learning about the environment and determining the Q-table. After that, the target policy will be given by

$$\pi_t(s) = \operatorname*{argmax}_a Q(s, a) \tag{3.11}$$

Note that, if the state-action pairs continue to be updated, Q will converge to q_*. The algorithm for the Q-learning looks as follows Figure 3-2:

Q-learning (off-policy TD), for estimating $\pi \approx \pi_*$

Algorithm parameters: step size $\alpha \in (0,1]$, $\varepsilon \in (0,1)$
Initialize $Q(s,a)\ for\ all\ s \in \mathcal{S}, a \in \mathcal{A}(s)$ = 0 (Q-table)

Loop for each episode:
    Initialize S
    Loop for each stage of episode (t):
        Choose A_t from $\mathcal{A}(S_t)$ using policy derived from Q (e.g., ε-greedy)
        Take action A_t, observe R_{t+1}, S_{t+1}
        Use Q-learning update rule (3.10)
        $S_t \leftarrow S_{t+1}$
    Until S or t terminal

Figure 3-2 - Q-learning algorithm (R. S. Sutton & Barto, 2018)

For each episode, there the cumulative reward $R_{episode}$, where:

$$R_{epsode} = \sum_{t=0}^{T} R_t \tag{3.12}$$

Hyperparameters

We will denote as *hyperparameters* all those parameters that will affect the Q-learning model. Different algorithms (and implementation) will require different hyperparameters, as this can highly influence the learning of the reinforcement learning agent (Yang & Shami, 2020). For the Q-learning described above, we can consider the initialization of the Q-table, α, or ε as hyperparameters. This hyperparameters need to be tuned, as different problems require different values.

3.3.3 Limitations of Q-learning

It is also important to discuss some of the limitations of Q-learning approach, as it will give some insights into possible sources of error. As explained before, Q-learning is part of the tabular solution methods, where all the information for state-action pairs needs to be stored (R. S. Sutton & Barto, 2018). This may lead to slow convergence and high computational requirements.

Associated with the tabular method, we have one of the challenges discussed priorly: *curse of dimensionality*. To add to the already slow performance, as environments get exponentially more complex, it may become challenging to represent the state in a way that can still be stored. Here is where *state space aggregation* is useful. The main goal behind it is to group similar state together into a smaller set of aggregated states, which allow the learning algorithm to just focus on smaller sets of representative states (R. S. Sutton & Barto, 2018).

3.3.4 Other methodologies

Besides Q-learning, there are multiple other RL solution methods, which may be used for different problems depending on their action and state spaces. R. S. Sutton & Barto (2018)

introduces other methods for approximating the Q-value such as value function approximation methods or DeepQ learning.

Value function approximation methods come into play when tabular solution methods are no longer possible, as all the sate-action values can no longer be registered. This methodology assigns a vector for the state action pair and combines it with a certain weight. Through interaction, the weight varies to more precisely estimate $Q(s, a)$.

An expansion of the Q-learning methodology is DeepQ learning, where instead of using a table or a function to approximate $Q(a, s)$ a neural network is used, which allows for a larger state-action space (and reducing the previously mentioned dimensionality issues)(Yahyaa, 2015).

Despite these methods existing, through this thesis we will only approach the problem presented through Q-learning.

3.4 Q-learning in finite horizon sequential decision problems

Now that we have defined both the type of problem and the Q-learning methodology, we can explore how a finite horizon sequential decision problem for order acceptance can be solved with Q-learning. First, we will justify the use of Q-learning as main method for approaching the Flaschenpost sequential decision-making problem, and then we will dive into similar decision-making studies that implement Q-learning.

3.4.1 Q-learning as RL solution method

There are various reasons why Q-learning has been chosen as the RL solution method. Firstly Q-learning is a model-free RL method. As mentioned previously, for the problem at hand we do not count with complete knowledge of the transition probabilities, for that reason a model-free RL approach should be the way to go. Q-learning also greatly simplifies the RL procedure, while still showing excellent learning ability (Jang et al., 2019). This will greatly help solving the decision-making problem, as the simplicity of Q-learning will help when designing a simulation environment for the offline learning.

Besides Q-learning being useful for the specific problem at hand, as it was discussed on Section 1.2.2, among the newly researched RL algorithms for supply chain and logistics, most of them are based on Q-learning (Yan et al., 2022). This is another reason favoring the use of Q-learning for our research.

The main disadvantages are when facing an extremely large action-space, or when multiple agents are interacting with the environment (Jang et al., 2019). Despite that, we will reduce the state-action space with state-space aggregation, and only one agent (opening/closing store decision-maker) is involved in our problem.

3.4.2 Q-learning in supply chain/logistics sequential decision-making problems

Now that we have decided to use Q-learning, we need to briefly explore different research that solve sequential decision-making problems in the fields of urban logistics, last-mile delivery, or FMCGs using this methodology.

Regarding order acceptance, there has been diverse studies that use Q-learning-derived methodologies for the decision-making process. We can highlight Kang (2018), where it explores single-vehicle order acceptance using DeepQ learning. We can go even further on the literature, Chen et al. (2023) explores the sequential decision-making problem of order acceptance but in same-day delivery. In this paper, Chen also balances between number of orders accepted and overall service rates. This research also makes use of DeepQ learning, as a stable and efficient learning algorithm. Kavuk et al. (2022) goes even a step closer to our problem, as they explore these types of problems in the fields of FMCGs for ultra-fast delivery services. In this research, DeepQ learning is also used, as well as different decision-making agents, as they have a wider action-state space.

From the literature on similar problems, we can already spot some differences compared to our research. We will focus on tabular Q-learning, rather than using DeepQ learning. DeepQ learning offers a more accurate estimate of the Q-value (and thus gets closer to the optimal policy) but greatly increases complexity and becomes harder to implement. On the contrary, tabular Q-learning offers a simpler implementation, which can be crucial, given that for Flaschenpost's problem we also need to create a simulation environment.

In the literature, there are also some uses of tabular Q-learning for sequential decision-making. Lopes Silva et al. (2019) explores the use of this methodology for urban logistics. Despite that, the focus of the research is on a routing and scheduling problem, rather than order acceptance, as well as making use of multiple learning agents.

This research will then not only help Flaschenpost with their decision-making problem but also help explore a sequential decision-making problem for order acceptance of FMCGs using a single agent tabular Q-learning approach. This specific problem-solution pair seems not widely explored in the literature, and thus it can help push forward the scientific body of knowledge.

3.5 Conclusions

This section brings an answer to one of the knowledge questions raised during the introduction: What is the state-of-the-art for reinforcement learning and Q-learning, and how can it be applied to Flaschenpost's case?

Flaschenpost is a same-day FMCG delivery company facing a sequential decision-making problem for order acceptance. These types of problems can be modeled using MDP mathematical modeling and solved using RL.

RL algorithms are a model-free methodology, used to find the optimal policy for MDP modeled problems when the transition probabilities are not known. Q-learning belongs to the

RL algorithm solution methods and is especially relevant for sequential decision problems in the field of logistics and supply chain (Yan et al., 2022). Among all the variations of Q-learning, the tabular approach is the most fitting for our problem due to its implementation simplicity. Inside the reinforcement learning dimensions, tabular Q-learning is an off-policy model-free solution method. With this method, the agent interacts with the environment and updates its estimate of $Q(a, s)$, registering all the state-action pairs in a table. We will use an offline methodology for the agent-environment interaction, where the environment will be a simulation that replicates the behavior of Flaschenpost operations.

Tabular Q-learning comes with different challenges and limitations, mainly regarding slow convergence and other potential dimensionality issue. State space aggregation is used for dealing with those issues, as reduces the state space by combining similar states into one value. Besides, there are some other more complex Q-learning-based methods to solve these problems, such as value function approximation or DeepQ learning.

Among the current research in Q-learning-based solution methods for logistics and supply chain, most of the literature utilizes more complex Q-learning methodologies (mostly DeepQ learning) when solving order acceptance sequential decision-making problems. There is very little to no research on how to implement tabular Q-learning for an order acceptance sequential decision-making problem.

4 Mathematical model

Through this chapter we will model the problem as a finite horizon MDP and model the tabular Q-learning algorithm, based on the theory explored in Chapter 3. This chapter will aim to address the 4th section of the MPSM framework. Section 4.1 will discuss the modeling approach and specific modeling decisions. Section 4.2 will include the complete mathematical formulation. Section 4.3 will address the Q-learning model and formalize the algorithm. Finally, section 4.4 will address how the model will deal with dimensionality issues.

4.1 Modeling approach

We will then utilize the related literature concepts and notation to model the decision-making process as a finite horizon MDP, as well as the interaction between the agent and the environment.

4.1.1 RL dimensions

As seen in Section 3.2, for implementing reinforcement learning, it is first needed to contextualize the problem in RL dimensions. As mentioned before, Q-learning is an off-policy method. This implies that the agent will need to be trained using a *behavior* policy (π_b), where the agent explores the environment. The *behavior* policy will follow the ε-greedy method where, at each episode, there is an ε probability for the agent to perform the best know action. The epsilon will decay through the follow up episodes, to make sure that the ε-greedy policy converges to π_*.

As mentioned before, we will implement an offline Q-learning algorithm, thus there will need to be an environment that the agent will use for the training (a simulator). For our specific problem, our environment would need to simulate the orders arriving through the day, as well as the routes generated for delivering those orders. The agent is then free to explore the environment created in the simulation.

On Chapter 5 we will go also more in detail on the creation of the simulation environment.

4.2 Mathematical formulation

Once the modeling approach has been determined for implementing the Q-learning methods, we need to discuss all the elements of the MDP.

4.2.1 Stages

The opening time of the online store occurs from 8:00 until 20:00. Through the day, decisions to open or close the store occur every 5 minutes. This means there is a total of 144 decisions per day. Thus, we will consider a time variable $t \in \mathcal{T} = \{0,1,2,\dots,143\}$

4.2.2 States

The state space $\mathcal{S}$ for our MDP formulation will contain the same elements as they are currently considered by Flaschenpost for their decision-making process, giving information about the state of the drivers, the orders to be satisfied at each time t, and the expected arrival and punctuality of those orders.

Supporting parameters

Before diving into the state space, we need to define the following parameters.

Notation	Definition
$d^i \in D_t$	Driver No. (i) where D_t is the set representing all drivers at stage t. $D_t \subseteq D = \{d^1, d^2, \dots, d^{N_D}\}$
$o^i \in O_t$	Order No. (i) where O_t is the set representing all active orders to be fulfilled at stage t. $O_t \subseteq O = \{o^1, o^2, \dots, o^{No}\}$

Where N_D, N_O represent the total number of drivers and number of orders respectively.

Both drivers d and orders o have a series of associated parameters, that define the order at each given point in time. This information is later used for calculating the state of the environment. We can divide the information as follows:

Notation	Definition
o^i_x	Variable representing the order i x coordinate
o^i_y	Variable representing the order i y coordinate
$o^i_{time\_stamp}$	Point in time t when the order i was placed
$o^i_{acc\_lateness}$	Point in time until when the order i will still be considered to arrive "in time" (acceptable lateness of the order)
$o^i_{max\_lateness}$	Point in time until when the order i can still be delivered (maximum lateness)
$o^i_{assigned}$	Binary variable [0,1] that gets value 0 if $o^i \in O_t$ has not been assigned to a driver, and 1 otherwise.
o^i_{load}	Variable representing the number of load units of the order i
$o^i_{e\_delivery}$	Variable representing the expected delivery of order o^i
$o^i_p \in [0,1,2]$	Variable that represents whether order is predicted to arrive in time ($o_p = 0$), late ($o_p = 1$), or not arrive ($o_p = 2$).

Table 4-1 - Variables associated with order o

Note that:

$$o^i_p = \begin{cases} 0, & if\ o^i_{e\_delivery} < o^i_{acc\_lateness} \\ 1, & if\ o^i_{acc\_lateness} < o^i_{e\_delivery} < o^i_{max\_lateness} \\ 2, & if\ o^i_{max\_lateness} < o^i_{e\_delivery} \end{cases} \quad (4.1)$$

Where:

$$o^i_{acc\_lateness} = o^i_{time\_stamp} + 24 \tag{4.2}$$

$$o^i_{max\_lateness} = o^i_{time\_stamp} + 48 \tag{4.3}$$

For formulas 4.2 and 4.3 add 24 and 48 to the $o^i_{time\_stamp}$ because, as discussed in Chapter 2, those are the boundaries for an order arriving late (more than 2 hours) or not delivering it (over 4 hours). The drivers d also have some associated parameters.

Notation	Definition
d^i_{load}	Load assigned to driver d
$d^i_{max\_load}$	Maximum possible load of driver d
d^i_{length}	Current route length of driver d
$d^i_{\max\_length}$	Maximum possible length of route for driver d
d^i_{orders}	Set of orders associated with driver d
$d^i_{t\_remaining}$	Time remaining until driver d is available for new orders

Table 4-2 - Variables associated with driver d

State space parameters.

The state space will then be divided into the following parameters:

- $\ell_t \in \mathcal{L}$. Loading utilization. Average across all drivers of load by maximum possible load at stage t.

$$\ell_t = \sum_{d^i \in D_t} \frac{d^i_{load}}{d^i_{\max\_load}}, \qquad where\ \ell_t \in \mathcal{L} \tag{4.4}$$

- $x_t \in \mathcal{X}$. Trip duration utilization. Percentage of used route length per driver over the total possible at stage t.

$$x_t = \sum_{d^i \in D_t} \frac{d^i_{length}}{d^i_{\max\_length}}, \qquad where\ x_t \in \mathcal{X} \tag{4.5}$$

- $\sigma_t \in \mathcal{O}$. Orders assigned. Percentage of all active orders $|O_t|$ that have been assigned to a driver at stage t.

$$\sigma_t = \frac{1}{|O_t|} \sum_{o^i \in O_t} o^i_{assigned}, \qquad where\ \sigma_t \in \mathcal{O} \tag{4.6}$$

- $\mathcal{p}_t \in \mathcal{P}$. Order punctuality. Estimated percentage of active orders that will be delivered without delay at stage t.

$$\mathcal{p}_t = \frac{1}{|O_t|} \sum_{o^i \in O_t} 1\,, \qquad where\ \mathcal{p}_t \in \mathcal{P}\ and\ o_p^i = 0 \tag{4.7}$$

- $\mathcal{m}_t \in \mathcal{M}$. Average tardiness. Expected number of stages an order will be late at stage t.

