Optimizing truck to dock assignment with Deep Reinforcement Learning

Tuin, Harro R.

Award date:
2022

EINDHOVEN UNIVERSITY OF TECHNOLOGY

Department of Industrial Engineering and Innovation Sciences
Information Systems Group

Optimizing truck to dock assignment with Deep Reinforcement Learning

4th April 2022

in partial fulfilment of the requirements for the degree of
Master of Science in Operations Management and Logistics

Supervisors:
TU/e — dr. Yingqian Zhang
TU/e — dr. Zaharah Bukhsh

Author:
H.R. Tuin

Student ID:
0864204

Abstract

The number of stores of large retailer X has rapidly increased over the past years. Due to the increasing number of stores the capacity of the distribution centers (DC's) becomes relevant. This study aims to optimize the assignment of inbound trucks to docks at the distribution centers with Deep Reinforcement Learning (DRL) such that the operational costs are minimized. As the costs are partially based on the time needed to unload trucks and to transport carriers inside the DC, this can be used as a proxy for the capacity of the DC's. First, a discrete-event simulation of the current inbound operations is developed. This simulation is used to evaluate new policies as well as the current policy of assigning trucks to docks. It is expected that the current greedy policy falls short when a queue of inbound trucks starts to form. To improve the policy in this scenario, a new policy is created by training a DRL agent on an environment that simulates the inbound operations of the DC's. The DRL agent reduces the costs of the inbound operations by 26.9% compared to the greedy approach in a simulated environment. The trained agent is analyzed using SHAP values to get insight into the learned policy. The policy analysis reveals that the availability features have the largest influence on the decision-making process of the agent, followed by distance and waiting time, which is consistent with expectations. Subsequently, the size of the state space is reduced by more than 20% based on the SHAP values found in the policy analysis. The model trained on the the reduced state space achieved a similar performance as the agent trained on the original state space in less time steps.

Executive Summary

The problem discussed in this report concerns the optimization of truck to dock assignment at distribution centers of retailers. The project was carried out at consulting firm company Y. Large retailer X, one of company Y's clients, has noticed an increasing number of problems concerning the capacity at their distribution centers. Specific areas that could provide opportunities for improvement are the docks and staging lanes of the DC's of large retailer X. As costs can be used as a proxy for capacity, this report specifically focuses on the optimization of the truck to dock assignment of inbound trucks such that the waiting costs and travel costs inside the DC are minimized. The main research question of this research is:

How can large retailer X use deep reinforcement learning to optimize the assignment of inbound trucks to docks such that the waiting and transportation costs are minimized?

The current greedy policy of assigning trucks to docks does not work well when the capacity at the docks or staging lanes is pressing. When all dock-staging lane combinations are occupied, an inbound truck cannot immediately dock after arriving. Currently, the next truck in the queue is assigned to a dock when this dock becomes available. However, it might be that another truck in the queue is a better fit for that dock with respect to the distance between the dock and the products inside the DC. This forms the basis for an improved policy.

Methods

Recently, Deep Reinforcement Learning (DRL) has proven to be successful in optimizing processes where sequential decision-making under uncertainty plays a role. The inbound trucks arrive at the DC according to a stochastic process. After trucks arrives they have to be sequentially assigned to a dock. Therefore, it is expected that DRL could be a suitable way to optimize the truck to dock assignment of inbound trucks.

First, the environment needed to train the DRL agent is developed. To make the research of this study reproducible and easy to understand the environment follows the OpenAI gym interface. The environment relies heavily on a discrete-event simulation of the inbound operations. The parameters for this simulation are partly derived from data, when this is not possible the parameters are estimated in consultation with a subject matter expert. Initially, the state space consists of all information experts deem necessary to make an optimal truck to dock assignment. The state space contains information on the availability of docks, staging lanes, distances between docks and weighted product locations for trucks in the queue as well as expected arrivals. Altogether the state space consists of 322 features for a scenario with 10 docks. The action space is discrete with 11 actions — select one of the first 10 trucks in the queue to be docked, or wait. The queue is not always the same length which means not every action is valid at each time step. Therefore, invalid actions are masked such that the agent can only choose valid actions. After a truck has been selected it is assigned greedily to the dock that minimizes the distance between the dock and the weighted product location inside the DC. The agent gets an episodic reward

that is equal to the negative mean costs per truck for that episode, where each episode represents 1 day (16 hours) of real time inbound operations. Proximal Policy Optimization (PPO) is used to train the DRL agent.

To get insight into the policy learned by the agent a policy analysis is performed. The first purpose of the policy analysis is to learn what information in the state is important for the agent to decide on taking a certain action. The second purpose is to extract heuristic rules from the policy of the agent, these rules can give further insight into the policy and allow for easier adaptation of the solution in practice. Shapley Additive Explanations (SHAP) are used to determine feature importances, whereas a decision tree is used to derive the heuristic rules. Based on the SHAP values, features can be excluded from the state space if they are not important in the decision-making process. This approach is novel and thus contributes to the field of AutoRL by linking evaluations of SHAP values to state space reduction.

Experimental setup

The current greedy approach and the DRL solution are compared in two main scenarios. These scenarios are the *large* and *small* case. Of these two cases the large case is closest to the real operations of the DC, the small case is a simplified version. In the large case the distribution center is assumed to have 10 docks and 16 hours of operations per day, whereas the small case has 4 docks and 4 hours of simulation time. Next to the small and the large case, other scenarios based on the large case are introduced. These scenarios have slightly different parameter settings, for instance the arrival rate will be slightly lower or higher compared to the training setup. The goal is to use these scenarios to test how well different approaches can handle changes in the environment.

Results

Compared to the greedy method the DRL agent reduces the costs with 26.9% in the large case on average, for the small case the mean costs are reduced by 24.5% on average. Additionally, results show that the mean performance of the agent is better than the greedy approach in every aspect (i.e. transportation costs, waiting costs, mean waiting time at the docks, mean waiting time at the staging lanes) except the mean queue length at the docks. This makes sense because the agent has the option to let trucks wait whereas the greedy method never lets a truck wait if there is a possibility to dock. The sensitivity analysis shows the agent outperforms the greedy approach when the arrival rate is increased. However, if the arrival rate decreases the greedy approach performs marginally better.

The heuristic rules are extracted from a decision tree classifier fitted on collected trajectories generated by the trained agent that achieved the lowest mean costs per truck during evaluation. The tree shows the first split is made on the estimated departure time for the truck at dock 2. This is interesting since it is not expected to be such an important feature. However, when there is no truck at this dock, this value is equal to the current time. Hence, it might be that the first split is actually based on the time feature. It could indeed be that the current time in the simulation indicates what action the agent takes. The simulation always starts with empty docks, therefore the decisions could change depending on the time in the simulation. Furthermore, one specific dock (dock 2) seems to have large influence on the decision to dock the first truck. For docking the second truck it seems that the distances for dock 3 are most important. For the wait action no clear pattern is found.

The policy analysis shows the availability features of the docks and lanes are the most important in the agent's decision. Furthermore, the distance feature has a large influence on the decisions. It seems the agent is able to learn a connection for the distance between the docks

and the product locations and the reward. Additionally, the waiting time is an important feature in the decision to dock the second truck in the queue. Even though there is only an episodic reward, which might make learning the effect of waiting time on the reward difficult, the agent still appears to be able to learn the relationship between waiting time and the episodic reward.

In general, it seems the agent has a focus on dock 1 and 2. This focus can be explained by the central location of these docks, which makes these the most interesting places to dock for most trucks. Some other features have little influence on the agent's decision, such as the time or the features related to the third truck in the queue. Furthermore, it seems the agent is not influenced by the current time in the simulation.

Based on the SHAP values the state space for the small case is reduced from 45 to 35 features by excluding features that do not influence the model output. After retraining the agent on the smaller state, the learning curve reveals the agent is able to achieve similar performance during training in less time steps. A further reduction of the state space showed a significant drop in performance. For the large case, the state space reduction seems to be more complex as the agent trained on a reduced state space is unable to achieve the same performance as the agent trained on the original state space.

Conclusion

It can be concluded that the master's thesis project filled five distinct gaps in literature. First, the problem context and the stochasticity in the system make the assignment problem in this project is unique. Second, although Deep Reinforcement Learning (DRL) has been applied to similar problems, the application of DRL to this setting, but also to the closely related stochastic sequential assingment problems (SSAPs), is novel. Third, the application of interpretable and explainable DRL methods on practical problems adds to the existing body of literature. Fourth, the thesis project provides an environment in accordance with the OpenAI gym interface that could be used by any researcher or practitioner to explore the application of DRL to assignment problems. Finally, the project uses SHAP values to reduce the state space by a significant amount without compromising on performance of the model. In addition, the study also has practical relevance. The study revealed weaknesses of the current greedy policy used by large retailer X. Additionally, a DRL solution that surpasses the performance of the greedy approach in nearly all KPI's was developed. With the results of this study the main research question can be answered.

How can large retailer X use deep reinforcement learning to optimize the assignment of inbound trucks to docks such that the waiting and transportation costs are minimized?

To use DRL solution an MDP has to be formulated. A discrete-event simulation forms the basis for the environment with which the agent interacts. The agent can choose which truck to dock next, subsequently it is assigned greedily to the dock that minimizes the distance to the weighted product location inside the warehouse.

Compared to the greedy method the DRL agent on average reduces the costs with 26.9% in a simulated scenario. Additionally, results showed that the mean performance of the agent is better than the greedy approach in every aspect except the mean queue length at the docks. The sensitivity analysis showed the agent outperforms the greedy approach when the arrival rate is increased. However, if the arrival rate decreases the greedy approach performs marginally better.

Preface

This master's thesis report describes the results of the research project I conducted in partial fulfilment of the requirements for the degree of Master of Science in Operations Management and Logistics at the Eindhoven University of Technology.

I thank company Y for offering me the opportunity to put the knowledge I obtained during my studies into practice. I also want to thank my company supervisors for their critical feedback on my work and showing me what it is like to work on a project that can have real impact. In addition, I would like to thank my academic supervisor, dr. Yingqian Zhang, for her guidance and advise during the master's program. I am grateful that you helped and guided me in pursuing my interests in machine learning. When I was sometimes doubting my approaches during my thesis project, you gave me the confidence I needed to keep going.

I also thank my friends for moral support and the wonderful time I had as a student. Finally, I would like to express my gratitude to my parents for their unwavering support during my time as a student, I am forever indebted to you.

Harro Tuin

Eindhoven, March 2022

Table of contents

List of Figures

List of Tables

Chapter 1

Introduction

This report details the project carried out in partial fulfillment of the requirements for the degree of Master of Science in Operations Management and Logistics. The problem discussed in this report concerns the optimization of truck to dock assignment at distribution centers of retailers. The project was carried out at consulting firm company Y. Large retailer X, one of company Y's clients, has noticed an increasing number of problems concerning the capacity at their distribution centers. The problems concerning the capacity at distribution centers are not unique to large retailer X. A recent news article discussed that large e-commerce companies such as Amazon are facing similar capacity issues at their fulfillment centers (Garland, 2021). According to the article all indications point to another peak season capacity bottleneck, triggered by the acceleration of e-commerce development following the COVID-19 epidemic. Therefore, it seems the problem is more general and widespread than just the DC's of large retailer X. For years, large retailer X and company Y have collaborated on the development of smart algorithms to make the processes in the distribution centers (DCs) of large retailer X more efficient. The specific areas that could provide opportunities for improvement are the docks and staging lanes of the DC's of large retailer X. Staging lanes are connected to the docks and are used to temporarily store products after delivery to the DC. A staging lane can be used for:

1. Inbound carriers delivered by a supplier
2. Inbound carriers delivered as transshipments from a different DC of large retailer X
3. Outbound roll containers that need to be delivered to an large retailer X supermarket

After a carrier is delivered by a supplier, the carrier is stored inside the DC. In contrast, the carriers delivered as transshipments are consolidated and loaded onto outbound trucks without, or with very limited, intermediate storage. The outbound carriers with products that need to be delivered to an large retailer X supermarket are compiled by order pickers.

Due to a limited amount of staging lane capacity and an increase in the number of stores that need to be supplied, the staging lane capacity is becoming increasingly important. A problem arises when there are not enough staging lanes available. On the one hand, this could lead to suppliers having to wait before they can unload their carriers. On the other hand, this could lead to delay in shipments to the supermarkets.

Currently, a specific number of staging lanes are reserved for either only in- or outbound trucks. As a result of this static planning, it can happen that staging lanes are unused for long periods of time. Company Y expects that with the help of mathematical methods the use of

staging lanes can be optimized. First, it is expected that with a smart truck-dock assignment the distance traveled inside the DC can be reduced. Additionally, optimizing the truck to dock assignment could lead to a decrease in waiting time and thus an increase in overall capacity at the DC. Second, it is expected that a more dynamic assignment of which staging lanes can be used for in- and outbound trucks can increase the staging lane capacity. This report specifically focuses on the optimization of the truck to dock assignment of inbound trucks such that the waiting costs and travel costs inside the DC are minimized.