As seen in 3.1, the state needs to contain all necessary information for taking an action, without needing to look to past states. For that reason, on top of the states already used by the company for decision-making, our state-space will also contain information about the time of the day (dependent on the stage *t*) when the decision is taken. We will call this parameter $n(t) \in \mathcal{N}$. Ideally, we would just use the stage t (and not use $n(t)$), but this would greatly increase the state space dimensions, so we introduce n for later aggregating the point in time of the day.

The entire set of states is a finite set $\mathcal{S}$. We define a given state $s \in \mathcal{S}$ by:

$$s \in \mathcal{S} = \{\mathcal{L}, \mathcal{X}, \mathcal{O}, \mathcal{P}, \mathcal{M}, \mathcal{N}\} \tag{4.8}$$

Note that, for the reinforcement learning application, the state space needs to be reduced, and constrained, to make sure that an optimal policy can be calculated using a tabular solution method. We will dive deeper into the state space constrained, and dimensionality issues when formalizing the Q-learning algorithm.

4.2.3 Decision variables

The action variable $a_t \in \mathcal{A}$ is a binary decision {0,1} taken at stage t. Here 0 represents closing the store for the following stage $t+1$, while 1 represent leaving the store open. Closing the store implies that no new orders will be arriving for the following stage. Note that, for our specific problem, the decision variable is not state dependent, but it can always be either 0 or 1.

4.2.4 Transition function

The new state s' depends on the previous state s, and the new number of orders that arrived in between stages w. The new number of orders will come from whether the store was open (a_t). We will define the probability of w orders arriving as:

$$P(w|a,t) \tag{4.9}$$

Where $P(w|a,t)$ represents the probability of w orders arriving given action a and stage t. Note that, as mentioned before, each stage represents an interval inside a day of operations, thus W is also dependent on t.

As mentioned before, s' depends on the previous state, action and new number of orders. We can then define the transition function T, and thus the transition to a following state s' will be given by:

$$s' = T(s, P(w|a,t)) \tag{4.10}$$

The details of how the transition function is computed will be discussed in Chapter 5.

4.2.5 Reward function

At each stage t there is an associated reward, representing the net increase in the number of orders delivered. At each stage t, there is the set of orders O_t, each order o^i can either arrive within the 0-2 hour from being placed ($o_p^i = 0$), arrive within 2-4 hours from being place ($o_p^i = 1$), and lastly not arriving at all (the time has surpassed the 4 hours after the order was placed, $o_p^i = 2$).

For this problem, the reward function R generates a reward depending on when the order o has arrived with respect to the stage t it was placed. Every time an order is placed (due to the store being opened), the system receives reward r_a, per order delivered late, the system receives reward r_l; and per order not delivered, the system receives reward r_n. Thus, the reward will be as follows:

$$R(a,s,t) = r_a * P(w|a,t) + \sum_{o^i \in O_t,\ o_p^i = 1} r_l + \sum_{o^i \in O_t,\ o_p^i = 2} r_n \tag{4.11}$$

4.3 Q-learning model

Once the problem has been modeled as a MDP, we now can explore the interaction between the agent and the environment in the context of a Q-learning application. This section will only discuss the main aspects of the Q-learning model, as the formalization of the algorithm will follow in chapter 5.

4.3.1 Q-learning structure

The Q-learning agent will interact with the environment through a series of episodes. Each episode is divided into 144 stages (t).

Figure 4-1 displays the relationship between the routing environment and the agent decision-making. We refer as "routing environment" to the environment that simulates the actual generation of routes for Flaschenpost.

At a given t, the state is defined as follows (from the MDP formulation):

$$S_t = [\ell_t, x_t, \sigma_t, \wp_t, m_t, n_t] \tag{4.12}$$

where $S_t \in \mathcal{S}$.

Using this information, the agent will act a_t, where:

$$a_t = \pi_b(S_t) \tag{4.13}$$

The routing algorithm gets the action plus the previous information of drivers and orders and generates the reward and the new state. As seen in Figure 4-1, the follow up state is connected

to the reward, as the reward represents the number of orders that have arrived due to the action a.

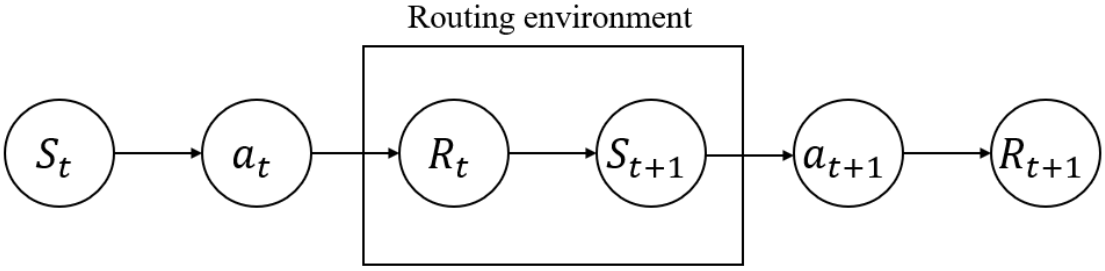

Figure 4-1 - Q-learning agent interaction with environment.

Once the information of $S_t,\ a_t,\ R_t,\ S_{t+1}$ is known, it will be used to update the $Q(s,a)$ value:

$$Q(S_t,A_t) \leftarrow Q(S_t,A_t) + \alpha \left[R_{t+1} + \gamma \max_{a \in \mathcal{A}(S_{t+1})} Q(S_{t+1},a) - Q(S_{t+1},A_t)\right] \quad (4.14)$$

This information will be used to update the Q-table, and to take following decisions. The process continues until the end of the episode.

4.3.2 Q-learning algorithm

The structure of the Q-learning algorithm is visualized in Figure 4-2.

As we discussed earlier, Q-learning is model-free, and thus there is the need to create a simulation environment that replicates the behavior of the company. This simulator oversees determining the daily demand and generate routes for the orders that arrive at each point in time. On chapter 5 we will explore more in detail the behavior of the environment, as well as the specific implementation for the ε-greedy explorer.

Each *episode* represents one day; thus, the agent runs through the same day an E number of times, adjusting the Q-values following equation 4.14. Each episode also has an associated cumulative reward $R_{episode}$.

The whole process of exploring the environment will be also referred as "training" the RL agent.

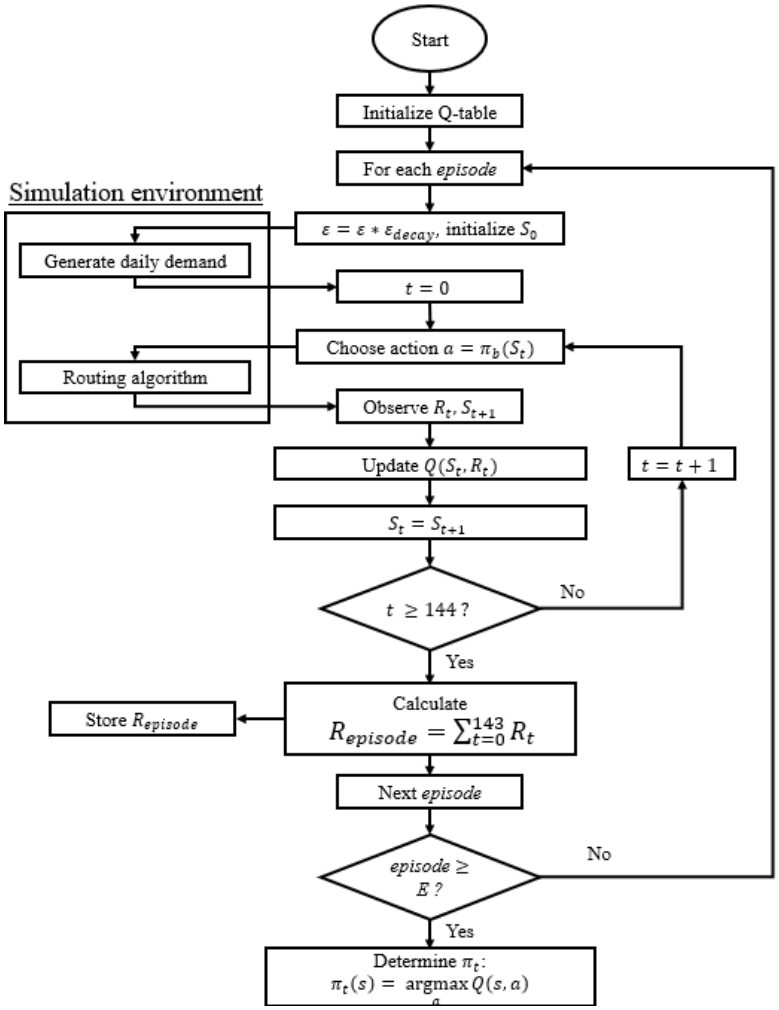

Figure 4-2 - Formalization of the Q-learning algorithm.

4.4 Dealing with dimensionality issues

From Section 3.2.3, one of the main challenges when implementing reinforcement learning applications is to deal with the curse of dimensionality. This is also one of the main limitations of tabular Q-learning, as ideally, we would use disaggregated information of the orders and drivers. For that reason, will then implement *state space aggregation*.

4.4.1 State space aggregation.

For aggregating the state space $\mathcal{S}$, each of the parameters that define a state have been grouped and constrained.

$\mathcal{L}$, $\mathcal{X}$, $\mathcal{P}$, are sets containing all the possible percentages. This percentages could get any real value from 0 to 1. For aggregating the state space, we will round the percentages to the first decimal and multiply it by 10, thus:

$$\mathcal{L}, \mathcal{X}, \mathcal{P} = \{0,1, \dots, 10\} \tag{4.15}$$

$m \in \mathcal{M}$ represents the average lateness (represented in t) of the active orders. This set could potentially contain any integer value from 0 to 143 (last stage). Despite that, the orders won't be delivered over 2h late (24 intervals), thus:

$$\mathcal{M} = \{0,1, \dots, 24\} \tag{4.16}$$

Lastly, we also have the time variant $n(t) \in \mathcal{N}$. If dimensionality was not an issue, we could establish $n(t) = t$, n taking any values from 0 to 143. Despite that, for reducing the state space, we will divide the day into 10 equally sized intervals (n), where:

$$n(t) = Floor\left(\frac{t}{interval\ size}\right), where \tag{4.17}$$

$$interval\ size = \frac{144}{10} = 14{,}4 \tag{4.18}$$

$interval\ size$ represents the number of stages contained in each n. With this, we get that:

$$\mathcal{N} = \{0,1, \dots, 9\} \tag{4.19}$$

There will still be 143 decisions per day, but the agent will not make a distinction between states at a similar point in the day. With the state space aggregation $n(142) = n(143) = 9$. This implies that, if the drivers and order states are the same at both stages, $S_{142} = S_{143}$, and thus $a_{142} = a_{143}$.

4.5 Conclusions

Through this section we have modeled the problem as an MDP based on the problem formulation of Chapter 2. This modeling framework was also used for modeling the specifics of the Q-learning. This includes determining the dimensions of our reinforcement learning model, as well as the state-action-reward behavior.

We can also conclude that, for training the Q-learning agent, we will need a reinforcement learning environment that simulates the company operations. This learning environment will be further discussed on the following chapters.

5 Implementation of the Q-learning algorithm

This chapter is related to the following knowledge problem: How can a reinforcement learning method be implemented to Flaschenpost? For answering the question, we divide the chapter in 2 main sections. Section 5.1 will discuss how the reinforcement learning environment was implemented (generating demand and routing algorithm). Section 5.2 will discuss how the Q-learning algorithm was implemented.

5.1 Reinforcement learning environment.

As seen in Chapter 4, the Q-learning agent learns by interacting with an environment that simulates a day inside the company's operations. This environment is divided into:

- Generate the daily demand.
- Create a routing algorithm that returns the decision parameters (S_t)

Through this section we will go on detail on how the demand was simulated, how the routing algorithm works for our simulation, and what assumptions are taken.

5.1.1 Learning environment assumptions

The main difference between the learning environment and the actual company operations is that our reinforcement learning agent will work under an artificial city. This artificial city will help both for generating realistic demand, and for calculating distances between orders and the warehouse during the routing algorithm.

The city will be 10x10 km size and divided into 4 different areas: Residential area 1 (R1), residential area 2 (R2), city center (C), and industrial area (I). For the implementation in the code, the smallest distance between points will be 10 m, thus the grid will be 1000x1000 units. In Figure 5-1 we have sketched how the grid is divided between the different areas, as well as the depot location.

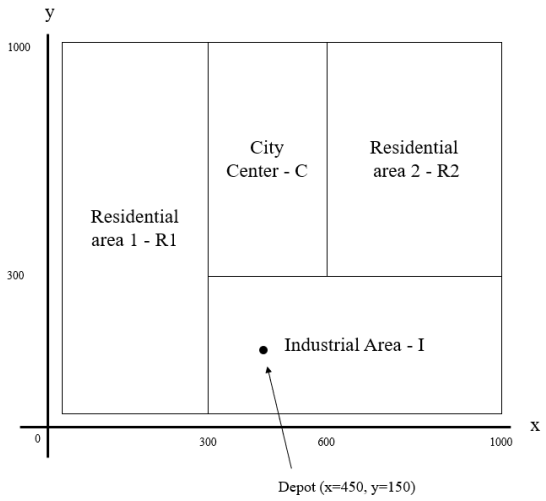

Figure 5-1 - Grid view of artificial city.

Generating demand

Despite the artificial city size being similar in size to Münster (where actual operation occurs), the behavior of the demand in the artificial city is different from the actual demand patterns the company experiences, as demand analysis was outside of the scope of the research.

We will then assume around 1000 orders are arriving each day, divided across the 4 different areas, each one with its own independent demand pattern behavior. Also, we will assume that a customer has equal chance to be in any point inside the area, this if there are 4 orders at a given point in time coming from the city center, 4 random coordinates will be drawn at random. In section 5.1.2 we will go more in detail on the demand generation process.

Routing algorithm

We will assume that the routing algorithm will work with an N_D number of drivers. The capacity of each driver will be equal to 50 loading units, and they will drive at a steady speed of 50 km/h in the artificial city. Each driver will also have a maximum route duration of 2 hours, which translates to 24 intervals of 5 minutes (t). Each driver also will be able to carry any number of different orders, as long as the load does not exceed 50 units, and the route duration 2 hours.

Note that, overall, the learning environment does not need to adjust to the actual situation of Flaschenpost (same demand, or same number of drivers) it just needs to create a realistic scenario where the agent could learn. Later, the Q-learning agent's policy will be compared with a policy derived by intuition (similar to how Flaschenpost currently determines their open/close policy). By comparing both policies we can understand the potential of Q-learning for this problem.

5.1.2 Generation of daily demand

One of the challenges for creating a simulation environment is to create realistic demand patterns. For our case, we have selected to simulate demand utilizing Non-Homogeneous Poisson Process (NHPP). The difference between a NHPP and its homogeneous counterpart is that the distribution between two different point in time changes, by changing the arrival rate parameter (Ross, 2009).

The Poisson distribution then, has a parameter $\lambda_{k,t}$ that indicates the average number of orders that arrive from area k given a stage t. Thus:

$$C_{k,t} \sim Poisson(\lambda_k(t)\,) \tag{5.1}$$

Where $C_{k,t}$ is a random variable representing the number of orders arriving at area k at stage t.

Following the NHPP, the arrival rate $\lambda_{k,t}$ will change depending on t. For our case, we will derive this value using a normal distribution. We need then to define the density function of the normal distribution.

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{(x-\mu)^2}{2\sigma^2}} \tag{5.2}$$

For each given area k, we will determine a peak arrival rate $\lambda_{k,p}$, a base rate $\lambda_{k,b}$, a peak stage $t_{k,p}$ and a peak width b_k. From these parameters and 5.2 we can derive the following function for $\lambda_{k,t}$.

$$\lambda_{k,t} = \lambda_{k,b} + \left(\lambda_{k,p} - \lambda_{k,b}\right) * e^{-\frac{1}{2}\left(\frac{t-t_{k,p}}{b_k}\right)^2} \tag{5.3}$$

In this case, t is the current interval, t_p is when the interval when the peak is reached, and width is 2 times the std deviation. When $t = t_p$, then $\lambda_t = \lambda_p$.

Each area has the arrival rate changing differently, with a different base and peak average arrivals. When combining the different NHPP of the 4 areas we arrive at the desired demand pattern.

To guarantee that there is a consistent average number of orders arriving to each of the areas, the distribution of arrival rates will also contain a scaling factor ρ, where:

$$\rho_k = \frac{Expected\ daily\ orders\ from\ area\ k}{\sum_t \lambda_{k,t}}, \qquad \forall k \tag{5.4}$$

Thus, the updated rates per area $\lambda^*_{t,k}$ are as follows:

$$\lambda^*_{t,k} = \lambda_{t,k} * \rho_k \tag{5.5}$$

Table 5-1 includes the set-up we have selected for the demand pattern of each of the areas k.

Area (k)	**Peak interval ($t_{p,k}$)**	**Peak rate ($\lambda_{p,k}$)**	**Base rate ($\lambda_{b,k}$)**	**Width factor (v_k)**	**Expected daily orders**
R1	110	12	2	4	300
R2	90	9	4	6	300
C	72	8	4	7	250
I	25	10	2	5	150
			Expected orders per day:		**1000**

Table 5-1 - Demand parameters per area k

To serve as an example, Figure 5-2 shows the demand pattern of area I for a random day. The yellow line represents how $\lambda_{I,t}$ varies depending on the stage. The blue line represents the number of orders $C_{I,t}$ arriving at each stage.

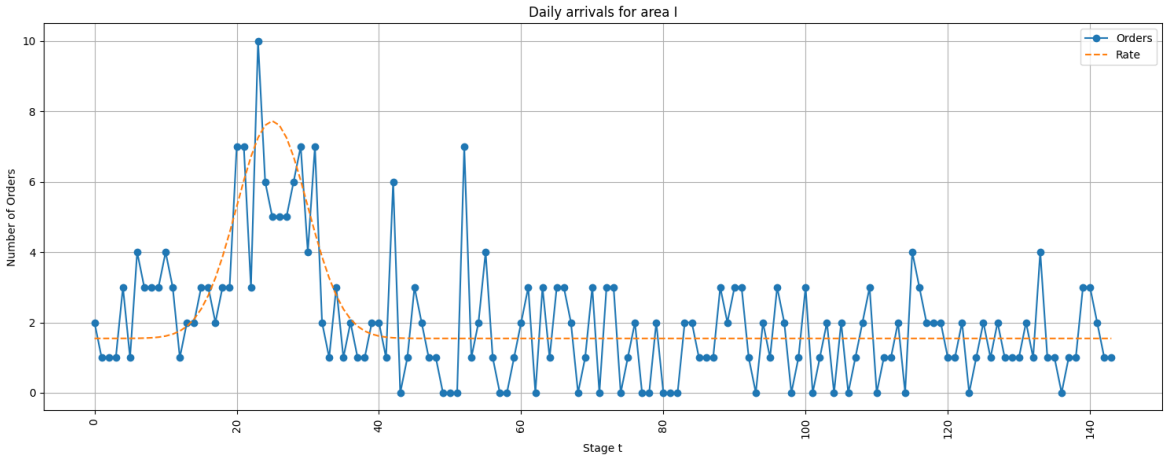

Figure 5-2 - Example demand pattern for industrial area I

For a given point in time, the total number of orders arriving is then calculated as:

$$C_t = \sum_{k \in Areas} C_{k,t} \tag{5.6}$$

Note that, as discussed on the mathematical model, each order contains a series of parameters, out of which the coordinates and the order load is dependent on the area. When generating the daily orders at the beginning of the episode, a random point inside the area is selected from a uniform distribution. The unit load is selected from a random normal distribution, with a given average and deviation.

Table 5-2 indicates how is each parameter is computed depending on the area, for a random order o coming from the area k.

Area (k)	**Order x coordinate (o_x)**	**Order y coordinate (o_y)**	**Order load (o_{load})**
R1	$U(0,299)$	$U(0,1000)$	$Normal(15,7)$
R2	$U(600,1000)$	$U(300,1000)$	$Normal(15,7)$
C	$U(300,599)$	$U(300,1000)$	$Normal(10,4)$
I	$U(300,1000)$	$U(0,299)$	$Normal(20,9)$

Table 5-2 - Distribution of coordinates a unit load per order given area k

Note that all values are rounded, to ensure that the coordinates and the load units of each order are integer. The order load is bounded to a minimum of 1, to make sure that there is no order with a 0 (or negative) unit load.

For a given area k, a set $W_{k,t}$ of orders is then generated using those parameters, containing all the orders ($o_{k,t}$) arriving at each stage t:

$$W_{k,t} = \{o_{k,t}^1, o_{k,t}^2, \dots, o_{k,t}^{C_{k,t}}\} \tag{5.7}$$

$$W_t = \bigcup_{k \in Areas} W_{k,t} \tag{5.8}$$

Thus, the incoming set of orders at a given point in time are:

$$w_t(a) = \begin{Bmatrix} W_t, & if\ a = 1 \\ \emptyset, & if\ a = 0 \end{Bmatrix} \tag{5.9}$$

5.1.3 Routing algorithm

We will build a routing algorithm based on a Nearest-Neighbor Algorithm (NNA). This type of algorithm creates a route by starting at a given customer and inserting (or "visiting") the nearest e nearest customer to the last one visited (Gutin et al., 2001) . The distance between two customer's orders (o^1, o^2) is calculated using Euclidean distance (Equation 5.10)

$$d(o^1, o^2) = \sqrt{(o_x^1 - o_x^2)^2 + \left(o_y^1 - o_y^2\right)^2} \tag{5.10}$$

Once the basic routing algorithm has been defined, we can merge it with the problem definition to explore how can an NNA be implemented into our simulation environment by adding different constraints of driver capacity and priorities when generating routes. The following Figure 5-3 illustrates the algorithm starting at a given stage.

Routing algorithm implementation

Generate routes.

Obtain O_t, D_t from previous stage, a from ε-greedy policy (Figure 4-2)
Sort O_t by min o_{time_stamp} and D_t by min $d_{t\_remaining}$
Loop for d *in* D_t:
    Loop for o in O_t
        If $o_{assigned} = 0$ and $t < o_{max\_lateness}$
            Update d parameters with o.
            $o_{assigned} = 1$.
            $o = starting\_order$.
    Exit loop.
    Loop until no suitable o can be found.
        Find out $new\_order \in O_t$ such that:
        $$new\_order = \underset{o \in O_t,\, o_{assigned}=0}{\text{argmin}} \; d(starting\_order, o)$$
            If new_order insertion is possible for driver d
                Calculate $new\_order_{e\_delivery}$
                If $new\_order_{e\_delivery} < new\_order_{max\_lateness}$
                    Update d parameters with new_order
                    Update new_order_p given $new\_order_{e\_delivery}, t$
    $starting\_order = new\_order$.

Calculate state S_{t+1}

Using D_t, O_t, calculate $S_{t+1} = \{\ell_t, x_t, \mathcal{p}_t, m_t, n_t\}$, following section 4.2

Calculate reward R_t

Obtain $w_t(a)$ from demand generator, given action a taken
Calculate R_t

$$R_t = R(a_t, S_t, t) = r_a * w_t + \sum_{o \in O_t,\ o_p=1} (r_l * o) \sum_{o \in O_t,\ o_p=2} (r_n * o)$$

Update O_t, D_t

For o *in* O_t
    If $o_{assigned} = 1$ or $o_p = 2$
        Remove o from O_t
If $a_t = 1$
    Append w_t to O_t
For d *in* D_t
    If $d_{t\_remaining} = 0$
        $d_{t\_remaining} = d_{length}$.
    $d_{t\_remaining} = d_{t\_remaining} - 1$.
    Reset drivers:
    $d_{load} = 0, d_{length} = 0, d_{orders} = \{\}$.
$O_{t+1}, D_{t+1} = O_t, D_t$.

Figure 5-3 Iteration of the routing algorithm given stage t.

On the "Generate routes" part of the algorithm there are a series of variations from the classical NNA. Firstly, the route generation won't start in a random order, but will start in the oldest order from the O_t set. When inserting new customer's orders into the route, we will also check for "suitability". This implies that the order can only be added to the route if it won't exceed the total capacity of the driver d, both in load and maximum route length.

At the beginning of each episode ($t = 0$), following the routing algorithm assumptions (5.1.1), O_0 and D_0 will be initialized. O_0 will start as an empty set {}, as we will assume that orders won't arrive until $t = 1$. Regarding $D_0 = \{d^1, d^2, \dots, d^{N_D}\}$, where N_D is the total number of drivers. Each d^n represents one driver, and its parameters are initialized as shown by Figure 5-4 (following 5.1.1):

Initialization parameters for $D_0 = \{d^1, d^2, \dots, d^{N_D}\}$	
$d_{load} = 0$	$d_{\max\_length} = 12$
$d_{max\_load} = 50$	$d_{orders} = \{\}$
$d_{length} = 0$	$d_{t\_remaining} = 0$

Figure 5-4 - Initialization parameters for D_0

5.2 Q-learning implementation

This section follows from formalization of the Q-learning algorithm in Section 4.3.

5.2.1 Implementation as code

For the implementation of both the routing algorithm and the RL, we decided to use the gymnasium Python[2] library. Among other reasons for selecting Python as the programming language is due to its simplicity, as well as the considerable large documentation available compared to other programming languages.

Python also allows us to use the gymnasium library, which greatly simplifies defining states, stages, learning environment, and the learning agent. Thus, the code will be divided into 4 main modules: The RL Agent, the RL Environment, the Routing algorithm, and the Demand generator.

The RL Agent and RL Environment will be the ones that make the most use of the gymnasium library, as it gives a framework for defining the steps, stages, the terminal stage, the reward, and the transitions.

The RL Environment module will connect the gymnasium with the Routing algorithm and Demand generator modules, as this module defines how the environment behaves. At the beginning of each episode, the RL Environment module will call the Demand Generation module to determining the demand of that episode. After that, at each stage t during the episode, the RL Environment will give the state information to the Routing algorithm

[2] For more information of how the gymnasium library works, visit https://pypi.org/project/gymnasium/.

module, which processes it and returns the reward and the follow-up stage (as described in Section 5.1).

Besides gymnasium, we will also be using other important Python modules, such as "csv" (used to export the data) or "NumPy"[3] (used to generate random values and complex matrix operations). The "NumPy" module will also be used for the "seeding" of the daily demand.

As mentioned in the previous chapter, the demand is drafted from a NHPP. At the beginning of the agent's training, we will select a seed for generating the demand. This seed will determine the chain of daily demand observed by the agent across multiple episodes. This means that, if two agents are trained with the same seed for 1000 episodes, the demand experienced by both agents across the training is identical (same number of orders, same location of the orders, same order load, etc.)

This approach will help us to i.e. compare two different policies by testing them across identical 100-day periods. We will also experiment with different seeds, to prove the effectiveness of the Q-learning agent across multiple possible instances of the daily demand.