This report is structured as follows: first, background information, a description of the problem, and the research questions are discussed in the remainder of this chapter.Subsequently, Chapter 2 discusses literature relevant to the problem that large retailer X is facing. Additionally, this chapter indicates how this thesis project aims to add to the existing literature. Chapter 3 outlines the methods used to optimize the truck to dock assignment problem, followed by Chapter 4 which discusses the experimental setup. After that, the results of the experiments are discussed in Chapter 5. Finally, Chapter 6 concludes the report and provides recommendations for future research.

1.1 Current process

Inbound trucks

Inbound trucks transport products from suppliers to the DC's of large retailer X. For most inbound trucks ($\approx$ 90%), large retailer X receives an advanced shipping notice (ASN) that states a time window in which the truck is estimated to arrive. The ASN contains information on the type and number of products that will be delivered. Nonetheless, approximately 10% of the inbound trucks do not have an ASN, for these trucks only the arrival date is known.

Ideally, the inbound trucks do not have to wait before they can dock but in practice waiting is common. If more than one truck has to wait, a queue starts to form. The assignment of trucks in the queue is handled on a first-come-first-serve (FCFS) basis. However, some trucks are prioritized. For example, if a certain product is completely out of stock in the DC and in high demand, the inbound trucks in the queue that carry this product could have priority. Other types of priority can occur when a truck in the queue also serves as an outbound truck at a later time. Although queuing trucks can have distinct priorities, there is only one queue and the prioritization is done ad-hoc.

When trucks are assigned to a dock the products can be unloaded. Upon delivery, the carriers are temporarily stored on the staging lanes before being transferred to the bulk storage. The time required to transfer the carriers to the bulk storage is subject to uncertainty as the availability of operators inside the DC is limited. The assignment of trucks to docks influences the time needed to clear the staging lanes as this determines the travel distance inside the DC. Fig 1.1.1 shows a simplified map of a distribution center. The figure shows the connection between docks and staging lanes. Please note that the schematic shows exactly one staging lane per dock, in reality, there might be two staging lanes connected to one dock. Furthermore, the actual DC's of large retailer X have over 50 docking doors, therefore, the impact of truck assignment on the travel distance inside the DC increases. Currently, the inbound trucks from suppliers are assigned greedily with respect to the distance between the staging lane and the location of the products inside the DC. The inbound trucks with transshipments are assigned to a special cross-docking area. Finally, it is important to note that the inbound staging lanes and docks that can be used for in- and outbound trucks are currently fixed. For one specific DC this is fixed throughout the year, the other DC's have a more dynamic schedule. For example, each day at 17:00 *all* docks and staging lanes are temporarily (until the next day) opened for inbound trucks.

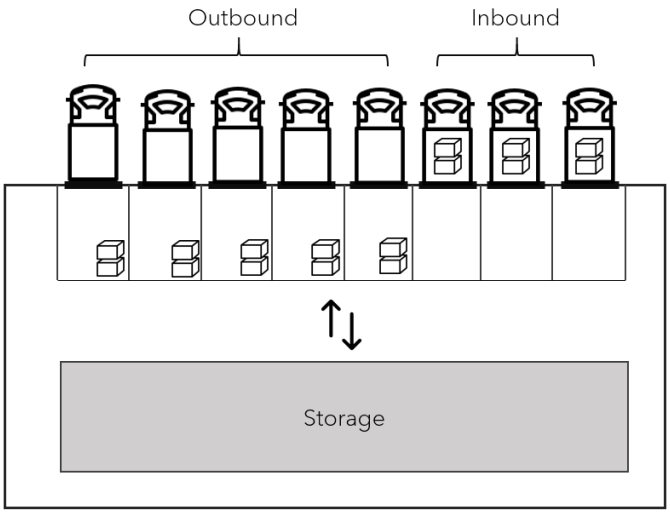

Figure 1.1.1: Simplified schematic of DC operations

1.2 Challenges

The main challenges related to the assignment of trucks to staging lanes are the following. For the assignment of inbound trucks, the main challenge is uncertainty. First, the arrival time of trucks is uncertain, both for trucks that do and do not specify a time window of arrival. Second, due to the limited capacity of operators in the DC, the total time needed to clear a staging lane is also variable. Third, the current policy of assigning trucks to docks does not work well when the capacity at the docks or staging lanes is pressing. When all dock-staging lane combinations are occupied, an inbound truck cannot immediately dock after arriving. Currently, the next truck in the queue is assigned to a dock when this dock becomes available. However, it might be that another truck in the queue is a better fit for that dock with respect to the distance between the dock and the products inside the DC.

Another challenge for assigning inbound trucks is the waiting costs of truck drivers. The waiting costs are dependent on a number of factors, for example, whether or not (1) a truck arrives in the time window as specified in the ASN, (2) a truck is needed for outbound operations as well, and (3) the content of a truck is out-of-stock in the DC *and* needed to complete orders for outbound trucks. These factors complicate the cost function needed to come to a representative and accurate solution. To complicate matters further, there are indirect costs to queuing trucks. As one can imagine, if trucks from suppliers have to wait too long or too often, this could have a negative impact on large retailer X's position in negotiating shipping costs.

For outbound operations, the main challenge lies in the interaction with the order picking process. As the algorithms that decide on the composition of orders use the staging lane planning as input this is hard to optimize without adapting the existing algorithms as well. The algorithms for the order picking process try to minimize the number of stops and travel distance during the picking process. This travel distance includes the distance from the picking circuit to the drop-off location at the staging lane. Therefore, if the objective of the assignment of trucks to staging lanes is to also optimize the travel distance, a cycle of changes occurs. This means that if the staging lane planning changes, the composition of orders will most likely change as well and vice-versa. It is yet unclear how this problem can be tackled without adapting the existing algorithms used to optimize the order picking process.

1.3 Scope

With the background on the operations of the DC's, it is possible to define the scope of this project. This project addresses the problem of the inbound operations at the distribution centers of large retailer X. The goal is to minimize the costs of the inbound operations and increase the dock capacity by optimizing the truck to dock assignment of inbound trucks. This project solely focuses on the assignment decision and does not take into account other aspects of the DC's operations. The following is not included in this project:

- Long-combination vehicles, as these have to dock at special docks
- Trucks that do not send an ASN.
- Subcontracted DC's, as these manage their own operations.
- The order picking operations, the algorithms used in the order picking process are restricted from any modifications. The order picking operations are also not part of the simulation for this project.
- Cross-docking operations are not taken into account as this is not the main interest of large retailer X and would make the problem increasingly complex.
- The slotting of products in the DC is not included in the project as this was previously optimized by company Y.
- All outbound operations.
- Dynamic docks are not considered as it is unknown how often and exactly when this process occurs which makes it hard to model.

1.4 Research Questions

The main research question of this research is:

How can large retailer X use deep reinforcement learning to optimize the assignment of inbound trucks to docks such that the waiting and transportation costs are minimized?

To answer the main research question, several sub-questions are developed that help structure the research. These sub-questions are the following:

- What is the current process of assigning trucks to docks and how can this be modeled?
- What is Deep Reinforcement Learning and how can it be used to solve the truck to dock assignment problem?
- What are methods to interpret and explain the behavior of a Deep Reinforcement Learning agent?
- What information about the inbound operations is required for a Deep Reinforcement Learning agent to perform well?
- Is it possible to reduce the state space based on SHAP values without diminishing performance?

Chapter 2

Literature Review

This chapter discusses relevant literature regarding the truck to dock assignment problem large retailer X is facing. First, literature on assignment problems is discussed. Thereafter, literature on the modeling of operations is explored. Subsequently, literature on reinforcement learning and deep reinforcement learning are discussed. Finally, literature on interpretation methods for deep reinforcement learning is reviewed.

2.1 Assignment Problems

Assignment problems are typical examples of combinatorial optimization problems (Burkard, 1979). The interest in these problems stems from their simple structure and relevance to problems faced in practice. Hillier (2012) describes the assignment problem as a linear programming (LP) problem where jobs have to be assigned to agents. An example of such a problem is that of assigning jobs to people, however, the agents do not have to be people. For example, in this project, the inbound trucks could represent the jobs and the docks could represent the agents. For a problem to be defined as an assignment problem, it has to be formulated in such a way that the following assumptions hold (Hillier, 2012):

- The number of agents and jobs are equal
- Each job is performed by *one* agent
- Each agent is assigned to *one* job
- There is a cost related to assigning an agent to a job
- The objective function is to minimize the cost of assigning all jobs to the agents

Many different applications, extensions, and variants of the assignment problem have been proposed since its inception. This section of the literature review focuses on the problems most relevant to the problem described in Chapter 1. As the Classical Assignment Problem forms the basis for most variants of assignment problems this is discussed first. Subsequently, a problem more similar to the truck to dock assignment problem is discussed; the Stochastic Sequential Assignment Problem.

2.1.1 Classical Assignment Problem

The Classical Assignment Problem (CAP) is a formulation of the assignment problem in its simplest form. The CAP is used to find an optimal one-to-one pairing among n agents and n jobs. The objective of the CAP is to minimize the total costs of assignments. As such, the CAP can be mathematically formulated as follows (Pentico, 2007):

$$
\begin{aligned}
\text{Minimize} \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \\
\text{Subject to:} \quad & \sum_{i=1}^{n} x_{ij} = 1 \qquad i = 1,\ldots,n, \\
& \sum_{j=1}^{n} x_{ij} = 1 \qquad j = 1,\ldots,n, \\
& x_{ij} = 0 \text{ or } 1
\end{aligned}
$$

In the formulation above, x_{ij} is a binary representation of the assignment of agent i to job j, with the corresponding costs c_{ij}. Many small modifications of this formulation can be handled, for example, a problem with a maximization objective. This maximization objective can often be converted into a minimization problem by multiplying the costs by -1, in this case, the "regret" needs to be minimized. Another example is an imbalance in agents and jobs, in this case, dummy agents or dummy jobs can be introduced.

Solution Methods

Ever since the introduction of the Hungarian Method by Kuhn (1955) an assignment problem that satisfies the previously mentioned assumptions, such as the CAP, can be solved in polynomial time ($\mathcal{O}(n^4)$). The Hungarian Method is a dual method based on the graph theorem developed by König and Egerváry in 1933. Dinic and Kronrod (1969) presented the first method to solve the assignment problem in only $\mathcal{O}(n^3)$.

The methods by Kuhn (1955) and Dinic and Kronrod (1969) can be used to find an optimal solution to the CAP in polynomial time. However, when the state space becomes large it becomes infeasible to find an optimal solution. To overcome this problem, Kurtzberg (1962) used heuristics to efficiently approximate the optimal solution. Additionally, more recent research showed how Neural Networks (Eberhardt et al., 1991) and Genetic Algorithms (GA) (Avis and Devroye, 1985) can be used to efficiently approximate the optimal solution of assignment problems as well. Finally, it is important to note that, the CAP is mathematically equivalent to the *weighted bipartite matching problem* from graph theory. The results from the weighted bipartite matching problem can therefore be used in solutions for the CAP.

CAP Applications

Even though the CAP has a simple formulation, it has many real-world applications. Dessouky and Kijowski (1997) showed how the CAP can be applied to production scheduling problems. The production scheduling problem concerns a "single-stage multi-product batch chemical process with fixed batch sizes" (Dessouky and Kijowski, 1997, p. 399). The authors propose a mixed-integer nonlinear programming approach to find the optimal batch size while minimizing the overall costs of scheduling the batches. When the batch sizes are known, the problem can be reduced to a CAP. Therefore, an optimal solution can be found in polynomial time.

Soumis et al. (1980) studied an aircraft scheduling problem with the objective to maximize profit while considering the satisfaction of customers. The authors show how the problem can be split into two systems that interact: (1) determining the flight schedule and (2) the passenger assignment problem to these flights. The model for the passenger assignment problem needs to predict how passengers will travel from their points of origin to their destinations given a flight schedule. The proposed solution uses an adaptation of the Frank-Wolfe algorithm for integer programming. For an elaborate explanation of the Frank-Wolfe algorithm please refer to p. 197 of Soumis et al. (1980).

Wang (2006) demonstrated how a single-machine scheduling problem with common due dates can be reduced to a classical assignment problem. The objective is to minimize the processing time by finding an optimal sequence of jobs to be processed. First, the author shows how the processing times can be calculated for any given sequence. Subsequently, the problem can be reduced to finding the optimal sequence, which can be formulated as a CAP. While the CAP can be solved by algorithms such as the Hungarian Method, the author presents a novel approach that can reduce the complexity to find an optimal solution of some instances to $\mathcal{O}(n \log n)$.

Research by Wang et al. (2019) studied the problem of scheduling off-shore oil-and-gas fields. In this research, the objective is to solve the allocation-scheduling problem of oil-and-gas field development that maximizes the net-present-value (NPV). Part of the problem, the location-allocation problem, is reduced to a CAP. The solution to this problem tries to determine the number, capacity, and location of the drilling platforms. The paper presents a mixed-integer linear programming (MILP) approach to solve the problem.