5.2.2 Implementation of the exploration

In reinforcement learning, one of the most important aspects is to determine the balance between exploration and exploitation. For our specific problem, we decided to use ε-greedy exploration, where for each episode the agent has a $1-\varepsilon$ probability of choose the action with the highest know $Q(s,a)$, and a ε probability of choosing an action at random.

For each following episode, the ε is reduced according to:

$$\varepsilon_{episode+1} = \varepsilon_{episode} * \kappa \tag{5.11}$$

Where κ represents the epsilon decay rate for the algorithm, which can be computed following different methodologies. For our specific training set-up, we will stablish an initial maximum epsilon (ε_{max}) and an ending minimum epsilon (ε_{min}). The ε will decay exponentially through the E episodes of the training following:

$$\kappa = \sqrt[E]{\frac{\varepsilon_{min}}{\varepsilon_{max}}} \tag{5.12}$$

Figure 5-5 graphically represents the ε decaying through a training of 1000 episodes, when $\varepsilon_{max} = 0.99$ and $\varepsilon_{min} = 0.01$.

[3] For more information of how the NumPy library works, visit https://numpy.org/doc/stable/.

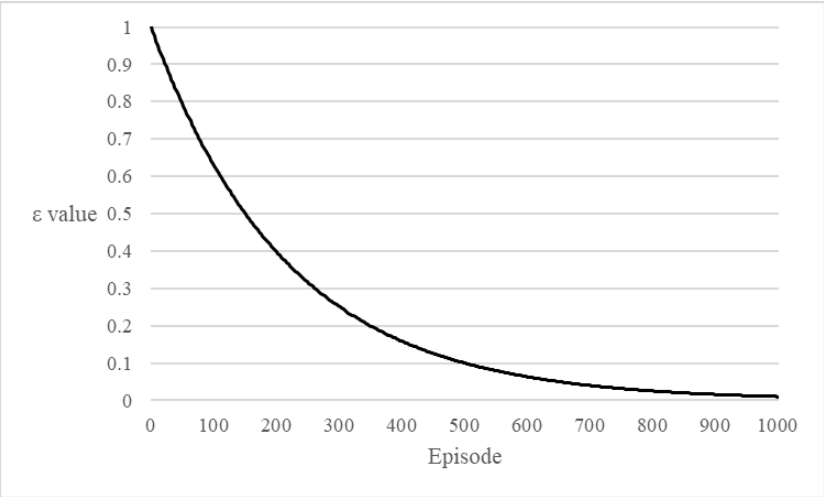

Figure 5-5 - Graphical representation of the ε decay after 1000 episodes.

5.3 Conclusions

From this chapter we can bring some conclusions regarding the implementation of the Q-learning agent for the problem at hand. First, we discussed the implementation of the learning environment. The demand will be generated using a NHPP among 4 different areas. This will allow the agent to experience pseudo-realistic demand. The routing will use the NNA and will ensure to compute the decision parameters (state) in accordance with the model presented in Chapter 4.

Regarding the Q-learning implementation, we used Python (mainly the gymnasium module) for implementing the model. We also determined how the exploration will behave across multiple episodes (κ).

6 Numerical experiments

On Chapter 5 we have formalized the Q-learning algorithm, but we will do some numerical experiments to tune the learning agent and compute the policy. Through this section we will determine the hyperparameters of the Q-learning algorithm, obtain the decision-making policy and compare the performance with other policies to explore what is the potential improvement. With this objective, a series of numerical experiments will be done. Section 6.1 discusses the experimental set-up, Section 6.2 describes the different experiments, and Section 6.3 shows the numerical results.

We will be addressing the performance of the Q-learning agent in terms of "cumulative reward per episode". This refers to the cumulative reward experienced during one episode ($R_{episode}$). We will refer to "learning curves" as the development of $R_{episode}$ across multiple episodes (as the exploration of the agent decreases).

6.1 Experimental set-up

During Chapter 5 we have discussed the main implementation procedure of the Q-learning algorithm. This implementation includes information on how the demand and the routes are generated in the simulation environment. We also connected the simulation environment with the Q-learning algorithm itself. For running experiments on the Q-learning algorithm we need to stablish the experimental set-up. Table 6-1 introduces the set-up values for all experimental parameters considered.

Experimental parameter	Default Value
Reward for accepting an order	1
Reward for order arriving late	-0.5
Reward for not delivered an order	-5
Average orders arriving per day	1000
Number of drivers (N_D)	15
Discount factor (γ)	0.9
Number of episodes (E)	1000

Table 6-1 – Starting experimental set-up for Q-learning agent

Some of these values will be altered through experimentation, to test the Q-learning agent and obtain significant results.

It is important to highlight the number of drivers (N_D). This parameter determines the capacity of the company in the simulator. If the agent counts with an infinite large number of drivers, then the optimal policy may just be to always leave the store open. The ratio demand-capacity can also be adjusted by altering the demand patterns presented in Section 5.1.2, but for our experimentation we will only adjust the number of drivers.

Regarding the reward, it was selected that an order accepted would increase by 1 the reward, an order arriving late would reduce it by 0.5 and an order not arriving will reduce it by 5. These values are assumed to be representative of how the cost inquired by the company

would work, as not delivering to a customer due to lateness is "worth" the same as delivering to 5 other customers[4].

Due to the nature of this problem, decisions taken at stage t may not have an impact until later in the day, as accepting an order will only have a positive/negative reward once the order is (or is not) delivered. For that reason, the discount factor γ has also been fixed to 0.9 as a high discount factor implies that future rewards will have a higher impact when approximating $Q(s, a)$ at a given t.

Note that, due to the size of the state space, the Q-learning agent will not get to test all state-action pairs, especially the more extreme scenarios. If we are in an extreme state that was not experienced during the training, the "safe" action would be to close the store. For that reason, we decided to set the default action to be 0 if the agent has no knowledge of the state-action pair.

6.2 Experimental description

Through this chapter, we will perform different experiments in order to adjust the Q-learning algorithm and obtain meaningful results for the decision-making policy. This section summarizes the different experiments, as well as the questions each experiment aims to answer.

The experiments for this research are mainly divided into two categories: algorithm-tuning experiments and problem-specific experiments. The first experiment category aims to tune and adjust the algorithm for experimentation, as well as explore how the learning behaves depending on the seed. The second experiment category aims to understand how the policy behaves depending onN_D, and how it compares to different mark-up policies.

6.2.1 Algorithm-tuning experiments

The experimentation needs to start by tuning the hyperparameters, showing whether the agent converges to an optimal reward, and determining the number of episodes needed for the training. For these experiments we will set $N_D = 15$.

Hyperparameter tuning.

The numerical experiments will start by tuning the hyperparameters to improve the learning curve of the reinforcement learning agent. These tests will be done in accordance with the base set-up described previously. When implementing Q-learning, the usual hyperparameters tuned are the learning rate α and the exploration parameter ε. Despite that, as introduced in Section 5.2.2, our ε will have a constant decay κ (dependent on the number of episodes of the training). We will also fix $\varepsilon_{max} = 0.99$ and $\varepsilon_{min} = 0.01$ across all experiments.

[4] These proportions were obtained after asking a company expert, to make sure the values are representative

The hyperparameter we will be exploring is then the learning rate α. We will perform a series of experiments changing its value and explore how the learning varies across 1000 episodes.

We will then look to answer the following question: what learning rate should be used for the Q-learning agent?

Seeding experiments

Once the hyperparameters have been found, we will explore the effectiveness of the Q-learning agent by changing the seed used to generate the demand. The main reason for this experiment is to prove that the Q-learning agent can determine the optimal policy regardless of the demand pattern experienced through the training.

Following the implementation, the demand of the different episodes across the training is determined by the seed used. We will change this seed changes across different experiments, determining the actual performance of the Q-learning. We will target to answer the following question: how does different seed affect the Q-learning agent learning performance?

We will train the agent for 1000 episodes, changing the seed on each run. Then we will draft conclusions by comparing each one of the learning curves.

Episode testing

Once the Q-learning agent has been determined to work across different daily demand instances (by changing the demand generation seeds) we need to determine what is the optimal number of episodes for training the agent.

Thus, this section will be tailored to answer the following question: how much does the policy improve depending on the number of episodes used for training? And how many episodes should we use for training our agent?

We will train the agent for episodes ranging from 10 to 15000 episodes (10,100,1000,10000, 15000). From each of the experiments, we will compare the learning curves of each one of the trained agents. After determining the policy of each agent, we will do a 100-day experiment using the same seed, to see how the performance compares. It will include the average episode reward, the reward variation, and the time taken to finalize its learning.

6.2.2 Problem-specific experiments

After performing the algorithm-tuning experiments, we will have a good overview on what are the training parameters, number of episodes and whether the algorithm converges to an optimal episode reward. As mentioned in Section 6.1, the main factor affecting the capacity of the company in the simulation is the number of drivers (N_D).

The problem-specific experiments will give the output of the decision-making policy found by our Q-learning agent for two different scenarios, $N_D = 15$ and $N_D = 10$. Note that we also considered the possibility of $N_D = 20$ but, as shown in Appendix 9.2.1, this scenario is quite like $N_D = 15$. The main reason may be because 15 drivers (or more) lead to all daily orders being satisfied, and thus the overall daily reward converges to the same number.

Besides exploring the output, we will also compare its performance on both scenarios with different mark-up policies. We will aim to answer two main questions with this problem-specific experiments:

- How does the Q-learning agent policy behave given a state and a different N_D?
- How does the performance of the Q-learning policy compare to different policies?

For the second questions, we will compare the Q-learning policy with three different policies: Random decision, always open and company-derived policy. The first two are naïve policies, where the decision will be random and always open respectively.

For the company-derived policy we will set a series of constraints that, if met, the company will close the store. The objective of this policy is to replicate (in a simplified version) the current decision-making heuristic of the company's decision-making process. Under this policy, the decision will be as follows:

$$\pi(s_t) = \begin{cases} 0, & if\ (L_t \geq 8) \vee (X_t \geq 8) \vee (O_t \leq 5) \vee (P_t \leq 3) \vee M_t \geq 10 \\ 1, & o.w \end{cases}$$

6.3 Numerical results: Algorithm-tuning experiments

6.3.1 Hyperparameter tuning

The first numerical experiments were focusing on the hyperparameter tuning: adjusting learning rates. Note that, each of the experiments were done by altering the learning rate and using the base set-up from Section 6.1.

We also have averaged the rewards across bins of 10 episodes. I.e. the first reward point is the average reward of the first 10 episodes.

The learning rate and discount factors will determine in which way the new information updates the previous value associated with a state-action pair. Figure 6-1 represent the learning curve of the Q-learning agent depending on different α.

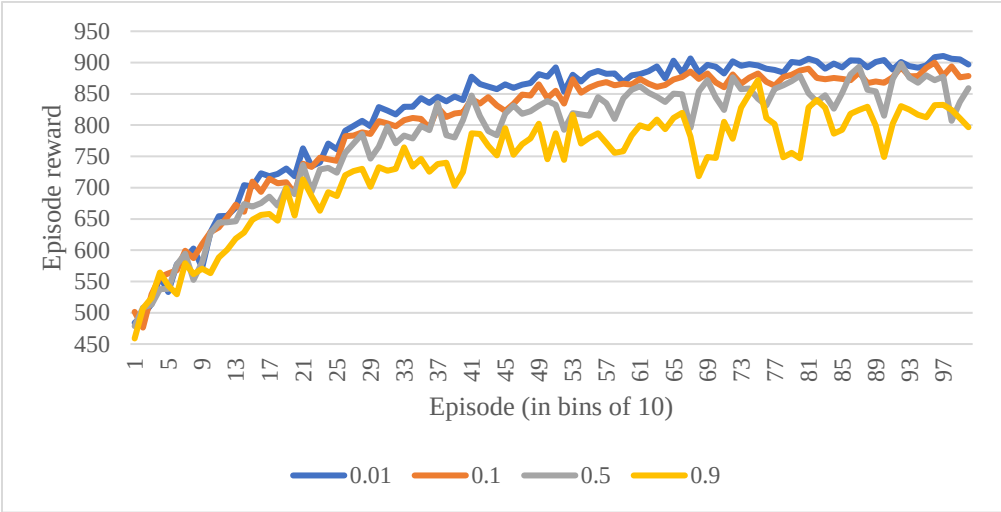

Figure 6-1 - Q-learning agent learning curve across different α

In Figure 6-1 each line represents a different α: 0.01, 0.1, 0.5 and 0.9. From the figure it becomes clear that the smaller learning rates yield a higher (and more consistent) result. In this case, both $\alpha = 0.01$ and $\alpha = 0.1$ yield the highest reward (the orange and the blue line respectively). Despite $\alpha = 0.01$ yielding a higher output overall, a higher learning rate greatly reduces the overall run time of the training. The 1000-episode training was completed in 2 minutes 53 seconds, compared to the 3 minutes 50 seconds when using 0.01. For that reason, for the remainder of our experiments, we will use 0.1 as the best fit for the algorithm.

6.3.2 Seeding experiments

After determining the α, we performed 5 runs changing the seeds of the demand generation. This implied that, across each one of the runs, the agent faces different number of orders, order load and order coordinates, thus the decisions may need to be different. Figure 6-2 shows how the cumulative rewards varied through each one of the runs for the different demand seeds.

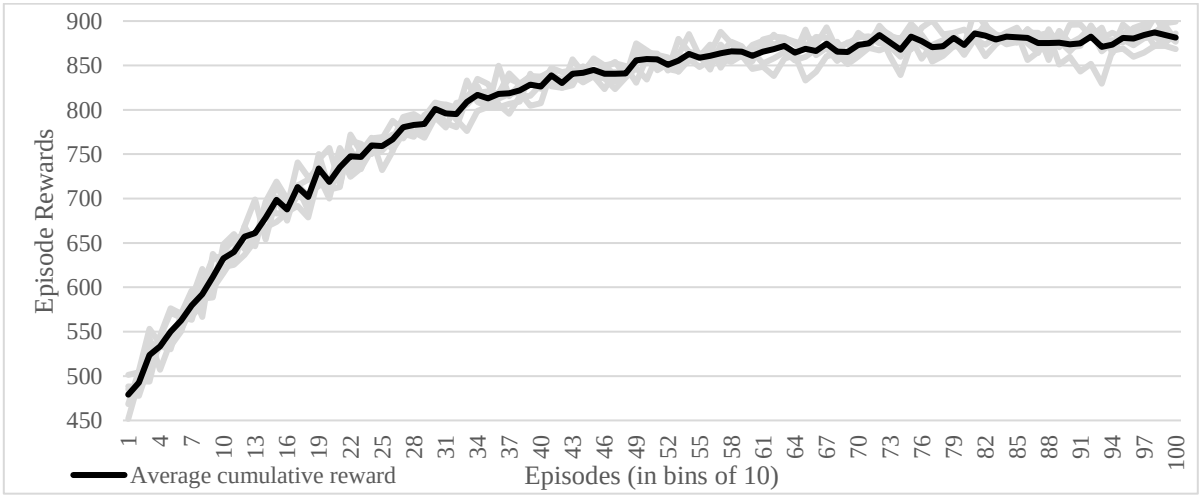

Figure 6-2 - Leaning curves across different demand seeds.

We can observe how, regardless of the seed, the Q-learning agent always converges to a cumulative reward. In gray we have the learning curves for each one of the seeds, and in black we show the average across all the runs. The reason some of the seeds have a higher end reward is probably because the seed of that run lead to an overall higher demand.

6.3.3 Episode testing

As mentioned, we decided to train 6 different agents, using increasingly more episodes for its training (10, 100, 1000, 5000, 10000 and 15000). The found policies for each one of the agents were used during a 100-days simulation. Table 6-2 includes the observed sample average daily reward, sample standard deviation and time taken for complete the Q-learning agent.

Episodes per training (E)	Training time (h:mm:ss)	Average sample daily reward (μ_R)	Sample standard deviation (σ_R)
10	<0:00:00	748.88	22.91
100	0:00:12	831.07	23.65
1000	0:02:52	901.45	29.02
5000	0:17:04	912.83	33.68
10000	0:34:01	924.93	23.58
15000	0:52:48	918.99	43.20

Table 6-2 - Comparison results of the learning with increasing number of episodes.

We can see how, when increasing the number of episodes for the training of the agent, the average daily reward increases. This can mainly be attributed to a higher exploration of the environment, and thus a higher knowledge of the optimal decision at each state. It is also noticeable that, after the 10000-episode threshold the overall daily reward average decreases, and a higher standard deviation is yielded. This may indicate that the agent has already found the best policy, and thus more episodes won't increase the average episode reward. We then need to balance the learning possibility with the time consumption of the training. For that reason, we will train our agent with 10.000 episodes, as demonstrates a balance between reward and run time.

Note that there are multiple factors potentially affecting the run time for each one of the trainings. In Appendix 9.1 there is included a complete list with the hardware and software specifications used. Besides those, the run time may also have been affected by some other factors. Despite that, all agents were trained sequentially, so the run time proportions is still significant.

6.4 Numerical results: Problem-specific experiments

Based on the previous experiments, we can update our Q-learning agent set-up. Firstly, we will use 0.1 as our learning rate (from hyperparameter experiments). We will also set the number of episodes for the training to 10000 (from the episode experiments). Using this learning rate and experimental set-up, we trained the Q-learning agent, both for 15 drivers and 10 drivers, and derived a policy for both N_D values.

6.4.1 Q-learning policy for $N_D = 15$

We first trained the agent for the $N_D = 15$ sscenario. Using the policy found by the agent, we run a day of operations on the 15-driver scenario, registered all the states encountered and the decision taken at each point in time. For visualization purposes of the policy, Figure 6-3 shows how three different values of the state space develop through the day.

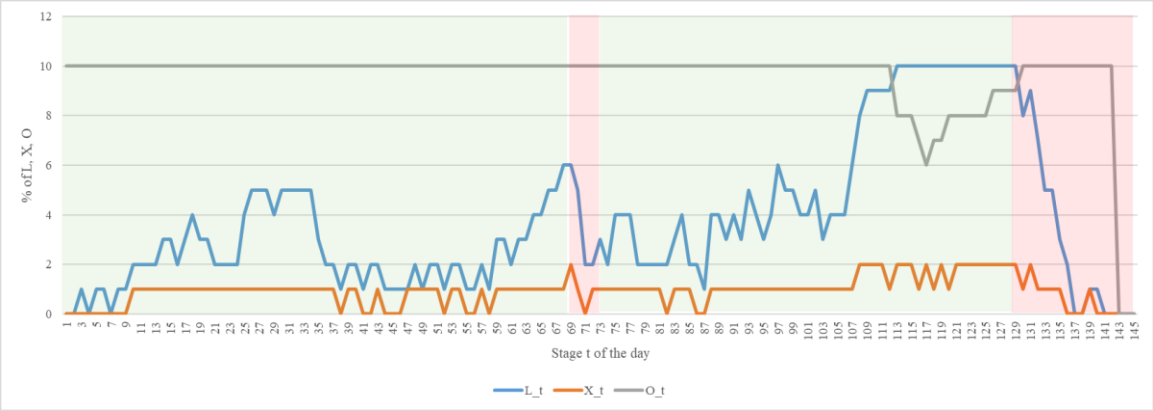

Figure 6-3 -Opening/closing action with respect to $L_t, X_t, and\ O_t$

The blue, orange and grey line represent $L_t, X_t\ and\ O_t$ respectively, as the load percentage, route length percentage and order assignment percentage (following from 4.2). In green we represent the periods when the store was open, and in red we represent the periods when the store was closed.

We can observe then that the main period when the store was closed was towards the end of the day. The agent then decides to close the store when the sore reaches the end of the day and most of the drivers are being utilized (very elevated $L_t = 10$). It will then leave the store closed until the end of the day. There is also a moment before the day where the agent decided to close the store when the $L_t = 6$. The store is kept close for three stages before re-opening.

We can also observe that, on the busiest moment of the day, L_t reaches a maximum (10). Despite that, the trip duration utilization (X_t) never goes higher than 2. This gives some insight of how the environment behaves, as all the drivers are completely full on those points in time, but only 2/10 of the 2h total trip duration utilization is used in average.

Figure 6-3 it is just a simulation of a given day. On another day the states may have affected differently due to randomness. For that reason, we decided to create a heatmap, where the policy is averaged over one state parameter: loading utilization (L). Figure 6-4 is a heatmap that represents what is the decision that the policy makes over each possible time interval $n(t)$. Note that the heatmap cannot be given over t, as the state parameter representing "time" in the Q-table is $n(t)$ (following Section 4.4).

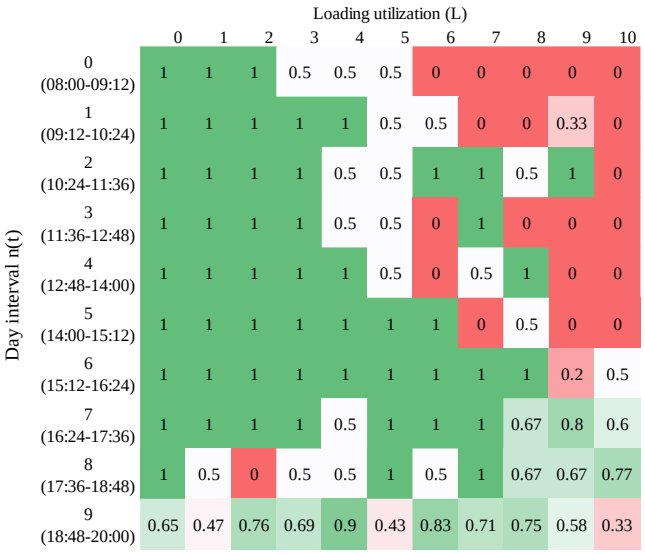

Figure 6-4 - Day interval decision heatmap averaged over the loading utilization for $N_D = 15$

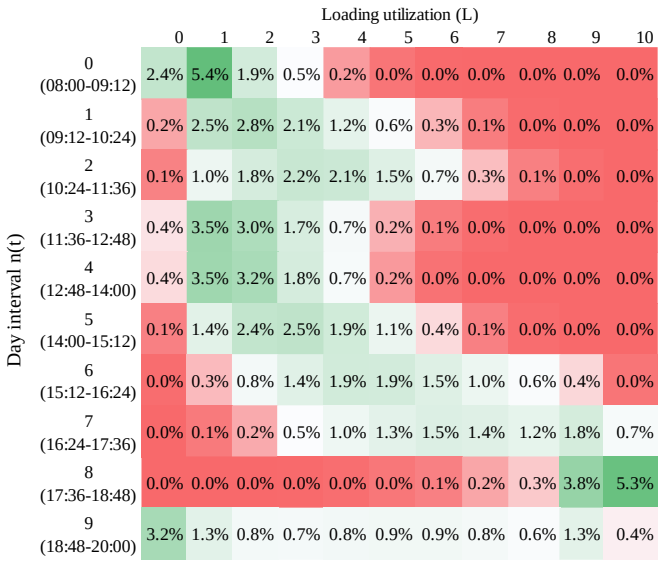

Figure 6-5 - Percentage of time the agent visited each loading utilization after a 100-day simulation for $N_D = 15$

The values on Figure 6-4 represent the average decision when the agent encounters a given L- $n(t)$ pair. This means that, I.E. if the loading utilization is 0 during the first day interval, then the store will always be kept open. In the heatmap, the green color is assigned to the highest value, while red is assigned to the lowest.

We also associated each interval of time $n(t)$ with respect to which point of the day it represents (remember $n(t) \in \mathcal{N} = \{0,1,\dots,9\}$ and Flaschenpost store is open from 8:00 until 20:00).

It is quite noticeable how, in some cases, the store decides to open counter intuitively. This is the case for, i.e. when the loading utilization is at 9, during the 2nd day interval, as the store is always kept open. The main reason for this potential flaw is that not all states are visited the same number of times, and thus the policy may be less trained on that state.

For that reason, Figure 6-5 shows the percentage that the agent spent on each state after a 100-day simulation of the 15-driver scenario using the derived policy. We can see that through the day the load is increased periodically (at interval 0, the average load is between 0 and 2, and towards the end of the day it shifts to 9/10). We can conclude then that the agent is mostly opening the store through the day except on the last interval. Here each possible loading utilization values is found through the 100-day simulation; thus, the decision open/closes varies more. This indicates that, at the interval 9, the decision is more complex and depends on other parameters of the state.

We also included in Appendix 9.2 a heatmap indicating what is the average decision the agent took after the 100-day simulation. This shows how the policy behaved in the experimental setup at hand. The figure is included for both $N_D = 15$ and $N_D = 10$.

6.4.2 Q-learning policy for $N_D = 10$

After obtaining the policy for $N_D = 15$ we decided to reduce the capacity of the environment by reducing the number of drivers. This made the overall scenario more challenging, as the previous policy of maintaining the store open until the last interval may no longer be optimal.

We also used the same averaged heatmaps visualization tool from the previous section. Despite that, we will not compare the time interval with the loading utilization, but with the order punctuality. On this scenario, the drivers are over 80% of the time with a loading utilization of 9/10 (see Appendix 9.2), thus the punctuality gives a better insight on the actual decisions being made. Both the averaged decision heatmap and state percentage heatmap are shown in Figure 6-6 and Figure 6-7 respectively.

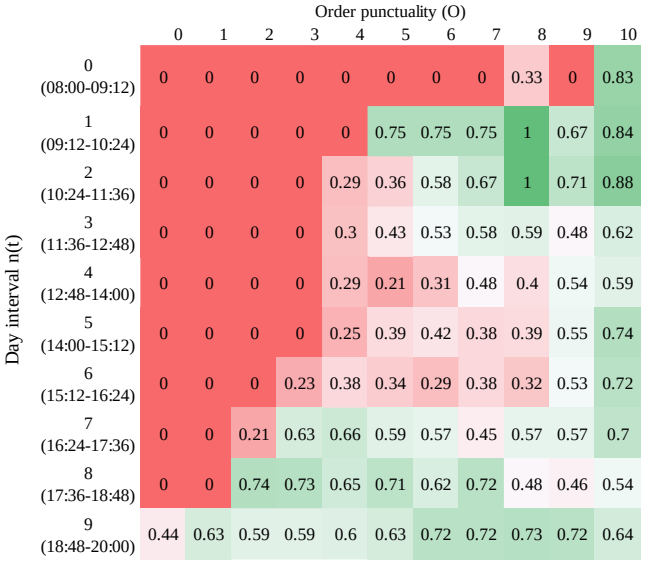

Figure 6-6 - Day interval decision heatmap averaged over the order punctuality for $N_D = 10$

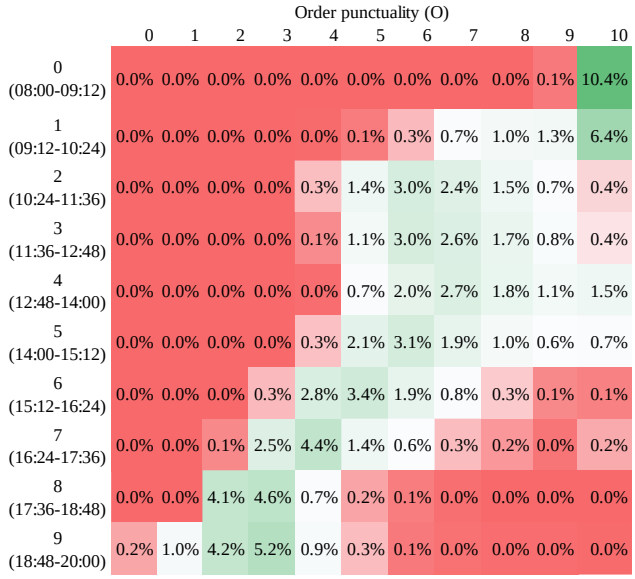

Figure 6-7 - Percentage of time the agent visited each order punctuality state after a 100-day simulation for $N_D = 10$

We can observe how the policy tends to leave the store open the higher the order punctuality is. We can also observe how the order punctuality decreases as we advance on the day, reaching the lowest point on the last interval. On this last interval, the decision is also not as clear as the beginning stages. The open/close decision is closer to 0.5, which indicates that multiple state parameters need to be looked at, rather than just the order punctuality.