2.1.2 Stochastic Sequential Assignment Problem

The Stochastic Sequential Assignment Problem (SSAP) was first introduced by Derman (1972). In his paper, the problem is described as jobs that have to be assigned to agents to maximize the total expected reward. The jobs arrive sequentially with the value of each job represented by a random variable X. Additionally, a success rate p is assigned to every agent. If an agent with success rate p gets an "$X = x$" job assigned, the expected reward can be represented by px. After an agent gets assigned a job it becomes unavailable for future assignments. The main result of the paper is that the optimal assignment of agents to jobs is independent of the p_i's. The paper demonstrates that if there are n agents left to choose from, there are $n - 1$ numbers splitting the real line. If x falls into the ith interval then the optimal solution is to assign an agent with a success rate p_i to this job. A dynamic programming solution is presented to find the optimal policy.

Albright (1974b) considers a variant to the SSAP introduced by Derman (1972). In his paper, the jobs have a random arrival time and a reward that is only revealed upon arrival. The jobs have to be assigned to agents with different qualities (or success rates) that are known, however, it is also possible to reject a job. The objective is to assign the jobs to agents such that the total expected reward is maximized. The author uses three different ways to model the arrivals; according to a Poisson process, a renewal process, and a "timeless" model. An optimal solution is found by solving a system of Ordinary Differential Equations (ODE). However, solving these ODEs is inefficient and often impractical (Dervovic et al., 2021). The main contribution of Albright (1974b) is showing how time can be excluded as an explicit parameter in solving this problem. Furthermore, it shows that the optimal policies do not depend on the qualities of the agents. Another paper by Albright (1974a) showed how the same problem can be formulated as a Markov Decision chain. The approach of Albright (1974b) considers the case with independent random variables. Kennedy (1986) generalizes this case and shows the optimal policy for dependent random variables.

SSAP Applications

A common application of the SSAP is that of allocating kidneys to candidates on the transplant waiting list. In this application, n candidates are assigned to m kidneys that arrive sequentially. The reward for assigning a kidney to a candidate depends on the type of patient and type of kidney, where the type of kidney is only revealed upon arrival. Su and Zenios (2005) studied this application with the objective to maximize the total expected reward while candidates can only accept a kidney if it maximizes their own expected reward as well. The paper shows that an asymptotically optimal policy can be found if all candidates never decline an offer for a kidney. The authors recommend policy makers to take patient choice into account in the development of a kidney assignment system. Other studies of similar problems can be found in the papers by David and Yechiali (1985) and Zenios et al. (2000).

The SSAP could also be applied to an investment decision problem (Derman et al., 1975). In this problem, a limited pool of capital is available for investment. An investment opportunity arises with probability p during each period, which is similar to the uncertainty in arrivals of the SSAP. The problem is to decide how much to invest with the objective to maximize the expected profit. The profit of an investment is not known in advance, similar to the SSAP. The paper presents ways to find the optimal policy given a concave reward function as well as closed-form solutions for special cases.

A study by McLay et al. (2009) introduces the Sequential Stochastic Passenger Screening Problem (SSPSP) for aviation security which can be seen as a variant of the SSAP and the Dynamic and Stochastic Knapsack Problem (DSKP). In this problem, passengers have to be assigned to airport security personnel. The paper considers two classes with different screening procedures for each class. The passengers (jobs) arrive sequentially with an unknown level of risk. The level of risk of each passenger becomes known upon check-in, however, a pre-screening already gives a *perceived* risk level. The objective is to find the optimal policy that maximizes the expected number of true positives, based on the perceived risk levels. The authors formulate the SSPSP as a Markov Decision Process (MDP) and find an optimal policy using dynamic programming.

Recently, research by Dervovic et al. (2021) showed how the optimal sequential assignment algorithm of Albright (1974b) can be applied to detect financial fraud. The authors consider financial transactions as jobs that arrive sequentially according to a non-homogeneous Poisson process. The value of a job is defined by a function that considers the amount of the transaction and the confidence score found by a classifier. The agents are represented by transaction inspectors that have a certain budget of inspections. The optimal policy is found by determining critical curves that indicate when a job should be accepted or not. These critical curves are determined by a novel approach that efficiently finds an optimal solution.

Solution Methods

Existing solution methods to the SSAP include dynamic programming (DP) and solving a system of ODEs. McLay et al. (2009) presented a solution to a variant of the SSAP using dynamic programming. DP is a technique that can solve a problem recursively by breaking a complex problem down into parts. An advantage of using DP is that it can find exact solutions. However, DP needs significant computational resources in order to find this exact solution. Nonetheless, it is much faster than a brute force search.

Albright (1974b) proposes a solution that involves solving a system of ODEs to find the optimal policy. Similarly, Dervovic et al. (2021) propose an improved approach based on ODEs. However, as Dervovic et al. (2021) points out, solving these systems of ODEs is expensive and impractical, even for trivial processes. Dervovic et al. (2021) presents an efficient method that only needs M observations to arrive at an optimal solution.

A solution that could be promising is that of deep reinforcement learning. This tackles the intractability of a DP approach for large state spaces. Furthermore, it relies on approximations of optimal results instead of finding an exact optimal result. Therefore, finding a close to optimal policy becomes significantly less expensive while maintaining practical relevance.

Conclusion

This section introduced assignment problems, with special attention to the classical assignment problem (CAP) and the stochastic sequential assignment problem (SSAP). Furthermore, applications and solution methods of both CAP and SSAP were discussed. The problem of large retailer X in the Master's Thesis project can be described as a variant of the SSAP. In the original formulation of the SSAP the objective is to maximize the expected reward, in the large retailer X problem the objective is to minimize the costs of the assignment. However, assignment problems can be easily modified to include such differences. Furthermore, in the formulation by Derman (1972) the rewards of the jobs are only known upon arrival, in the large retailer X problem, the cost of assignment can be partly determined before arrival as the distances between the docks and product locations are known. Another difference is the differentiation in skill or success rate between agents, in the problem of large retailer X all agents are identical with guaranteed success. Albright (1974b) already examined the model where all agents are identical and jobs arrive according to a non-homogeneous Poisson process. It can be concluded that while the problem of large retailer X is similar to the SSAP it has unique characteristics. Some existing methods that are used to solve the SSAP could be used to optimize the truck to dock assignment, such as dynamic programming (DP). However, DP becomes intractable for large state spaces.

2.2 Modeling Operations

The main focus is on modeling the current inbound operations. As discussed in Section 1.2 the main challenges for the inbound operations lie in the uncertainty of arrivals. First, the uncertainty is discussed. Second, simulation methods, which can be used to model the inbound operations, are discussed.

Uncertainty

In the problem large retailer X is facing, there is uncertainty about the arrivals. Although trucks have an expected arrival time they could be delayed, for example, due to traffic. The problem at hand is most similar to the SSAP, therefore the SSAP is taken as the starting point for modeling the inbound truck arrivals. Albright (1974b) considers three different ways to model the arrivals; according to a non-homogeneous Poisson process, a renewal process, and a "timeless" model. More recently, Dervovic et al. (2021) considered arrivals according to a non-homogeneous Poisson process as well.

Poisson process

A Poisson process is frequently used in queuing theory to model random events, for example, the arrival of trucks (Kleinrock, 1975). The Poisson process is a stochastic process that counts the number of events in an interval $(0, t)$, where the time between two events is independent and exponentially distributed with parameter λ. The Poisson process is frequently used due to these convenient mathematical properties (Kingman, 1992). Specifically, the *Markov property* of the Poisson process is of importance. As a result of the Markov property, events in the process are independent, which means that the occurrence of an event gives no information on the likelihood of another event occurring. There are two main distinctions in the rate for Poisson processes. The

Poisson process is called *homogeneous* if the rate λ is a constant. The expected number of arrivals $E(N(t))$ during time $(0, t)$ is then equal to λt. In a *non-homogeneous* Poisson process the arrival rate is allowed to vary with time; $\lambda(t)$. This generalization leads to the loss of the property of stationary increments.

Renewal process

The Poisson process can be seen as a unique renewal process with the Markov property (Grimmett and Stirzaker, 2020). Whereas the Poisson process has exponentially distributed events, a renewal process can have any independent and identically distributed (IID) distribution of events. Therefore, the renewal process is a generalization of the Poisson process that allows arbitrary holding times. As Albright (1974b) points out, the optimal policy in the SSAP is not dependent on the distribution of the arrivals, and consequently, this may be altered. However, modifying the arrival rate leads to a reward that is much harder to determine. A new assumption of the expected reward of a policy is introduced to deal with this problem.

2.2.1 Simulation methods

Scheduling and assignment problems are often modeled and solved by using a simulation-based methodology (Chia and Lin, 2016; Zhou et al., 2018; Azab et al., 2020). A simulation is used to model the behavior of the actual process. In this project, a simulation could be used to model the inbound process of arriving trucks that are being processed (i.e. emptied) by operators of the DC. Mourtzis et al. (2014) categorizes simulations into three types: static, discrete dynamic, and continuous dynamic. A dynamic simulation is dependent on time whereas a static simulation is not. In a dynamic simulation time can be continuous or discrete, in a discrete simulation changes in the simulation occur only at discrete points in time, whereas in a continuous-time simulation this can happen at any point. It can be concluded that the type of simulation needed for this project is dynamic, as time is important.

Within the dynamic simulation techniques system dynamics (SD) and discrete-event simulation (DES) are the most widely used techniques (Jahangirian et al., 2010). Other common simulations techniques include agent-based modeling. Agent-based modeling is used for simulating actions and interactions between agents. This technique is frequently used in social sciences where the operations are defined by interactions between diverse systems. Agent-based modeling is hardly used in literature, Siebers et al. (2010) concludes that this might be due to lack of frameworks, methodologies, and textbooks, that help practitioners use agent-based modeling. Another technique, system dynamics, is mostly applied in domains of policy and strategy development (Jahangirian et al., 2010). Furthermore, it is a continuous-time simulation. Jahangirian et al. (2010) note that SD is most suitable for modeling high-level perspectives, whereas discrete-event simulation is more appropriate for detailed simulations. This finding confirms earlier work by Kellner et al. (1999). As the modeling of the inbound operation involves simulation on a process level, a discrete-event simulation seems to be the most suitable technique for this project.

Discrete-event simulation

Discrete-event simulation (DES) is a simulation technique that is dynamic and discrete. DES is the most popular simulation technique in manufacturing and business (Jahangirian et al., 2010). Specifically, DES is convenient for resource utilization and queuing, which is key in the modeling of the inbound operations. Discrete-event simulations simulate the dynamics of the system on an event-by-event basis. Instead of checking at each time step whether or not an event occurs, DES focuses on future events. These future events can be sorted on time, with each arriving event updating the time of the simulation. Finally, the events occur at different time steps, which makes

DES suitable to model complex systems with numerous processes.

Dutta and Dobe (2016) reviewed open-source software that can be used to create discrete-event simulations for operations research. Although commercial software could be used in this project, it has some disadvantages. Dutta and Dobe (2016) point out that commercial software is often limited in flexibility and reusability. One open-source tool reviewed by Dutta and Dobe (2016), called SimPy, is based on Python generator functions that can be used to create discrete-event simulations. Although the tool has no GUI it is accessible for researchers that are not software experts (Dutta and Dobe, 2016). Furthermore, there are many other scientific packages written in Python such as SciPy (Virtanen et al., 2020a), NumPy (Oliphant, 2006), and Pandas (McKinney et al., 2011), which makes integration easier. Additionally, an extensive array of machine learning packages is available in the Python programming language, such as scikit-learn (Pedregosa et al., 2011), Keras (Chollet, 2015), PyTorch (Paszke et al., 2019), RLlib (Liang et al., 2018), and stable-baselines (Raffin et al., 2019). The packages related to deep learning (Keras, PyTorch) and reinforcement learning (RLlib, stable-baselines) could be useful if the simulation is integrated with a deep reinforcement learning approach. Instead of using a Python package like SimPy, it is also possible to develop a discrete-event simulation from scratch. The advantage of this approach its flexibility and extensibility. However, it might more demanding to develop a discrete-event simulation without using a package like SimPy.

Validation

Before the results of a simulation can be used the simulation has to be validated (Kaizer et al., 2015). One way to validate a simulation is by comparing it to the actual process, however, this can only be done if sufficiently detailed data on the actual process is available. Kaizer (2013) proposes an assessment framework that can be used to evaluate simulations without the need of detailed data about the actual process. Kaizer et al. (2015) extend the work by Kaizer (2013) by demonstrating how this framework can be used.

The framework consists of four steps revolving around the maturity levels of: framework development, requirements selection, simulation assessment, judgement. In the first step it is determined what attributes of the simulation should be assessed and how these can be combined into a framework. In the second step, the minimum required level of maturity for each attribute is determined based on the intended purpose of the simulation. In the third step, the attributes are assessed with respect to the requirements. In the final step, the results of the second and third step are compared and a decision can be made about the adequacy of the simulation.