6.4.3 Comparison with different policies

Once the Q-learning derived policy was found for both the 10 and 15 driver scenarios, it is now relevant to compare it with the random policy, always open policy and the company-derived policy. For each of the policies, we have runed 100-days simulation changing the number of drivers and registering the daily reward. Figure 6-8 and Figure 6-9 compare the daily reward results between the 4 policies for the two driver scenarios.

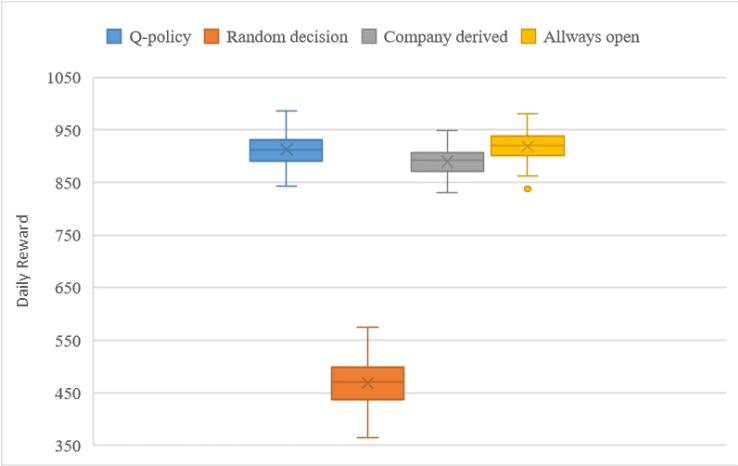

Figure 6-8 - Box and Whisker plot comparing the daily reward of the Q-learning, random, always open and company derived policies across a 100-days simulation when $N_D = 15$

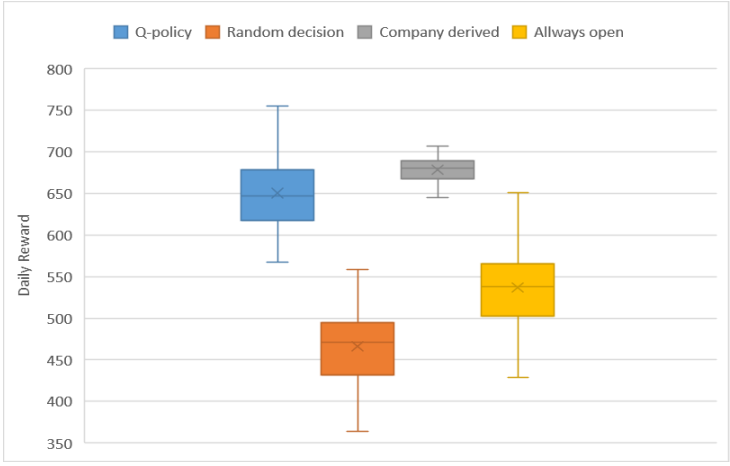

Figure 6-9 - Box and Whisker plot comparing the daily reward of the Q-learning, random, always open and company derived policies across a 100-days simulation when $N_D = 10$

From Figure 6-8 we get that the Q-policy and the always open policy obtain the overall highest reward with an average reward of **912.52** and **918.06** respectively. These results show that, for the 15-driver case, the best policy should be leaving the store open for most of the time. Both these policies show an improvement over the random policy (462.8 average reward) and the company derived policy (889.79 average reward). Overall, the Q-policy improves the reward by around 2.5%.

Figure 6-9 shows the results of both policies on the 10-driver scenario. We can now observe that the always open policy does no longer perform the best, thus is no longer optimal to keep the store open for most of the day. Now the company derived, and the Q-policy have a very similar performance. The company policy has the highest average reward, with a 678.12 average, while the Q-policy averages at 650.26. Despite that, the maximum reward experienced in any of the 100 days came from the Q-policy. This could indicate that the Q-policy follows a greedier approach for the decision-making, yielding some punctual higher results, but an overall lower daily reward average.

Either way, we observe that for both scenarios (10 and 15 drivers), the Q-policy shows learning progress and good overall results. This reinforces the validity of Q-learning as solution method for Flaschenpost sequential decision-making problem.

Table 6-3 compares the average value of each KPI in the state space over the complete day between the 4 policies for the 10-driver scenario, as well as the average decision (% of the day the store was open).

Policy	Average L	Average X	Average O	Average P	Average M	Average a
Q-policy	8.60	2.87	6.25	5.98	0.93	0.9
Random decision	2.27	0.61	9.65	9.65	0.00	0.5
Always open	8.75	1.46	5.41	5.04	2.18	1
Company derived	5.55	1.23	9.67	9.67	0.00	0.75

Table 6-3– Comparison of the average state of the Q-learning, random, always open and company derived policies after a 100-days simulation when $N_D = 10$

From Table 6-3 we can see how, in general terms, the Q-policy opens 90% of the time, while the company derived policy opens 75% of the time. This derives in, the order punctuality of the Q-policy being lower than with the company derived, and thus the M is higher (the tardiness of the Q-policy is higher than the company derived policy).

The Q-policy has a higher loading L and trip length X utilization compared with the company derived. This may make the Q-policy look worse than the company derived option but, as seen on Figure 6-9, the Q-policy also yields similar rewards. This leads to think that the Q-policy is using the resources better, as obtains a similar reward by utilizing more the drivers (higher L and X).

For sake of completeness, we also included in the Appendix 9.3 an overview of the average decision taken by both the company derived policy and the Q-policy across the whole day, as well as some extra figures comparing the 4 policies.

Note that this detailed comparison was not done for the 15-driver scenario, as the solution was more simplistic, and the store was left open the complete day until the last time interval.

6.5 Conclusions

This section helps us complete the answer of the following knowledge problem: How can a Q-learning algorithm be implemented to the decision-making problem? In this section we determined a learning rate of 0.1 and an ideal training of 10000 episodes.

We also get some insight into how the Q-learning policy takes decisions based on the utilization of drivers and order assignment. The main moment when the company is closed is when reaching the last section of the day, and driver utilization is especially high

Lastly, we also explored the following question: what is the policy's performance based on the KPIs? For a 10-driver set-up, when comparing Q-policy with the company derived policy, we can conclude that the Q-policy takes a greedier approach to decision-making, opening the store for longer and utilizing more the drivers. This leads to more orders arriving late (higher M). Despite that the reward are very similar between both policies, and even the Q-policy obtaining higher daily rewards at some specific moments.

7 Conclusion and Recommendations

This chapter contains the main conclusions of the research, and recommendations for the company. The chapter will be divided into 3 sections. Section 7.1 includes the important findings and conclusions with respect to the research question. Section 7.2 discusses the scientific contribution of the research and its limitations. Section 7.3 covers recommendations to the company and future research.

7.1 Conclusions

This research had the goal of developing a Q-learning agent for the decision-making process of Flaschenpost, as a subclass of an order acceptance/denial problem for incoming orders. The main research question formulated was the following:

How can a Q-learning algorithm be used for deciding whether to allow incoming orders?

To answer this research question, we drafted 5 knowledge problems following the MPSM framework. Each of these problems was addressed through the chapters of this theses, with the objective of answering the research question.

What is the current open-close strategy of Flaschenpost's online store?

Firstly, the current open-close strategy the company compute a series of parameters using their order routing algorithm, and later uses them for making decisions. These parameters indicate both the current driver utilization and the status of the routes (such as the order assignment or punctuality). The decision is taken every 5 minutes, as that is when the routing algorithm is executed. The decision will determine whether the customer has possibilities of placing "immediate orders" or not. This decision is currently based on gut-which could lead to not the best option being selected at each point in time.

What is the state-of-the-art for reinforcement learning and Q-learning, and how can it be applied to Flaschenpost case?

After exploring Flaschenpost situation, we have determined they're facing a sequential decision-making problem. We have decided to model the problem with the MDP framework and use tabular Q-learning as a RL solution method. This solution method allowed us to determine a policy for a simplified version of Flaschenpost problem.

We conducted a literature review for similar research papers implementing RL for sequential decision-making on the fields of logistics for FMCGs, which further supported the use of RL for Flaschenpost problem.

How can the company's sequential decision-making process be modeled mathematically?

We have then modeled the company's sequential decision-making within the MDP framework, including the relevant assumptions and simplifications to make sure that the problem can be solved, but still outputting relevant outcomes. Besides MDP modeling being

a necessary step for implementing Q-learning, this mathematical model could be improved, reducing the number of assumptions made.

How can a Q-learning algorithm be implemented to the decision-making problem?

Following from the previous section for implementing a Q-learning algorithm, we also designed a learning environment which replicates the behavior of daily demand, as well as utilizes a simple routing algorithm for getting the state parameters. We also stablished the number of drivers (N_D) to be the factor influencing the capacity of the simulation environment.

Besides of the implementation of the algorithm, on Chapter 6 we also discuss the hyperparameters and other important learning factors, which resulted in using $\alpha = 0.1$ and training the agents for 10.000 episodes. These tests were performed fixing $N_D = 15$.

What is the policy's performance based on the KPIs?

For assessing the policy performance, we performed experiments for understanding the policy behavior, and then comparing its performance with 3 other policies over a 100 day simulation, checking how the KPIs perform. The experiments were done for two environment configurations:15 drivers and 10 drivers available.

Starting with the 15-driver configuration, the policy leaves the store open through the day as long as the loading utilization does not reach excessively high values (mostly 8 or higher), and the store is only closed on the evening (between 18:48 and 20:00). With this policy and the given driver utilization, the environment does not reach high loading utilization until the second to last interval of the day (17:36 to 18:48).

On the 10-driver configuration, all the drivers are more utilized across the entire day. This makes the store be closed for more time than with the 15-driver configuration. From this set-up we can conclude that the store is more likely to be closed the more the order punctuality approaches 0. With this policy we still experience a decreasing order punctuality through the day.

The most interesting results come when comparing the Q-policy with the two naïve policies (always open and random decision) and with the company derived policy. For the 15-driver set-up, the Q-policy shows an increase of 2.5% on average daily reward with respect to the company derived policy. This led us to conclude that on this configuration, the company derived policy takes a more conservative approach.

The conservative trend is even more noticeable on the 10-driver set-up. Here, the Q-policy obtains an average reward of 650 compared to the 678.12 of the company derived policy. Despite that, the Q-policy obtains this reward by maintaining the store open more (90% compared to 75%).

Overall, the Q-policy shows an overall good performance on both scenarios, which further supports the potential of Q-learning for this decision-making process.

7.2 Discussion

7.2.1 Scientific contribution of the research

Yan et al. (2022) explores the increasing trend on applying RL methods for solving sequential decision-making problems on the fields of logistics and supply chain. Among the research found, most of them implement complex DeepQ learning in urban logistics decision problems. Despite that, there is no research that implements tabular Q-learning for an order acceptance decision-making problem.

Besides, we also included state space aggregation methods, as well as connecting the Q-learning methodology with a learning environment that simulates the behavior of an artificial city. We included very specific state space, that focus on giving information about the generated routes and the delivery driver utilization.

Overall, the research shows that it is possible to use (and obtain meaningful results and conclusions) of the optimal decision-making process of a real-time delivery company utilizing Q-learning on a simplified learning environment.

7.2.2 Research limitations

Despite the results, this research comes with some major limitations, mainly driven by the learning environment. The learning environment had to be built from scratch, which came with different challenges. We had to develop a demand pattern that can be executed multiple times, but that was not identical through different runs. We also had to create a routing algorithm that can output the same decision parameters used by the company, so certain conclusions could be extracted from the derived policy.

The challenges faced lead to significant simplifications of the main problem (as seen during chapter 2). These simplifications affect the application of the results to practice. For the policy to be implemented into the Q-learning agent, we would need to expand on the elements modeled, as well as take into account other uncertainties (traffic, warehouse loading, etc.).

We can also highlight the “difficulty” of the environment created, as different scenarios were not explored. A more difficult problem would also require to re-adjust the learning hypermeters accordingly or increase the minimum number of training episodes.

7.3 Future work and recommendations

This thesis introduced a Q-learning for decision-making on the fast-moving goods. Based on this last section of the research, we developed a list of recommendations for the company based on the experimentations results. We will also look at which future work can be done in the Q-learning agent algorithm, the simulation environment, and the numerical experimentation.

7.3.1 Recommendations to the company

- The main advice would be to stablish a strategy. As shown when testing the Q-policy, a good reward can be obtained by accepting more orders which may indicate that the current company's approach is too conservative.
- Overall, the Q-learning approach has been shown to converge to a good policy, so we recommend to the company to continue with this approach by, for example, increasing the complexity of the model.
- Substitute the demand generation based on a NHPP for one based on historical demand patterns. This will make the training set-up for the Q-learning agent more realistic.
- All Q-value-based algorithms work on the same framework, so utilizing another of those algorithms may be both easier to implement and give a more accurate policy. This could include, i.e. approximate the Q-values with a value function approximator.
- We would also advise to utilize the learning gained from the new Q-learning algorithm to develop an automated heuristic that can suggest/take decisions. This tool is constantly running through a day, and it can already take decisions based on the state-action policy encountered.

7.3.2 Future improvements

To address the limitations of the research and further refinement of the results, two main areas have been identified for future improvements: the Q-learning algorithm itself and the numerical experimentation.

Q-learning agent algorithm

- Further explore the reward system for an episode. It can be linked with the actual cost factor of accepting an order, arriving late, or completely missing the company inquires, instead of being arbitrarily selected.
- Implement the possibility of pre-orders, as it is one of the main simplifications done to the problem. These pre-orders would need to be implemented also as a parameter on the mathematical modeling.

Numerical experiments

- Explore how the Q-learning agent would behave with a different epsilon approach (i.e.: fixed epsilon decay), and whether the daily reward would converge to the same values.
- Explore different initializations of the Q-table, as they may affect the agent's exploration and the speed to which it converges to an optimal policy. An approach could be to initialize the Q-values by first running the environment with the company derived policy.
- Explore the 10-driver scenario even further by, i.e. outputting the actual number of orders that have been fulfilled on average on a given day, and how many of the orders accepted were ultimately not delivered. This may give more insight on how the policy operates and how can the learning Q-learning algorithm be improved.

8 References

Bellman, R. (1957). A Markovian Decision Process. *Indiana Univ. Math. J.*, *6*(4), 679–684.

Bellman, R. (1958). Dynamic programming and stochastic control processes. *Information and Control*, *1*(3), 228–239. https://doi.org/10.1016/S0019-9958(58)80003-0

Bertsekas, D. P. (2007). *Dynamic Programming and Optimal Control, Vol. II* (3rd ed.). Athena Scientific.

Chen, X., Wang, T., Thomas, B. W., & Ulmer, M. W. (2023). Same-day delivery with fair customer service. *European Journal of Operational Research*, *308*(2), 738–751. https://doi.org/https://doi.org/10.1016/j.ejor.2022.12.009

Cooper, D. R., & Schindler, P. S. (2014). *Business Research Methods* (12th ed.). McGraw-Hill.

Gutin, G., Punnen, A., Barvinok, A., Gimadi, E., & Serdyukov, A. (2001). *The Traveling Salesman Problem and Its Variations*.

Heerkens, H., Winden, A. v., & Tjooitink, J.-W. (2017). *Solving Managerial Problems Systematically* (1st ed.). Noordhoff Uitgevers.

Hüfner, D. (2020). Dr. Oetker kauft Flaschenpost – angeblich für eine Milliarde Euro. *Business Insider*.

Jang, B., Kim, M., Harerimana, G., & Kim, J. W. (2019). Q-Learning Algorithms: A Comprehensive Classification and Applications. *IEEE Access*, *7*, 133653–133667. https://doi.org/10.1109/ACCESS.2019.2941229

Kaelbling, L. P., Littman, M. L., & Moore, A. W. (1996). Reinforcement learning: A survey. *Journal of Artificial Intelligence Research*, *4*, 237 – 285. https://doi.org/10.1613/jair.301

Kang, Y. (2018). An Order Control Policy in Crowdsourced Parcel Pickup and Delivery Service. In G. M. and P. J. and K. D. and von C. G. Moon Ilkyeong and Lee (Ed.), *Advances in Production Management Systems. Smart Manufacturing for Industry 4.0* (pp. 164–171). Springer International Publishing.

Kavuk, E. M., Tosun, A., Cevik, M., Bozanta, A., Sonuç, S. B., Tutuncu, M., Kosucu, B., & Basar, A. (2022). Order dispatching for an ultra-fast delivery service via deep reinforcement learning. *Applied Intelligence*, *52*(4), 4274–4299. https://doi.org/10.1007/s10489-021-02610-0

Ławrynowicz, A., & Tresp, V. (2014). Introducing Machine Learning. In *Perspectives on Ontology Learning* (Vol. 18, pp. 35–50).

Lopes Silva, M. A., de Souza, S. R., Freitas Souza, M. J., & Bazzan, A. L. C. (2019). A reinforcement learning-based multi-agent framework applied for solving routing and

scheduling problems. *Expert Systems with Applications*, *131*, 148–171. https://doi.org/https://doi.org/10.1016/j.eswa.2019.04.056

Powell, W. (2011). *Approximate Dynamic Programming: Solving the Curses of Dimensionality*. https://doi.org/10.1002/9781118029176

Ross, S. M. (2009). *Introduction to Probability Models* (Tenth). Academic Pres.

Statista. (2023). *Topic: Online grocery shopping in Germany*. Https://Www.Statista.Com/Topics/9848/Online-Grocery-Shopping-in-Germany/#topicOverview.

Sutton, R. (1988). Learning to Predict by the Method of Temporal Differences. *Machine Learning*, *3*, 9–44. https://doi.org/10.1007/BF00115009

Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: an introduction* (Second).

van Otterlo Martijn and Wiering, M. (2012). Reinforcement Learning and Markov Decision Processes. In M. Wiering Marco and van Otterlo (Ed.), *Reinforcement Learning: State-of-the-Art* (pp. 3–42). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-27645-3_1

Watkins, C. (1989). *Learning From Delayed Rewards*.

Yahyaa, S. (2015). *Explorations in Reinforcement Learning: Online Action Selection and Value Function Approximation*.

Yan, Y., Chow, A. H. F., Ho, C. P., Kuo, Y.-H., Wu, Q., & Ying, C. (2022). Reinforcement learning for logistics and supply chain management: Methodologies, state of the art, and future opportunities. *Transportation Research Part E: Logistics and Transportation Review*, *162*, 102712. https://doi.org/https://doi.org/10.1016/j.tre.2022.102712

Yang, L., & Shami, A. (2020). On hyperparameter optimization of machine learning algorithms: Theory and practice. *Neurocomputing*, *415*, 295–316. https://doi.org/https://doi.org/10.1016/j.neucom.2020.07.061

9 Appendix

9.1 Computer and software specifications

Resource	Specifications
Processor:	AMD Ryzen 7 4800HS with Radeon Graphics 2.90 GHz
RAM Memory:	16.0 GB
Storage:	952Gb SSD
Graphics card:	NVIDIA GeForce GTX 1660 Ti
Operating system:	Windows 11 64-bit
Python:	Python version 3.9
IDE:	PyCharm Community Edition 2023.1.3

Figure 9-1 - Computer and software specifications

9.2 Q-Policy output results

9.2.1 Comparison of Q-learning agent learning for $N_D = \{10, 15, 20\}$

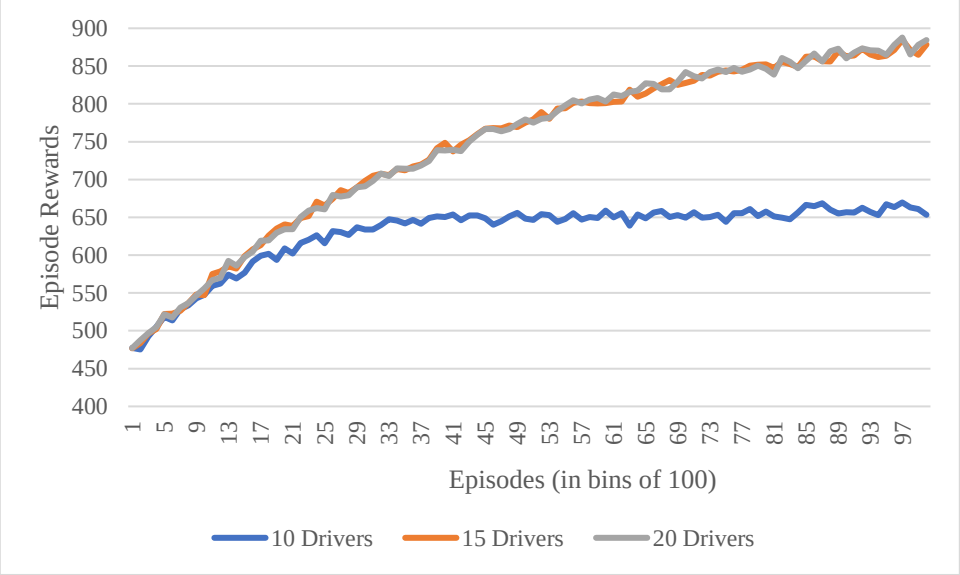

Figure 9-2 - Comparison of Q-learning agents for $N_D = \{10,15,20\}$

9.2.2 Heatmaps for order punctuality for $N_D = 15$

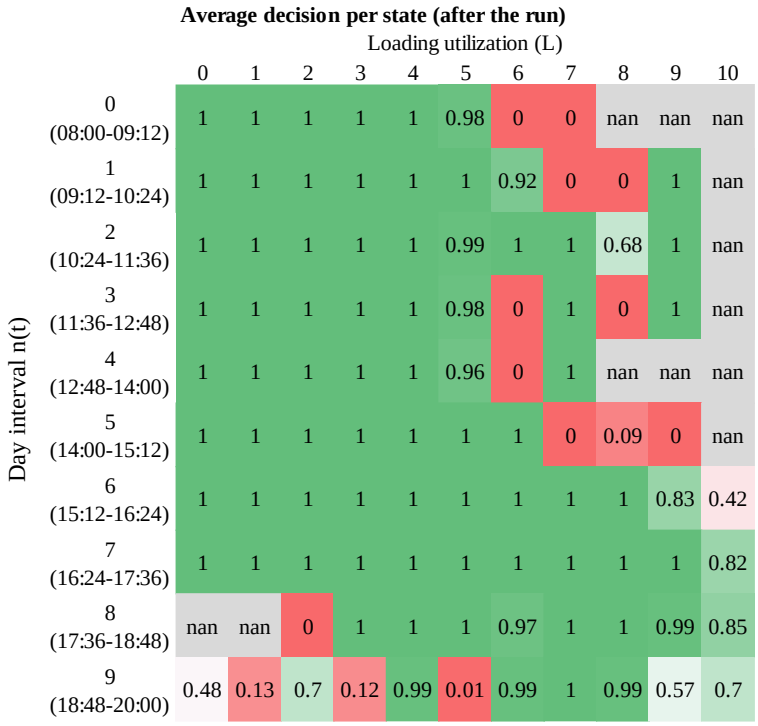

Figure 9-3 - Heatmap for average decision taken at each L state after a 100-day run for $N_D = 15$

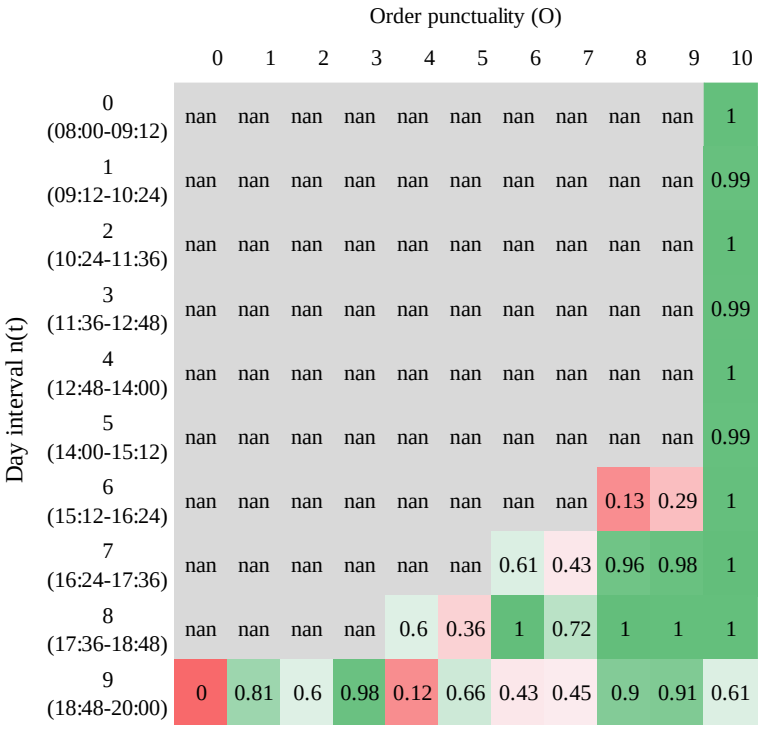

Figure 9-4 - Heatmap for average decision taken at each O state after a 100-day run for $N_D = 15$

In Figure 9-4 we show what was the average decision that the agent took after a 100-day run. Nan indicates that the agent never visited that state.