Even though the proposed framework allows a systematic review, it is still a subjective process (Kaizer et al., 2015). The authors note that the outcome is dependent on the expertise and experience of the assessor. Furthermore, the costs of a formal review using the framework of Kaizer et al. (2015) are high as it is time-intensive. Therefore, a formal review is only needed if the trustworthiness of the simulation for the intended purpose cannot be judged by only one assessor.

Conclusion

This section discussed how the inbound operations of large retailer X can be modeled. First, different methods of modeling uncertainty in operations was discussed. Specifically, the Poisson process seems interesting as it can be used to model the arrival of inbound trucks. Furthermore, the cost function was discussed. It seems that literature suggests this is a complex problem that should be determined and discussed with subject matter experts. Finally, simulation methods were discussed. A discrete-event simulation seems to be the preferred method to model the inbound operations. Additionally, different ways to develop a discrete-event simulation were

discussed as well as ways to validate a simulation. In collaboration with subject matter experts of large retailer X and company Y it was decided a formal review is not feasible for this project due to time constraints. Instead of a formal review, the outcomes of the simulation will be judged and validated in an informal review by experts.

2.3 Reinforcement Learning

In recent years Reinforcement Learning (RL) has been applied to many complex problems where sequential decision-making plays an important role (Mnih et al., 2013, 2015; Silver et al., 2016, 2017; Vinyals et al., 2019). In contrast with other Machine Learning (ML) methods, RL is concentrated on learning from interactions. The goal of RL is to map situations to actions, or learn an optimal behavior policy for an agent interacting with an environment, to maximize a certain reward signal. As reinforcement learning is centered around sequential decision-making it seems a promising method to solve the stochastic sequential assignment problem concerning the inbound operations of large retailer X.

The two most distinguishing features of RL are trial-and-error search and delayed reward (Sutton and Barto, 2018). The learner in RL is called the agent, and the thing the agents interacts with is the environment. The agent is learning through interacting with this environment and through evaluating its actions (trial-and-error). The delayed reward can be imposed by the environment. For example, in chess the reward (winning/losing) is only obtained at the end of a match, however, the actions that led to a win could have happened many timesteps ago. One aspect that all examples of RL share, is the *interaction* between the agent and its environment, where the agents tries to maximize the reward despite the uncertainty about the effects of its actions and the environment (Sutton and Barto, 2018). Another unique challenge in RL is the trade-off between exploration and exploitation. On the one hand, exploration is necessary to discover actions that lead to a higher reward. On the other hand, exploitation is needed to exploit the things the agent already learned to get a good reward. Since it is often impossible to select an action that both explores and exploits, a conflict arises. One way to do exploitation is called *greedy* action selection. Greedy refers to the action that is associated with the current highest expected reward. This greedy action selection can be made slightly less greedy by imposing a constraint with a probability of selecting a random action instead of the greedy action; this is referred to as ε-greedy action-selection. The epsilon controls the possibility of selecting a random action. As this epsilon introduces some randomness, the agent will continue to explore actions other than the greedy action. Even though the trade-off between exploration and exploitation has been studied a lot in the past decades, it remains an unsolved dilemma.

The framework that is often used to model the environment is called the Markov Decision

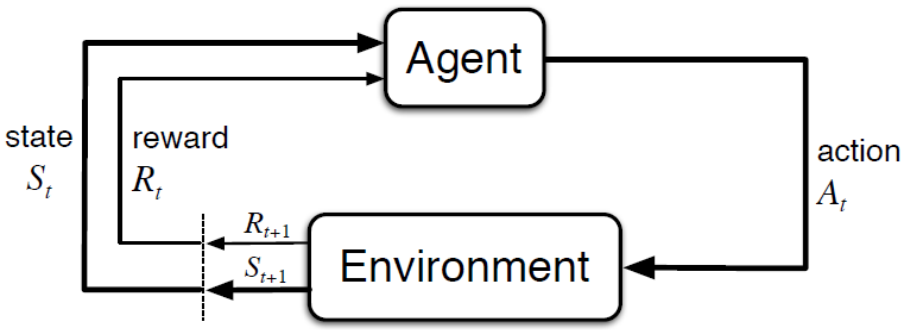

Figure 2.3.1: The agent and environment interaction in a Markov decision process (Sutton and Barto, 2018)

Process (MDP). Fig. 2.3.1 describes the MDP framework visually. The MDP is made up of a state and action space, and a transition and reward function. At each timestep t the agent gets the current state $S_t \in \mathcal{S}$ as an input. Based on the state S_t, the agent determines which action $A_t \in \mathcal{A}(s)$ to take. In the next timestep, the environment returns a reward signal $R_{t+1} \in \mathcal{R} \in \mathbb{R}$ and the new state S_{t+1}. The goal of the agent is to learn a policy π that maps the states to the actions that maximize the expected cumulative sum of rewards.

2.3.1 Approximate Solution Methods

Policy gradients methods can learn a policy without first having to learn action-values. One of the advantages of policy gradients is that it no longer has to perform maximizations over actions to find the optimal policy, therefore, these methods can handle large and also continuous action spaces.

The notation for the policy gradient methods follows that of Sutton and Barto (2018) with $\boldsymbol{\theta} \in \mathbb{R}^{d'}$ for the policy vector with d' referring to the degrees of freedom in the parametrization of the policy. Therefore, the probability of taking action a at time t with parameter $\boldsymbol{\theta}$ can be written as $\pi(a|s, \boldsymbol{\theta}) = P\{A_t = a|S_t = s, \boldsymbol{\theta}_t = \boldsymbol{\theta}\}$. Policy gradient methods learn the parameters of a policy through a performance measure $J(\boldsymbol{\theta})$ with respect to the policy parameter (Sutton and Barto, 2018). This performance measure is defined as follows:

$$J(\boldsymbol{\theta}) = \sum_{s \in \mathcal{S}} d^{\pi}(s) V^{\pi}(s) = \sum_{s \in \mathcal{S}} d^{\pi}(s) \sum_{a \in A} \pi_{\theta}(a|s) Q^{\pi}(s, a) \tag{2.3.1}$$

where d^{π} indicates the stationary distribution of the Markov chain for the on-policy state distribution under policy π, $V^{\pi}(s)$ the state-value of state s and $Q^{\pi}(s, a)$ the action-value of state-action pair (s, a).

Policy gradient methods try to maximize this performance measure $J(\boldsymbol{\theta})$, hence the parameters are updated via gradient ascent:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t + \alpha \widehat{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)} \tag{2.3.2}$$

where $\widehat{\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}_t)} \in \mathbb{R}^{d'}$ refers to a stochastic estimate with an expectation that approximates $\nabla_{\theta} J(\boldsymbol{\theta})$. This means gradient ascent can be used to move the policy vector $\boldsymbol{\theta}$ in the direction suggested by $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ to discover the optimal $\boldsymbol{\theta}$ that maximizes performance.

Another advantage of continuous policy parameterization is that the action selection changes smoothly with the updates, whereas with ϵ-greedy selection even a small update to the action values can cause dramatic changes if that update leads to a different action having a maximal value. Due to this property, policy gradient methods have stronger convergence guarantees compared to action-value methods.

Policy Gradient theorem

A problem arises when $\nabla_{\theta} J(\boldsymbol{\theta})$ needs to be computed. The problem is its dependency on both the action selection and the distribution of states in which those selections are made. Given that action selection and the distribution of states are (indirectly) determined by π_{θ}, and that the environment is typically unknown, it is difficult to estimate the effect of a policy update on the state distribution.

The policy gradient theorem solves this problem by rearranging $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ in such a way that it no longer involves the derivative of the state distribution d^{π}. The proof that the policy gradient

theorem is correct can be found in the book by Sutton and Barto (Sutton and Barto, 2018, p. 325). The result of the theorem is as follows:

$$\begin{aligned}\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}) &= \nabla_{\boldsymbol{\theta}} \sum_{s \in \mathcal{S}} d^{\pi}(s) \sum_{a \in \mathcal{A}} Q^{\pi}(s,a) \pi_{\theta}(a|s) \\ &\propto \sum_{s \in \mathcal{S}} d^{\pi} \sum_{a \in \mathcal{A}} Q^{\pi}(s,a) \nabla_{\theta} \pi_{\theta}(a|s)\end{aligned} \tag{2.3.3}$$

Now that we are familiar with policy gradients it is time to discuss algorithms that use policy gradients. There exist too many policy gradient algorithms to discuss them all, therefore, only a selection of policy gradient algorithms will be discussed. This report focuses on the following algorithms: REINFORCE (Williams, 1992), Trust Region Policy Optimization (TRPO) (Schulman et al., 2015a), Proximal Policy Optimization (PPO) (Schulman et al., 2017), and Phasic Policy Gradient (PPG) (Cobbe et al., 2021). The equations used in the following sections the algorithms are based on the work by Weng (2018).

REINFORCE

The REINFORCE algorithm is on-policy and naturally follows from the policy gradient theorem (Williams, 1992). The policy gradient theorem provides an expression that is equal to the gradient. All that is required is a sampling method that has an expectation that equals or approximates this expression. REINFORCE employs this sampling strategy, which works since the sample gradient's expectation equals the actual gradient. From the proof of the policy gradient theorem of Sutton and Barto (2018) we find the following expression:

$$\begin{aligned}\nabla_{\theta} J(\theta) &= \mathbb{E}_{\pi}[Q^{\pi}(s,a) \nabla_{\theta} \ln \pi_{\theta}(a|s)] \\ &= \mathbb{E}_{\pi}[G_t \nabla_{\theta} \ln \pi_{\theta}(A_t|S_t)] \qquad ; \text{Because } Q^{\pi}(S_t,A_t) = \mathbb{E}_{\pi}[G_t|S_t,A_t]\end{aligned} \tag{2.3.4}$$

where G_t is the return. Since REINFORCE relies on the return it is called a Monte-Carlo method. The part between brackets in the final expression can be sampled at each timestep, which is exactly what is needed. Using these samples stochastic gradient ascent can be used to update the policy. This yields the REINFORCE update:

$$\theta \leftarrow \theta + \alpha \gamma^t G_t \nabla_{\theta} \ln \pi_{\theta}(A_t|S_t) \tag{2.3.5}$$

A commonly used strategy to reduce variance in the gradient estimates are baselines. In REINFORCE with baseline the action-values are compared to a baseline. A baseline can be any function, but a frequently used baseline is subtracting the state-value from the action-value. If this baseline is utilized, then advantage $A(s,a) = Q(s,a) - V(s)$ is used in the gradient ascent update.

Actor-Critic methods

The REINFORCE algorithm with baseline (i.e. using advantage $A(s,a)$) estimates the value of the first state of each state transition. However, this estimation is made before the transition's action and can therefore not be used to to asses this action. Actor-critic methods learn the value function in addition to the policy. This is helpful, since the value fuction can improve the policy update. Actor-critic methods consist of two parts, the actor and the critic. The critic learns the value function ($Q_w(a|s)$ or $V_w(s)$) parameters $\boldsymbol{w}$ and the actor learns the policy parameters $\boldsymbol{\theta}$ for $\pi_{\theta}(a|s)$ using information provided by the critic. Optimizing the actor is similar to optimizing REINFORCE, but instead of using a Monte Carlo estimate of the rewards it uses a learned reinforcing signal. The critic is optimized by the bootstrapped temporal difference learning technique.

Trusted Region Policy Optimization (TRPO)

In normal policy gradient methods small updates to the policy can have great impact on performance. In practice, this means that even a one bad update can drop the performance of the policy significantly. Due to the impact that one bad update can have, the step-size is small for normal policy gradients.

TRPO is different in that it updates the policies by the largest step-size allowed within a certain constraint on the distance between the old and new policy (Schulman et al., 2015a). With this adjustment, TRPO guarantees monotonic improvement. The update of the old policy to a new policy is such that they are a *trusted* distance apart. The constraint is implemented in the form of the Kullback-Leibler (KL) divergence, which is a measure of the difference between one probability distribution (PDF) function and a certain reference PDF. This way we find the following new objective function (Weng, 2018):

$$J(\theta) = \mathbb{E}_{s\sim\rho^{\pi_{\theta_{\text{old}}}}, a\sim\pi_{\theta_{\text{old}}}}\big[\frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}\hat{A}_{\theta_{\text{old}}}(s,a)\big] \tag{2.3.6}$$

subject to

$$\mathbb{E}_{s\sim\rho^{\pi_{\theta_{\text{old}}}}}[D_{\text{KL}}(\pi_{\theta_{\text{old}}}(.|s)\|\pi_\theta(.|s)] \leq \delta \tag{2.3.7}$$

with $\hat{A}$ the estimated advantage, because the actual rewards are unknown.