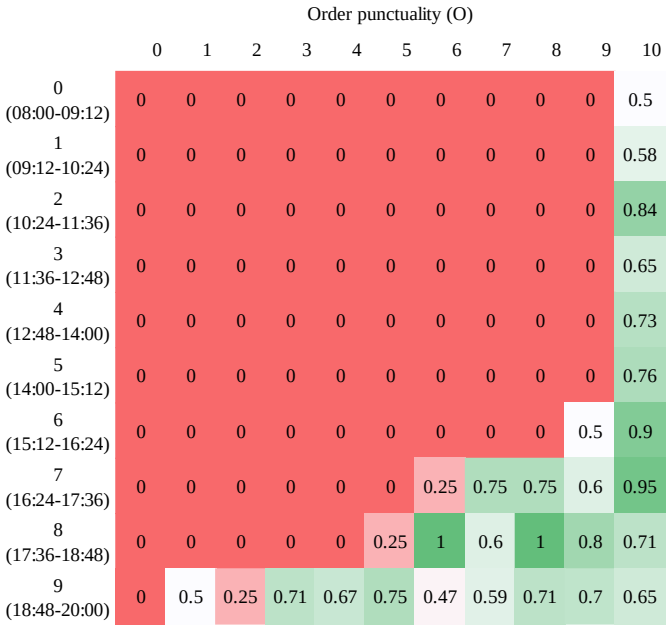

Figure 9-5 - Day interval decision heatmap averaged over the order punctuality for $N_D = 15$

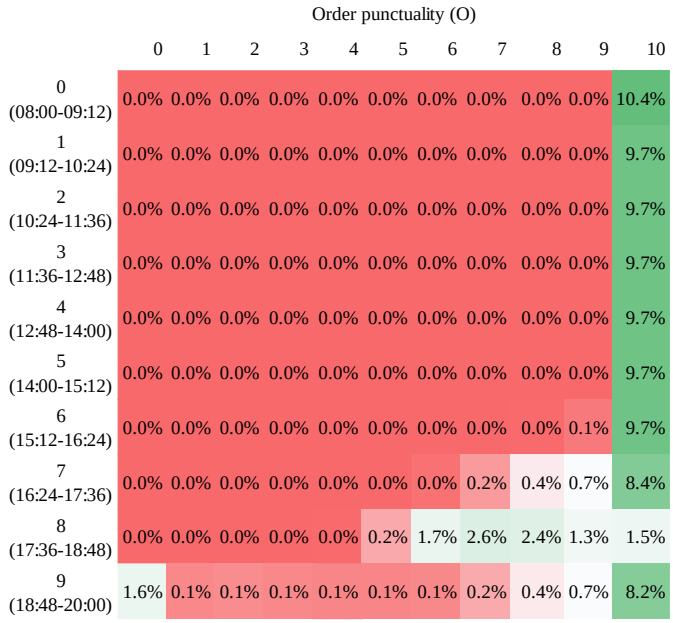

Figure 9-6 - Percentage of time the agent visited each order punctuality after a 100-day simulation for $N_D = 15$

9.2.3 Heatmaps for loading utilization for $N_D = 10$

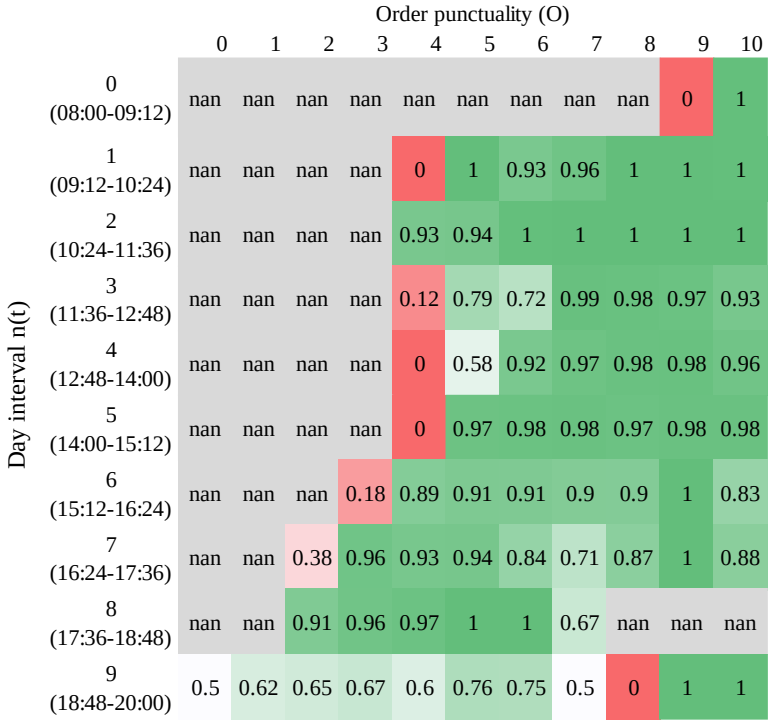

Figure 9-7 - Heatmap for average decision taken at each O state after a 100-day run for $N_D = 10$

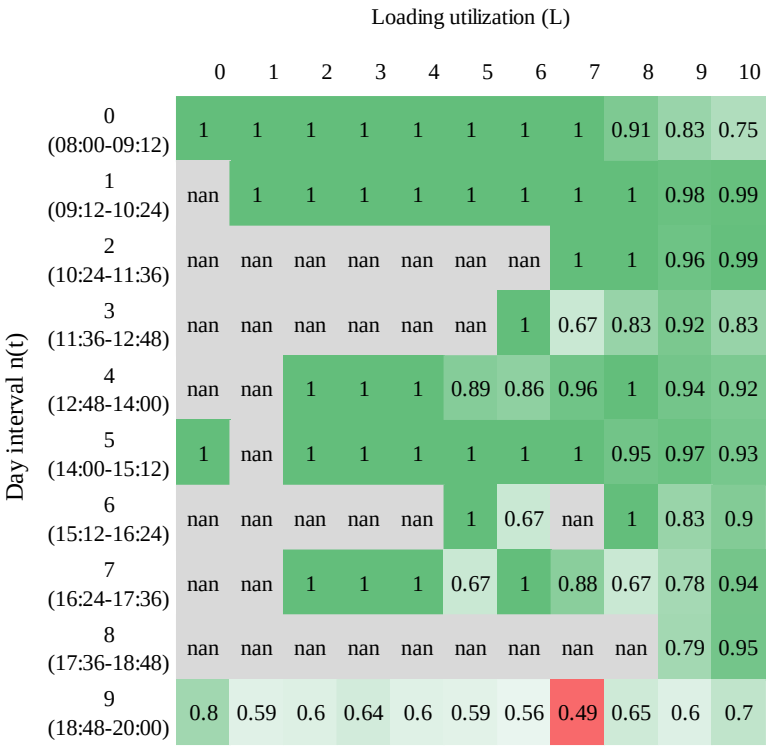

Figure 9-8 - Heatmap for average decision taken at each L state after a 100-day run

In Figure 9-4 we show what was the average decision that the agent took after a 100-day run. Nan indicates that the agent never visited that state.

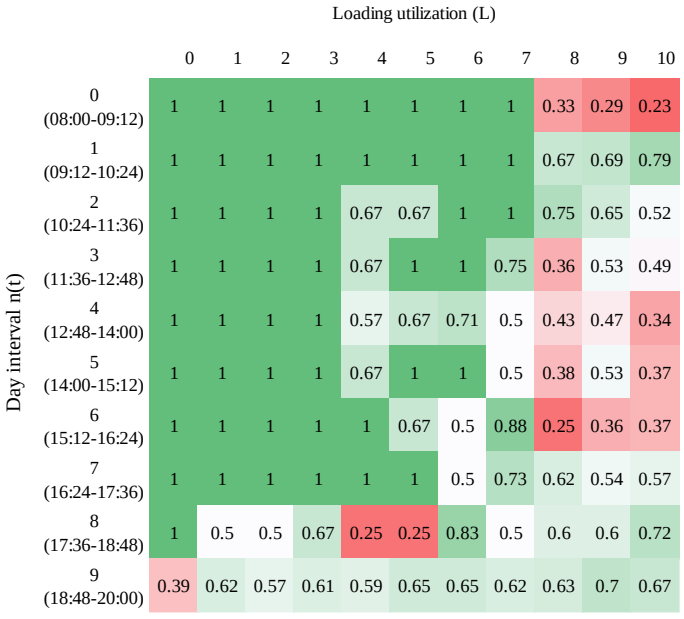

Figure 9-9 - Day interval decision heatmap averaged over the loading utilization for $N_D = 15$

Figure 9-10 - Percentage of time the agent visited each loading utilization after a 100-day simulation for $N_D = 15$

9.3 Policy comparison results

9.3.1 Average decision comparison across a complete day between Q-policy and company derived policy for $N_D = 10$

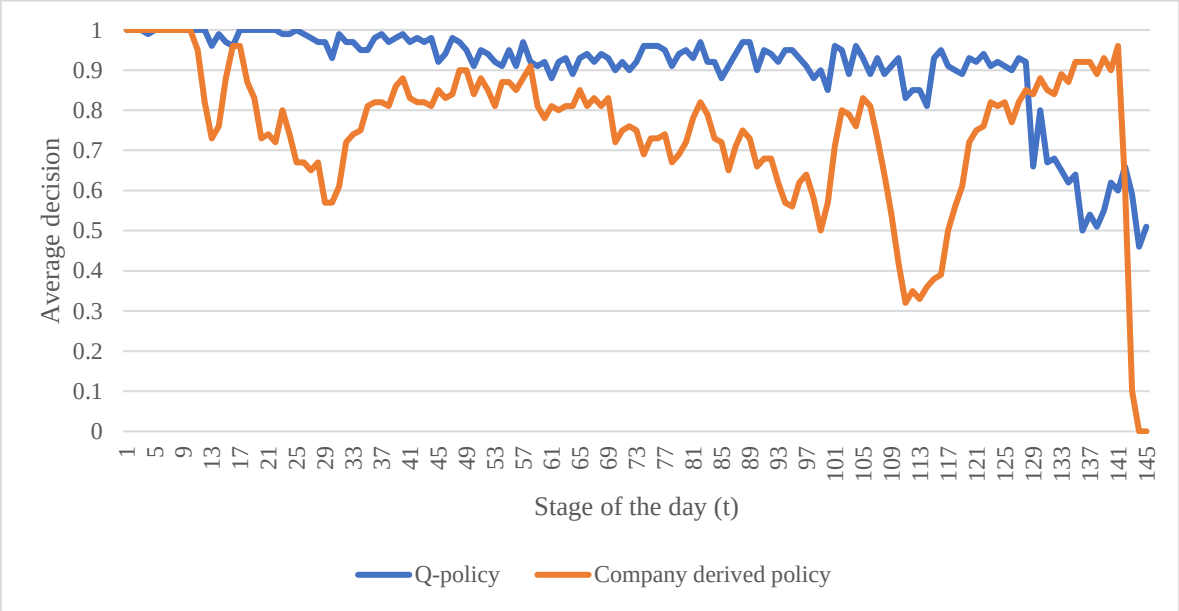

Figure 9-11 - Average action taken by the Q-policy and the Company derived policy per stage t after a 100-day simulation for $N_D = 10$

9.3.2 Average state value comparison across a complete day between Q-policy and company derived policy after a 100-day simulation for $N_D = 10$

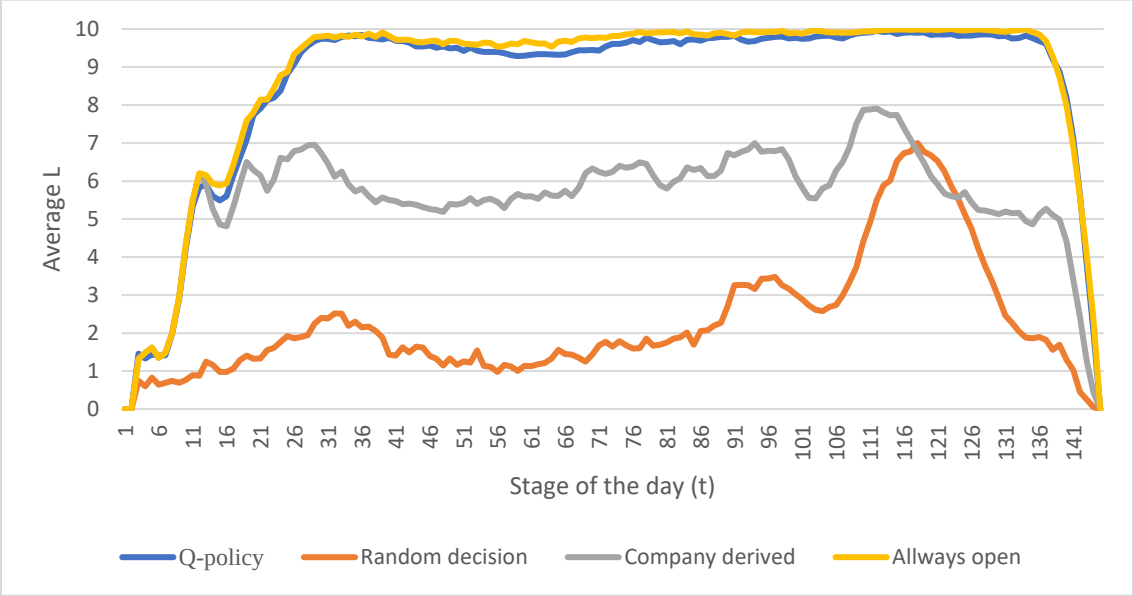

Figure 9-12 - Average loading utilization of the 4 policies per stage t after a 100-day simulation for $N_D = 10$

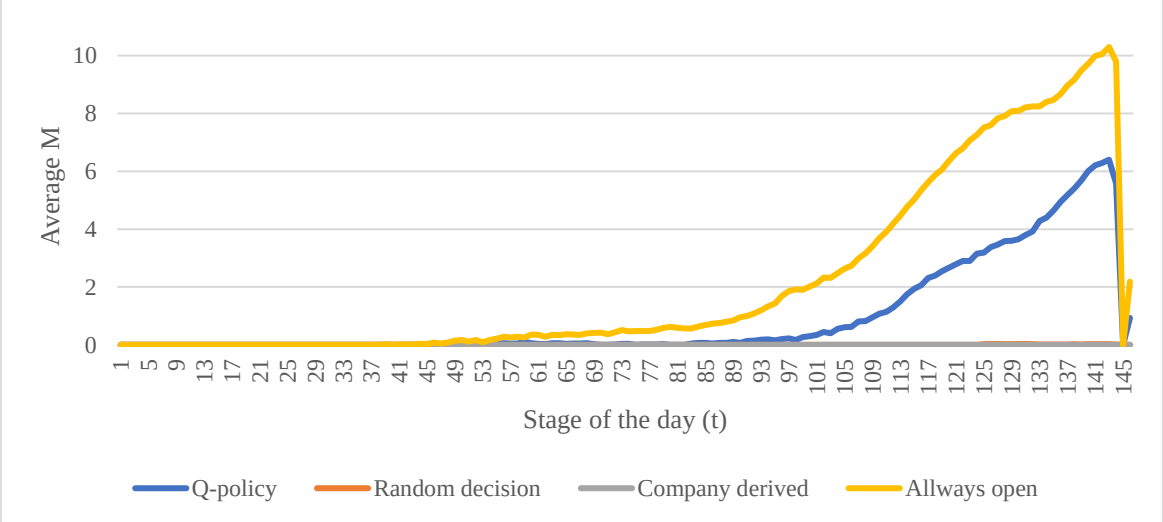

Figure 9-13 - Average lateness M of the 4 policies per stage t after a 100-day simulation for $N_D = 10$

Note that on Figure 9-13 only the Q-policy and the always open policy appear, as the other two policies do not show any lateness.