Proximal Policy Optimization (PPO)

The TRPO algorithm is rarely used in practice because the computation of the KL-divergence is expensive. Similar to TRPO, PPO tries stabilize the updates to the policy (Schulman et al., 2017). PPO simplifies the TRPO algorithm and combats the expensive computation of TRPO by clipping the objective. The objective function of the PPO algorithm is simpler than the objective function of TRPO, while maintaining the same performance. If we introduce the ratio between the new and old policy as $r(\theta) = \frac{\pi_\theta(a|s)}{\pi_{\theta_{\text{old}}}(a|s)}$ then the objective function of the TRPO algorithm can be written as follows:

$$J^{\text{TRPO}}(\theta) = \mathbb{E}[r(\theta)\hat{A}_{\theta_{\text{old}}}(s,a)] \tag{2.3.8}$$

This would lead to instability since there is no longer a constraint on the distance between the old and new policy. Instead of the KL-divergence, PPO imposes a constraint on the ratio $r(\theta)$. This constraint forces the ratio $r(\theta)$ to stay in a certain interval $[1-\epsilon, 1+\epsilon]$ with ϵ as a hyperparameter. In the original paper by Schulman et al. (2017) ϵ was set to 0.2. This constraint clips the estimated advantage $\hat{A}$ when it gets large and discourages large policy changes.

The objective function of the PPO algorithm — or *clipped surrrogate objective* — is to take the minimum between the clipped version and the original version of the estimated advantage $\hat{A}_{\theta_{\text{old}}}(s,a)$. Therefore, the objective function for PPO can be written as follows:

$$J^{\text{CLIP}}(\theta) = \mathbb{E}[\min(r(\theta)\hat{A}_{\theta_{\text{old}}}(s,a), \text{clip}(r(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_{\theta_{\text{old}}}(s,a))] \tag{2.3.9}$$

Phasic Policy Gradient (PPG)

Since its release PPO has been the de-facto standard policy gradient algorithm. A recently released paper by Cobbe et al. (2021) introduced the Phasic Policy Gradient (PPG) method, which is more sample-efficient than the PPO algorithm. This is achieved by separating the policy and

value function training into distinct phases. A trade-off has to be made between sharing or not sharing features between policy and value networks, PPG aims to achieve the best of both.

The PPG algorithm consists of two phases, the *policy phase* and the *auxiliary phase*. In the policy phase the agent is trained by using PPO. During the auxiliary phase features are extracted from the value function into the policy network. Like PPO, the policy network is trained using the clipped surrogate objective as in Equation 2.3.9. To train the value function network the optimization is done as follows:

$$J^{\text{VALUE}} = \hat{\mathbb{E}}_t \left[\frac{1}{2} (V_{\theta_V}(s_t) - \hat{V}_t^{targ})^2 \right] \tag{2.3.10}$$

where $\hat{V}_t^{targ}$ are the value function targets. The value function targets and the estimated advantage can be computed with generalized advantage estimation (GAE) (Schulman et al., 2015b).

The auxiliary phase optimizes the policy network with a objective that includes a behavioral cloning loss in addition to the auxiliary objective. This joint objective and auxiliary objective are as follows:

$$\begin{aligned} L^{\text{joint}} &= L^{\text{aux}} + \beta_{\text{clone}} \cdot \mathbb{E}_t[\text{KL}[\pi_{\theta_{\text{old}}}(\cdot \mid s_t), \pi_\theta(\cdot \mid s_t)]] \\ L^{\text{aux}} &= L^{\text{value}} = \mathbb{E}_t\big[\frac{1}{2}\big(V_w(s_t) - \hat{V}_t^{\text{targ}}\big)^2\big] \end{aligned} \tag{2.3.11}$$

with β_{clone} controlling the trade-off between optimizing the auxiliary objective and retaining the original policy.

Conclusion

This section introduced the topic of reinforcement learning. In the introduction the importance of the MDP framework was discussed. After the MDP framework, approximate solutions were introduced. These methods can learn a policy without having to learn action-values. Next, actor-critic methods were discussed which form the basis for various advanced policy gradient methods, such as TRPO, PPO, and PPG. Specifically, approximate methods seem promising as these do not have to learn action-values and are much faster in general in terms of wall-clock time. Furthermore, the policy gradient methods such as TRPO, PPO, and PPG have constraints on policy updates to increase stability during training.

2.4 Deep Reinforcement Learning

One of the downsides of the RL algorithms is that they need a lot of computation to achieve reasonable results. When the state-action space becomes large, it even becomes infeasible to achieve reasonable results. However, recent advancements in Deep Learning have provided new opportunities for RL algorithms. Instead of finding Q-values or policies exactly, neural networks can be used to approximate these.

2.4.1 Deep Q Networks

Mnih et al. (2013) were the first to successfully combine RL with Deep Learning. In a follow-up paper, the authors use a Deep Q-Network (DQN) to master control policies for various Atari 2600 games (Mnih et al., 2015). As the complexity and size of the state-action space increases, it becomes increasingly complex to calculate and memorize the Q-values. Instead of learning

control policies directly, a neural network is used to approximate the policy. In the case of the assignment problem in this research, the RL agent must, at each time a truck arrives, assign the truck to one of the available docks. Due to the large number of docks the number of pairs of states and actions becomes substantial. Therefore, it seems that this project can benefit from using neural networks as function approximators instead of exactly finding the policy.

Even though the DQN was successfully applied it can become unstable in certain situations. Instability and convergence issues arise when the following three elements, known as *the deadly triad*, are present: function approximation, bootstrapping, and off-policy training. Q-learning uses the latter of these two techniques, it bootstraps via the TD update rule, and it uses off-policy training. Hence, with the introduction of non-linear function approximators such as neural networks in the DQN, the deadly triad is activated. Mnih et al. (2015) solved the problem of instability and convergence by introducing *experience replay* (Lin, 1992). With experience replay the experiences of the agent are stored in a *replay memory* at each time step. This way, these experiences can be played again as if they were experienced again. These experiences can be sampled from the replay memory and inter weaved with new experiences. An advantage of experience replay is that it removes correlations in the observations by sampling experience from memory, which in turn leads to better convergence. An extension to experience replay was proposed by Schaul et al. (2015) in which the probability of sampling experience from the replay buffer is based on the TD-error of the previous update. When a TD-error was large the update was apparently important, taking this into account, prioritized experience replay can give more weight to samples that contain valuable information towards learning an optimal policy. It may come as no surprise that the implementation by Schaul et al. (2015) with prioritized experience replay indeed converges faster compared to the solution of Mnih et al. (2015).

An extension to the normal DQN is that of *dueling networks* (Wang et al., 2016). Q-learning uses a maximization over the Q-values, as these Q-values are noisy the estimate of the Q-values is often an overestimation of the actual value. To overcome this bias and overestimation dueling networks can be used. Dueling networks use two networks, the first network approximates the value function $V(s)$ and the second one the advantage function $A(s, a)$. Subsequently, the results of these two networks are combined to find the action-value function $Q(s, a)$. An additional extension to DQN is the Double DQN (DDQN) introduced by Van Hasselt et al. (2016). The Double DQN architecture consists of two identical neural networks, one learning from from experience replay and the other a copy of the model of the previous episode. The key idea is to have two separate networks for the maximization and the selection step. Similar to a dueling network this can reduce the overestimation of the Q-values. There are many more extension to DQN, such as: multi-step learning, noisy nets, and distributional RL. All of these extensions, including dueling networks and DDQN, are included in the Rainbow algorithm which showed state-of-the-art performance on the Atari benchmark (Hessel et al., 2018).

For approximate solution methods, neural networks can be used as a function approximator. As such, the output of a neural network can be used as a distribution over the possible actions. Exploration is ensured by sampling from this distribution. As approximate solutions methods were already elaborately discussed in the Section 2.3.1 it will not be discussed further in this section.

2.4.2 Invalid action-masking

In many real life applications a reinforcement learning agent is restricted from choosing certain actions. Consider the case of this project, an arriving truck cannot be assigned to a dock that is currently occupied. One solution to this problem is to give a negative reward to the agent if it chooses an invalid or infeasible action. However, there is simpler solution which prevents the agent from taking invalid action altogether, this solution is called *invalid action-masking*. In policy

gradient methods such as PPO, invalid action masking sets the probabilities of the actions that are invalid to zero in the neural network. This way, the agent can only sample its next action from valid actions. Although this technique has been used in several famous research papers, such as Silver et al. (2016), the technique is not explained in-depth. Huang and Ontañón (2020) are the first to explain this technique in-depth and show its implications. In their paper Huang and Ontañón (2020) show that invalid action-masking is of critical importance for performance, especially for policy gradient methods. Additionally, they show that invalid action-masking scales well for larger problem instances in which giving a negative reward is no longer sufficient.

2.4.3 Benchmarks

Since the introduction of the paper by Mnih et al. (2013) interest in the development of environments in which Deep Reinforcement Learning (Deep RL) algorithms can be tested and compared increased. Simple examples of environments already existed for training purposes, such as the inverted pendulum — also known as the cartpole — introduced by Anderson (1989). The cartpole is a simple control task in which the goal of the agent is to learn how to balance a cartpole without any a priori knowledge about the dynamics of the environment. In 2016 OpenAI introduced an open-source toolkit for RL research that includes the cartpole called OpenAI gym. Since then the cartpole has become one of the prime examples of a RL control task (Brockman et al., 2016). Fig. 2.4.1 shows a frame of the Cartpole-v0 environment of the OpenAI gym toolkit. Next to simple control problems, OpenAI gym includes classic Atari games, algorithmic problems, board games (including Go), and 2D and 3D robotic control problems that use the MuJoCo physics engine (Todorov et al., 2012). Furthermore, the OpenAI gym interface has become the standard for modeling reinforcement learning environments. Many other reinforcement learning packages can be integrated to work with the OpenAI gym interface. Therefore, developing a new environment with the OpenAI gym interface in mind can allow access to exiting RL algorithms implemented in packages such as stable-baselines. Furthermore, adhering to the OpenAI gym standard when designing a new environment could enhance accessibility and collaboration of other reseachers.

In recent years, Deep RL has also been frequently applied to problems in the Operations Research (OR) domain. Although OpenAI gym offers many applications, it is often not relevant for operational optimization problems. In order to explore applications more relevant for the industry, and specifically the OR domain, Hubbs et al. (2020) introduced the OR-gym library. This library extends and elaborates on benchmarks by Balaji et al. (2019) and has a similar interface as OpenAI gym. OR-gym provides environments related to prevalent problems in the OR domain such as bin packing, knapsacks, supply chain inventory management, vehicle routing, and asset allocation. Hubbs et al. (2020) show that RL is capable of learning a good policy for each of the problems introduced in OR-gym, for complex environments where uncertainty plays a role the RL algorithms often outperform traditional solutions or heuristics. As a stochastic sequential assignment problem is not yet part of OR-gym, it might be interesting to provide an environment for this problem should the Master's thesis project become a success.

2.4.4 Applications of Deep Reinforcement Learning

In recent years, Deep Reinforcement Learning (Deep RL) has been successfully used to solve problems in many fields including the field of OR and logistics. Although no research has been done on applying Deep RL to the SSAP yet, related literature can provide information on state and action space representation, reward functions, and other challenges in implementations. This section discusses applications of DRL in the field of operations management and logistics.

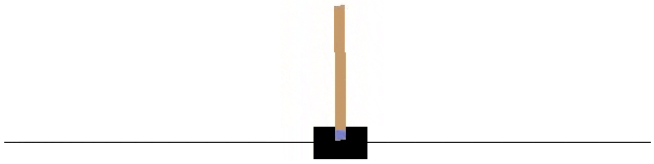

Figure 2.4.1: Frame of the OpenAI gym Cartpole-v0 environment (Brockman et al., 2016)

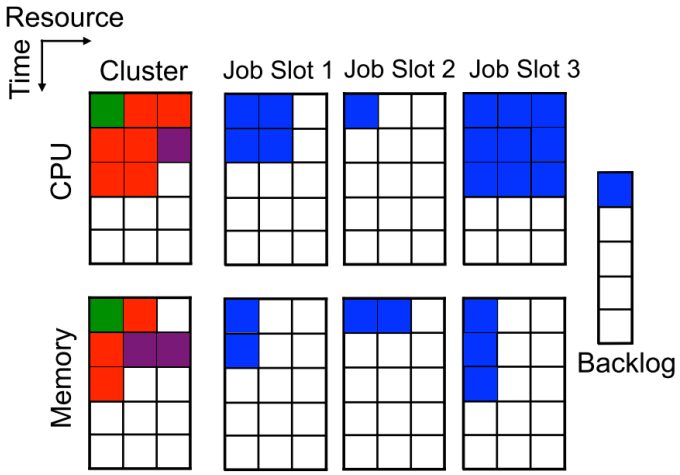

Figure 2.4.2: State representation of the cluster (2 resources) with each 3 slots for queuing jobs (Mao et al., 2016)

Resource management

Mao et al. (2016) consider the situation of cluster scheduling in which incoming jobs with a known computation time need to be allocated to a cluster. The goal is to minimize the average slowdown (i.e. the completion time over the ideal duration) of jobs arriving at the cluster. The state is represented by images of the clusters and profiles of the jobs in the queue, as displayed in Fig. 2.4.2. In the state representation, the colors indicate the computational resources (e.g. 1 CPU block, and 2 memory blocks) required for that job. The backlog is introduced to keep the state space fixed, this is desirable for the input of the state into a neural network. At each time step, the agent needs to allocate jobs to a slot, as the backlog can be infinitely large, the actions space can become very large as well.

The authors propose a "trick" that allows the agent to take multiple actions per time step, the agent can allocate jobs until a void or invalid action is chosen, this way the action space is still linear in the size on the backlog. The reward function is simple, for each time step a job is the system (i.e. processing or in the queue) the agent receives a negative reward. To train the agent, the authors propose to use REINFORCE. It should be noted that, more advanced policy gradient methods such as PPO did not exist yet. The performance of the proposed approach is comparable and sometimes better compared to heuristics.

Order batching

Cals et al. (2021) studied a DRL approach for the online order batching and sequencing problem

(OOBSP). The OOBSP is concerned with batching and picking decisions of arriving orders in a warehouse, such that the number of tardy orders is minimized. The problem is modeled as a Semi-Markov Decision Process (SMDP) instead of an MDP since time plays an important role in the environment. With the SMDP the time to transition τ is flexible, which allows this time to be represented by the time until the next order arrives. The state space is composed of the current remaining orders for picking, capacity available in the warehouse, the number of tardy orders, the number of processed orders, and the current simulation time. The authors note that more information could be included in the state, however this comes at the cost of computation time. With 15 different orders that can be picked in two ways, there are 30 possible actions. Since capacity is limited a "wait" action is included such that the action space comprises 31 actions in total. The reward function is constructed such that the majority of the reward is provided at the end of the episode. Penalties during the episode for tardy orders and infeasible action are intentionally kept low to discourage the agent from not taking any actions at all. Finally, the authors propose a DRL approach that uses PPO to train the agent because it requires less computational power than a DQN. The parameters used in the training algorithm are similar to that of Schulman et al. (2017), except that authors use a higher discount rate to emphasize the focus on future rewards. The results show that the Deep RL agent outperforms heuristics in most cases, however, the authors note some limitations regarding the interpretability of the learned policies.

Online bin packing problem

The bin packing problem is an optimization problem the has been frequently studied in the field of operations research and computer science. In the problem items with different sizes must be packed into fixed size bins, such that, the number of bins used is minimized. In the online version of the bin packing problem the items arrive individually with a size sampled from an unknown distribution (Balaji et al., 2019). The online variant of the bin packing problem shares similarities with the truck docking problem of this project as the supply is stochastic.

Balaji et al. (2019) solve the online bin packing problem using DRL. The authors use PPO with a two-layer neural network of 256 nodes each. Action masking is used to prevent the agent from taking invalid actions, this is done by assigning a zero probability to invalid actions. The state space consists of the current item and its size, and the number of bins. The reward is given by the total "waste" which is represented by the total empty space in the bins, the goal of the agent is to minimize the total waste.

The results of the DRL solution are compared to two baseline heuristics — Sum of Squares and Best Fit. Different scenarios are tested, these scenarios differ in bin size, number of items per episode, and the distribution used for sampling the item size. For each of the scenarios the RL policy matches or outperforms the baseline heuristics. The generalization of the trained agent is tested by using different distributions for the item sizes compared to what the agent was not trained on. The agent generalizes well on two distributions, but for one distribution it did not generalize as well. On this distribution the agent started to overfit and performed worse than the baselines. The authors suggest the overfitting and generalization as topics for future research.

Finally, it is interesting to note that the authors publish the training time for the algorithms. In the scenario in which the DRL agent outperforms the baselines, the optimal level of performance is achieved after 600 minutes of training. For reference, the DRL agent was trained on a single machine with 4 GPUs and 32 CPUs.

Online knapsack problem

The online knapsack problem is another well-studied combinatorial problem. In the online version of this problem there is a knapsack with fixed capacity and items arriving sequentially, associated with each item is a size and value. When an item arrives it must be either accepted or rejected, the goal is to maximize the total value of items in the knapsack without violating its capacity.

Kong et al. (2019) studied how DRL can be used to optimize the online knapsack problem. The authors' goal is to show how DRL can be used to find the classic "pen-and-paper" algorithms, as such, hyper-parameters and training setup are not tuned. The authors' also do not use action-masking or similar techniques to speed up training.

The authors use the REINFORCE algorithm to train the agent. The state space contains four variables — the value and size of the arriving item, the proportion of the knapsack already filled, and the fraction of the queue that has already been seen. As the agent can only accept or reject the arriving item the action space is small. The agent receives a positive reward equal to the value of the item added to the knapsack if it does not violate the capacity constraint; otherwise the reward is 0. The performance of the DRL model is compared with the Bang-for-Buck algorithm introduced by Dantzig (1957). Even though the models are not tuned, the results show the DRL model achieves a performance close to the baseline algorithm. A policy analysis confirms that the DRL model indeed learns a policy similar to the Bang-for-Buck algorithm.

Conclusion

In this section the value of combining neural networks with reinforcement learning was explained. Subsequently, the benefits of invalid action-masking were discussed. It seems that invalid action-masking could be interesting for the truck docking problem of this project. For example, when assigning trucks to docks, a truck cannot be assigned to a dock that is currently occupied. In this case invalid action-masking could be used to speed up training. Next, various toolkits that can be used for benchmarking algorithms were discussed. It was discussed how the OpenAI gym interface has become the standard interface for RL environments. Therefore, it seems useful to develop the environment for the truck docking problem of this project in accordance with the OpenAI gym interface. Additionally, as a stochastic sequential assignment problem is not yet part of OR-gym, it might be interesting to provide an environment for this problem should the master's thesis project become a success. The last part of this chapter introduced various applications of deep reinforcement learning to problems in operational decision-making and combinatorial optimization. These applications show the details related to applying deep reinforcement learning outside an academic setting.

2.5 Reinforcement Learning Interpretation methods

Section 2.4.4 showed various applications of Deep Reinforcement Learning (DRL) in the domain of operational decision-making. However, DRL has also been successfully appliad outside the domain of operations and combinatorics repeatedly. Despite the performance of DRL methods, their black-box nature (due to the use of neural networks) complicates the support of DRL in practice (Alharin et al., 2020). To overcome this problem, researchers have tried to interpret the policy learned by the agent. This chapter discussed the techniques and methods that can be used to interpret policies learned by DRL agents.

2.5.1 Explanation type

In general, three types of interpretation methods for DRL models can be distinguished (Alharin et al., 2020). First, a new method can be introduced to interpret the model while keeping the original DRL model as it is (Post-Hoc). Second, a new interpretable learning model can be used to substitute the original way of learning (Intrinsic). For example, a decision tree (which is naturally interpretable) can be used to replace the complex model (Roth et al., 2019). Third, a combination of both method can be used. In this combination part of the original model can be kept. Furthermore, the interpretation methods can explain either local or global decisions. Explaining local decisions is about specific actions of the agent, whereas the global decision are more about the global behavior and strategy of the agent.

Another distinction can be made between the time at which the interpretation is applied. The interpretation can be applied before training the model, which gives insight into the environment that the agent is interacting with. Applying the interpretation during learning can give insights into the learning procedure. When applied during learning it can be determined if the agent is just randomly changing weights, or actually learning something meaningful (Carvalho et al., 2019). If the interpretation is applied after training, insights about what the model actually learned can be derived. As intrinsic interpretation methods are no longer complicated by a black-box the time of application of the interpretation method becomes irrelevant. Fig. 2.5.1 shows the ontology of interpretation methods for DRL methods.

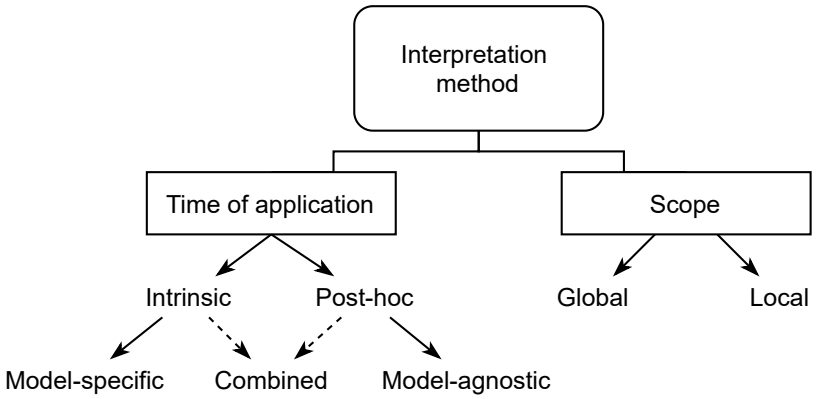

Figure 2.5.1: Pseudo ontology of interpretable methods for DRL. Adapted from Puiutta and Veith (2020) and Adadi and Berrada (2018).

This review specifically focuses on post-hoc local methods The post-hoc local methods can be used to identify which features of an observation were critical for an agent's action. On the other hand, the post-hoc global methods can identify the steps and actions that were critical for achieving the final reward. As the field of explainable DRL is very active, the methods discussed in this section are non-exhaustive. Only the most relevant methods to the research described in the introduction (Chapter 1) are discussed.

2.5.2 Post-hoc local methods

This section discusses post-hoc local methods for interpreting DRL models. More specifically, this section discusses methods that use model approximation methods such as tree based models.

Tree-based

Tree-based methods are self-interpretable models that can be used to mimic the behavior of a DRL agent. Once the DRL agent is mimicked the interpretation can be derived from the self-interpretable model. Next to an interpretation, heuristic rules can be extracted from the decision tree. Beeks et al. (2022) showed that heuristics based on a decision tree that mimicks a DRL agent can outperform advanced heuristics. Adadi and Berrada (2018) classifies tree based methods as post-hoc model-agnostic global methods as they can extract a set of rules from the policy of a trained agent. However, these rules can only map observations to actions, they do not give insight into how much, and which step contributed to the final reward. Furthermore, the DRL agent can learn to take future events into account which is not possible with a tree-based method. Therefore, tree based methods are considered post-hoc local methods in this review.

Bastani et al. (2017) proposed to use tree-based models to interpret black-box models such as neural networks and random forests models. These models were trained and evaluated on data sets provided by the UCI repository as well as simple OpenAI gym environments such as cartpole. The paper clearly discusses how the extracted model can be used to provide insight into what features led to which decisions. For example, the paper shows that the occurrences of the *chloride* feature in the *wine origin* data set correlated nearly perfectly with the low performance of the neural network. Bewley and Lawry (2020) builds on the idea of Bastani et al. (2017) by modeling the value function, policy, and temporal dynamics in conjunction.

Soft Decision Trees (SDTs) were first introduced by Frosst and Hinton (2017). SDTs combine binary decision trees with a predetermined depth and neural networks. The idea is to train a neural network on data generated by an agent and subsequently train a decision tree to mimic this neural network. The decision tree uses the filters generated by the neural network to create hierarchical decisions that are inherently interpretable. Although the performance is worse than the neural net, this trained SDT generalizes better than fitting a regular decision tree on a dataset as proposed by Bastani et al. (2017). Coppens et al. (2019) uses the idea of Frosst and Hinton (2017) to distill the output of the policy network into an SDT. The method of Coppens et al. (2019) is evaluated on the Mario AI benchmark (Karakovskiy and Togelius, 2012) and shows to perform worse than the PPO algorithm from which the SDT was derived. Furthermore, the authors note that the SDT still lacks human-readable abstractions as it is based on the filters generated by the neural network. Another drawback is the depth of the SDT has to be set manually before training the model, this was later solved in the work of Tanno et al. (2019) in which the depth is adaptive.

2.5.3 Local interpretable model-agnostic explanations (LIME)

Local interpretable model-agnostic explanations, or LIME, can be used to explain any classification method (Ribeiro et al., 2016). As LIME does not require any prior knowledge of the model, it can be used to explain any method, thus it is model-agnostic. LIME can interpret the original model by using a surrogate model that can approximate the original model locally. The idea is that although the global model can be complex, a single prediction can be made with a much simpler model. LIME is an additive feature attribution method that uses a linear function as an explanation model:

$$g(z') = \phi_0 + \sum_{i=1}^{M} \phi_i z'_i \tag{2.5.1}$$

Where $f(x)$ represents the original function and $g(z')$ the approximated function, $\phi_i \in \mathbb{R}$ the contribution of feature i, and $z' \in 0, 1^M$ where M is the number of input features. In order to find

the contribution ϕ of each feature, LIME minimizes the following objective function:

$$\xi = \arg\min_{g \in \mathcal{G}} \mathcal{L}(f, g, \pi_{x'}) + \Omega(g) \tag{2.5.2}$$

Where g is an explanation model out of $\mathcal{G}$ possible models. To avoid the explanation models to grow too complex complexity measure $\Omega(g)$ is added. This $\Omega(g)$ could, for example, be the depth of a decision tree. As a proximity measure around the prediction instance $\pi_{x'}$ is used. Finally, $\mathcal{L}(f, g, \pi_{x'})$ represents how faithful the model g is to the original model f. Through minimizing the loss term $\mathcal{L}(f, g, \pi_{x'})$ and keeping the model interpretable through $\Omega_{(g)}$ a trade-off is made between fidelity and interpretability of the model. Because LIME uses a linear function as the explanation model (Eq. 2.5.3) the objective function Eq. 2.5.3 can be solved using penalized linear regression.

2.5.4 Shapley additive explanations (SHAP)

Shapley Additive Explanations, or SHAP, is a unified measure of feature importance (Lundberg and Lee, 2017). SHAP is based on cooperative game theory and can explain any machine learning model by assigning credit for the model's output to each input feature. The main benefit of SHAP over LIME is that it can decompose the output of the model and contribute it to each feature where the sum of the contributions sums up to the original model whereas LIME cannot do this. Similar to LIME, SHAP is a feature attribution method based on Eq. 2.5.3 where $f_x(z') = E[f(z)|z_S]$, where S are the indices that are currently used. SHAP relies on a simplified input where features are missing to determine the influence of the missing feature. Because most models cannot handle missing features it is approximated by $E[f(z)|z_S]$. In SHAP, the ϕ_0 of Eq. 2.5.3 is the baseline which represents the expected output when there is no a priori knowledge about the features. The so-called SHAP values ϕ_i are added to the baseline to find the final output of the model. Fig. 2.5.2 shows how SHAP values can be used to explain the influence of features on a single prediction. Along with the paper by Lundberg and Lee (2017) the code to

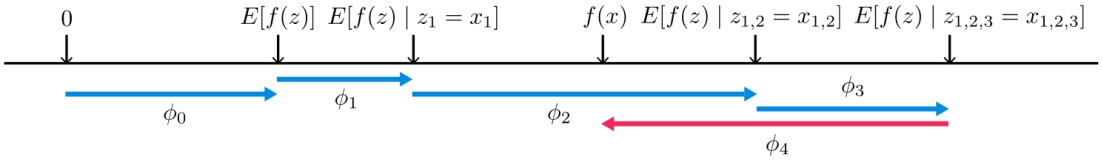

Figure 2.5.2: Using SHAP to explain a single prediction. The SHAP values explain how to transition from baseline $E[f(z)]$ to the output of the original model.

use SHAP was publicly released. The code can be used to easily find SHAP values for any machine learning method and to visualize the found SHAP values. For example, the global feature importance and a summary of local explanation summary can be visualized using SHAP, as can be seen in Fig. 2.5.3. The left part of Fig. 2.5.3 shows the global feature importances based on the mean SHAP values. The right part of Fig. 2.5.3 shows a summary of local explanations. The color hue indicates the value of the feature, the location on the x-axis indicates the impact on the model output measure by a SHAP value. For example, the first feature "Age" has a negative impact on the model output when the age is low and a positive impact when the age is high.

Application of SHAP in the RL domain

Until recently, SHAP values were mostly used to explain supervised black-box machine learning methods. The work by Liessner et al. (2021) shows how SHAP values can be used to explain the behavior of an RL agent which the authors call RL-SHAP. In this paper SHAP values are combined with the actions selected, which allows for creating representations in which the influence

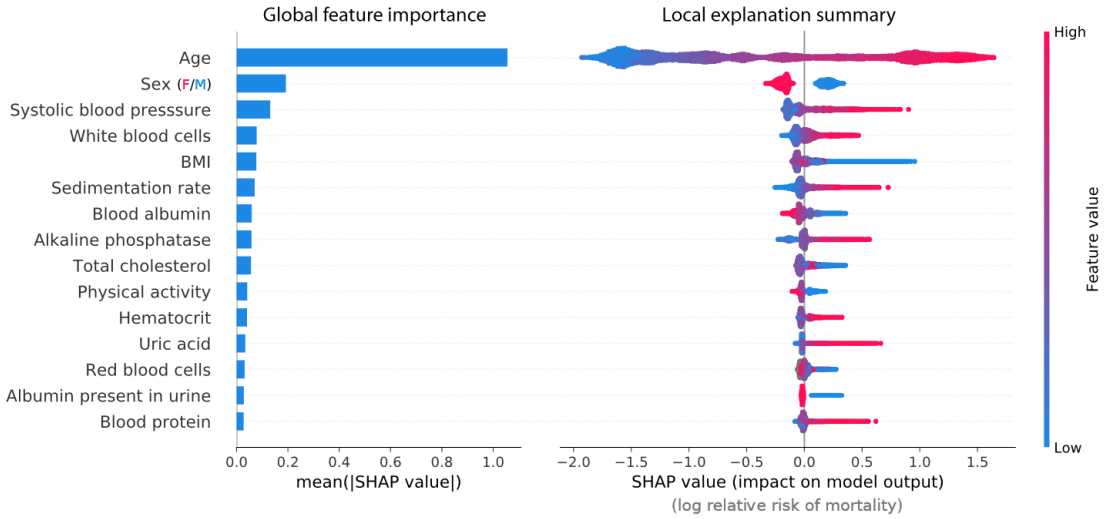

Figure 2.5.3: Global feature importance and local explanation summary plot example for a tree-based model for the chronic kidney disease data set (Lundberg et al., 2020).

of each feature on the agent's decision can be observed over time. Fig. 2.5.4 shows the RL-SHAP plot for the longitudinal control of a car that was analyzed in Liessner et al. (2021). The first subplot shows the velocity over time, the second subplot the actions of the agent over time, and the other subplots show the feature values and SHAP values over time. The color hue indicates the SHAP value. The blue hue indicates a negative influence on the model's output, whereas a red hue indicates a positive influence on the model's output.

2.5.5 Conclusion

This section provided an introduction to interpretation methods for DRL algorithms. First, the different types of explanations given by different interpretation methods were discussed. Second, post-hoc local methods were discussed. These methods could be used to explain what features of a state S_t were important in the decision of the agent to choose action A_t. The main post-hoc global method that was discussed could provide information on which time steps in an episode were important in achieving the final episodic reward.

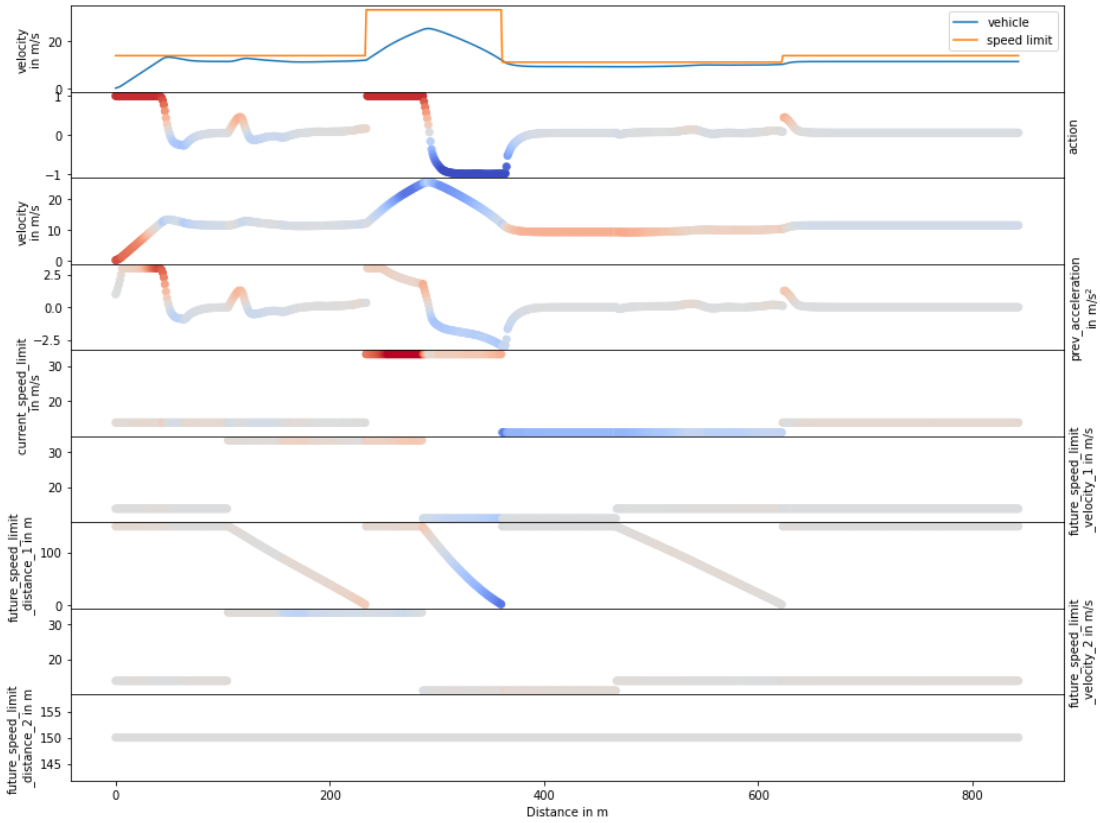

Figure 2.5.4: Example of an RL-SHAP diagram for longitudinal control of a car (Liessner et al., 2021)

2.6 Automated Reinforcement Learning (AutoRL)

This section discusses automated reinforcement learning (AutoRL) methods. The field of automated machine learning (AutoML) for supervised methods has been quite active in recent years. Only recently, the field of AutoRL got attention, hence there currently is not much literature available on this topic. Therefore, this thesis might be able to make a meaningful contribution to this new field.

2.6.1 AutoRL framework

Before beginning the learning process, AutoRL provides a framework for automatically making decisions regarding the settings of an MDP or RL algorithm. (Afshar et al., 2022) provides an overview of the RL pipeline containing the elements that can be optimized, this overview is shown in Fig. 2.6.1. It can be seen the AutoRL pipeline has multiple components. First, the process of modeling the MDP can be automated by automating state, action, and reward. Second, the RL algorithm used to train the agent can be automatically selected. Third, the hyperparameters can be optimized automatically. The problem of this project is not as complex as other problems found in literature, therefore, it seems algorithm selection and hyperparameter optimization might not be as relevant to this project compared to MDP modeling. Specifically, Afshar et al. (2022) mentions there is a lack of a generic state representation methods in literature. For that reason, the remainder of this section focuses on literature related to automatically defining the state space of the MDP.

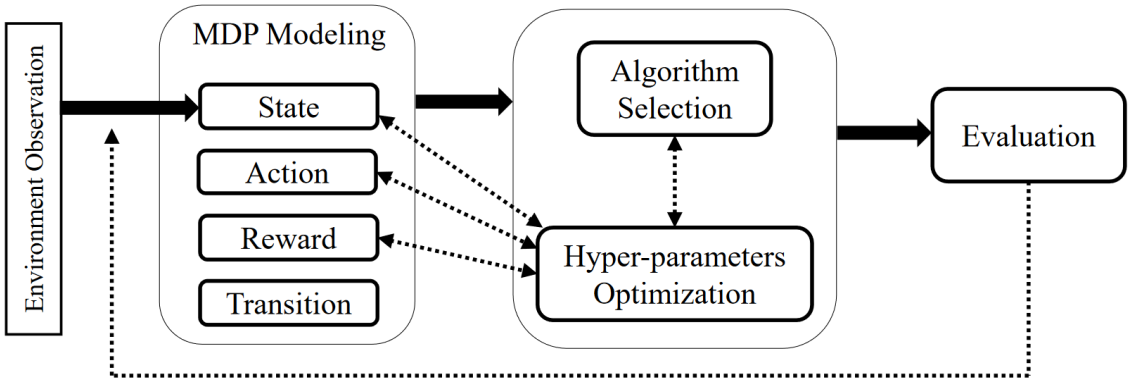

Figure 2.6.1: Elements of the RL pipeline. Dashed lines indicate the AutoRL pipeline that can be optimized based on evaluation results.

2.6.2 Automatically defining the state space

In practice, often all information that is deemed important by experts is part of the state space. This might lead to large state spaces with could be cause issues with learning a good policy. First, the curse of dimensionality necessitates a large amount of memory. Second, and perhaps more importantly, the number of state-action pars complicates the learning process because it becomes increasingly difficult for the agent to become familiar with each possible state.

One technique to deal with a large state space is to decrease its size and complexity by aggregating states that have comparable characteristics. Baumann and Buning (2011) studies state aggregation for a Q-learning algorithm with a continuous state space. To automatically learn state aggregation during training, Baumann and Buning (2011) introduces GNG-Q which combines Q-learning with Growing Neural Gas (GNG). GNG is an unsupervised method that uses a minimal collection of units to describe the topology of an environment. It adds units and adjusts the approximation by interacting with the environment until a representation is obtained. GNG-Q focuses on regions of the state where the policy of the agent changes often. After finding a sufficiently fine state representation the algorithm reduces to a normal Q-learning algorithm.

Recently, Refaei Afshar et al. (2020) used state aggregation to extract features and states for solving the knapsack problem. The idea is based on discretizing the feature values of items, where tabular RL is used to learn the best way to aggregate each item. This way, the number of unique values is greatly reduced. Results show that the agent learns a close to optimal solution for all tested problem instances. Furthermore, by using the state aggregation larger problem instances can be handled compared to existing DRL methods. Next to that, the approach can find an optimal solution in less timesteps compared to an approach without state aggregation.

Conclusion

This section briefly discussed AutoRL. First, the RL pipeline was discussed. This showed the elements that could be optimized automatically. Second, literature regarding automatically defining the state space was discussed. After manually selecting features to be included in the state space an agent can be trained to find a good policy. Subsequently, a post-hoc local method could be used to identify the feature importances. Using this information, features that are not important in the agents decision making could be excluded from the state to avoid the curse of dimensionality and speed up the learning process.

2.7 Position of this research in literature

The literature review described in this chapter touches upon relevant literature to the problem of optimizing truck to dock assignment as discussed in Chapter 1. The main interests are (1) understanding and modeling of the inbound operations, (2) understanding how Deep Reinforcement Learning can be applied to optimize the truck to dock assignment problem, and (3) finding the best way to assign trucks to docks.

First, Section 2.1 introduced assignment problems in general. Specifically, the classical assignment problem (CAP) and the stochastic sequential assignment problem (SSAP) and their solution methods and applications were discussed. It can be concluded that the problem that large retailer X is facing in its DC's is closely related to the SSAP described in literature. However, the problem of large retailer X has unique characteristics. For example, the uncertainty lies not with the rewards of arrivals but with the arrival time. Additionally, there is no success rate associated to agent completing jobs. Although the problem of large retailer X is unique in the sense that it cannot be found literature (to the best of the author's knowledge), it could be easily generalized to other DC's as well.

Second, Section 2.2 explored literature on modeling inbound operations. The uncertainty in modeling the operations was discussed first. From literature it seems that a Poisson process is often used in queuing theory to model random events, for example the arrival of trucks.

Third, the topic of sequential decision-making using Reinforcement Learning (RL) was introduced in Section 2.3. First, the MDP framework that is essential to RL was explained. Next, tabular methods to find exact solutions were explained. Although these methods are rarely used in practice due to their computational inefficiencies, they lay the foundation for more complex RL algorithms. Following the tabular methods, approximate solution methods were introduced. These methods do not have to learn action-values and are able to learn a policy directly. Specifically, REINFORCE, TRPO, PPO, and PPG were discussed.

Fourth, Section 2.4 introduced the concept of Deep Reinforcement Learning (DRL), which combines deep neural networks with RL. First, Deep Q networks and some its extensions were discussed. Afterwards, it was explained how neural networks can be used as a function approximator for approximate solution methods such as PPO. Subsequently, the benefits of invalid action-masking were discussed. It seems that invalid action-masking could be useful for the truck docking problem to speed up training. For example, when assigning trucks to docks, a truck cannot be assigned to a dock that is currently occupied. After that, the OpenAI gym and OR-gym toolkit were discussed. As the OpenAI gym interface is the de facto standard for modeling a RL environment it seems the environment for this project should be developed in accordance with the OpenAI gym interface. Finally, various successful applications of DRL to problems in operational decision-making and combinatorial optimization problems were discussed. The literature showed that DRL can indeed successfully be applied in problems similar to the truck docking problem of this project.

Fifth, interpretation methods for DRL were discussed. Due to the black-box nature of DRL there is a lack of support for such methods in practice. First, the ontology of interpretable methods for DRL was explained. The main distinctions are the time of application, the scope, and the model. Subsequently, post-hoc local methods and post-hoc local methods were discussed. Post-hoc local methods can be used to explain which features influenced the agent's decision to take a certain action. In contrast, post-hoc global methods can be used to explain which time step was critical in achieving the final episodic reward. To obtain a deep understanding of the agent and to create acceptance of a DRL agent interpretation methods could be useful in the truck docking problem of this project. For example, it could be interesting to use post-hoc local methods to derive features importances for the DRL agent. Post-hoc global methods could be applied as well,

however, it is not expected that this can provide new insights since experts do not expect one specific time step to be crucial for achieving the episodic reward. From literature it can be concluded that there is a lack of knowledge on the application of interpretable DRL methods outside of "toy" environments, therefore the application of such methods on the truck docking project would add to the existing body of literature.

Sixth, automated reinforcement methods were briefly discussed. Only recently, the field of AutoRL got attention, hence there currently is not much literature available on this topic. First, the RL pipeline was discussed. This showed the elements that could be optimized automatically. Second, literature regarding automatically defining the state space was discussed. After manually selecting features to be included in the state space an agent can be trained to find a good policy. Subsequently, a post-hoc local method could be used to identify the feature importances. Using this information, features that are not important in the agents decision making could be excluded from the state to avoid the curse of dimensionality and speed up the learning process.

Finally, it can be concluded that the master's thesis project fills five distinct gaps. First, the problem setting of the assignment problem in this project is a unique variant of a SSAP. Second, although DRL has been applied to similar problems such as the online bin-packing and knapsack problem, the application of DRL to this setting, but also to the closely related SSAPs, is novel. Third, the application of interpretable and explainable DRL methods on practical problems adds to the existing body of literature. Fourth, using post-hoc local methods like SHAP to select features to be included in the state space is novel and could contribute to the existing body of literature on AutoRL. Finally, the thesis project aims to provide an open-source environment in accordance with the OpenAI gym interface that could be used by any researcher or practitioner to explore the application of DRL to assignment problems.

Chapter 3

Methods

This chapter discusses the solution methods and simulation model that are used in this project. First, a mathematical model of the problem is formulated. Second, the design of the environment needed to train the DRL agent is discussed. Third, the simulation model of the inbound operations of DC's of large retailer X is discussed. Fourth, the benchmark and DRL algorithms are discussed. Finally, the policy analysis for the trained DRL agent is considered.

3.1 Problem definition

Based on the background information on the problem given in the introduction in Chapter 1, a formal problem definition can be defined. Considering time horizon $T = t_1, \ldots, t_{|T|}$, there are a set of docks D, a set of staging lanes S, a set of trucks K. At each time t, let $d^t \in 0, 1$ denote the availability of each dock $d \in D$, and $s^t \in 0, 1$ denote the availability of each staging lane $s \in S$. Each dock $d \in D$ at time t is associated with a processing time p_d^t and staging lane a processing time p_s^t. Each truck $k \in K$ is associated with the following parameters: estimated time of arrival w_k, number of pallets n_k, priority $r_k \in 0, 1$, weighted distance between a dock d and its products $u_{k,d}$, waiting cost w_k, transportation cost c_k.

Let K_Q^t be a set of trucks in the queue at time t. The decision variable $x_{j,d}^t \in 0, 1$ specifies at each time step t, which truck $j \in K_Q^t$ should be allocated to dock $d \in D$, such that over time T and all trucks K, the total cost $\sum_{t \in T} \sum_{k \in K} (w_k p_k + c_k u_{k,d})$, where p_k is the process time of truck, which is computed by the difference of its departure time and arrival time. The resource constraints in the system are the capacities of docks and staging lanes.

Based on the problem definition as described above, the problem could be solved exactly. However, in reality some parts of the problem are stochastic. This means the problem cannot be solve exactly a priori. For example, there is uncertainty about the arrival time of trucks. This warrants the use of a technique such as deep reinforcement learning as literature shows it can make sequential decisions under uncertainty.

3.2 Environment

Chapter 2 showed how the Markov Decision Process (MDP) framework is often used to model the environment with which the agent interacts. To recap, the MDP is made up of a state space, an action space, a transition, and reward function. At each time step t the agent gets the current

state $S_t \in \mathcal{S}$ as an input. Based on this state S_t, the agent determines which action $A_t \in \mathcal{A}(s)$ to take. In the next time step, the environment returns a reward signal $R_{t+1} \in \mathcal{R} \in \mathbb{R}$ and the new state S_{t+1}. The goal of the agent is to learn a policy π that maps the states to actions that maximize the expected cumulative sum of rewards.

In this project the simulation of the inbound operations plays a large role. Each time there is a possibility to dock a truck the simulation returns the current state and the reward of the previous action. Subsequently, the agent returns an action based on the new state. Since actions only need to be taken when a truck gets docked, it could be that multiple events are handled in the meantime.

3.2.1 State space design

For this project, the state space is represented by a vector that contains information deemed relevant to the truck to dock assignment by experts. The following information (datatype between brackets) is contained in the state:

- Docks: availability of each dock (0 or 1), estimated departure time of a truck if the dock is occupied (float), if the dock is available the estimated departure time is replaced by a zero
- Staging lanes: availability of each staging lane (0 or 1), estimated time the staging lane is cleared if the staging lane is occupied (float), if the lane is available the estimated time the lane is cleared is replaced with a zero
- Queue: for each truck in the queue the state contains the number of pallets (integer), time in queue (float), priority (0 or 1), weighted distances from the product locations to each dock (float)
- Future arrivals: for each arrival the state contains the number of pallets (integer), expected arrival time (float), priority (0 or 1), weighted distances from the product locations to each dock (float)
- Sum of remaining scheduled arrivals remaining for that day (integer)
- The current time (float)

Next to the availability of docks and staging lanes, the state contains information on the queue and the future arrivals. Because the queue is not always the same length and the future arrivals is not fixed either, the state is not always the same length. As the neural network used to find a distribution over the possible actions requires a fixed input size this problem needs to be handled. Therefore, the number of trucks in represented in the state is fixed to 10 for both the future arrivals and queue. If there are less than 10 trucks in the queue or future event set, the state gets padded with zeros until the correct length is achieved. With this padding in place, the vector representing the state grows linearly with the number of docks. As an example, the length of the state vector is 322 for a scenario with 10 docks. Finally, it should be pointed out that the expected arrival time for future events is based on the actual arrival time (in the simulation). To avoid having precise prior knowledge about the arrival time in the state noise is added to the arrival time. It is assumed the noise can be drawn from a uniform distribution $X \sim U(-0.25, 0.25)$, in which the bounds represent the time in hours.

3.2.2 Action space

The action space describes the actions the agent is allowed to take. To allow the agent to learn a policy that surpasses the current policy it useful to identify where the current policy falls short. Chapter 1 provided background information on this project which explained the currently used

greedy algorithm to dock inbound trucks. In the current greedy policy trucks are assigned on a first-come-first-serve basis to the dock that minimized the weighted distance to the products inside the DC. This greedy algorithm works well when there is enough capacity, however, it falls short when (nearly) all docks are occupied. When all docks are occupied and a truck finished unloading its goods, the next truck in the queue is assigned to the dock that just became available. In this case, the dock that just became available might not be good place to dock because of the distance that needs to be traveled inside the DC. There might be another truck in the queue that is more suitable to dock at this location. In order to keep the action space simple, the agent is trained to choose *which* truck from the queue to dock, after the agent selects that truck from the queue it is assigned greedily with respect to the distance between the docks and the products inside the DC. Although the current greedy approach is not optimal, it works reasonably well in quiet scenarios. Therefore, it is expected that the agent can easily learn to handle quiet moments at the DC as greedy already works reasonably well in this scenario.

On top choosing to dock a truck from the queue an additional action is introduced. This additional action is to *wait* (i.e. not assign any truck from the queue). It is expected that this could be useful in many scenarios. For example, consider the scenario with only one truck in the queue and only one dock available. If it is known that the available place to dock is sub-optimal with respect to the distance that needs to be traveled inside the warehouse, it might be better to wait until a different dock becomes available. Of course, this is dependent on information contained in the state, such as the sum and characteristics of future arrivals. It is important to note that the simulation is adjusted to the waiting action. This means that if there is a queue when all expected arrivals have arrived, the queue is emptied before an episode finishes. To speed up training, the agent is restricted from choosing the wait action when there are no future arrivals expected for that episode.

Since the state only contains information on the first 10 trucks in the queue, the agent can choose one of these trucks to dock. If the queue is longer than 10 trucks the additional trucks are not considered in the decision of the agent. Similar to the state space, the action space needs to be fixed as well. As there are not always 10 trucks in the queue to pick from the action space needs to be constrained, this is done through invalid action masking. Thus, the action space is fixed with 11 discrete actions.

Invalid action masking

There are multiple ways to avoid taking invalid actions. One common approach is to give the agent a negative reward when an invalid action is taken. Another way is to *mask* invalid actions and let the agent sample only from the available actions. Recent work by Huang and Ontañón (2020) shows that invalid action masking generally works much better than providing the agent with a negative reward for choosing an invalid action for the case of a real-time strategy game. Furthermore, their work shows that invalid action masking is critical for the performance of policy gradient algorithms. Based on the work by Huang and Ontañón (2020) this project will also use invalid action masking. For example, with the action space of this project, invalid actions would occur if the agent wants to assign the second truck in the queue when there currently is only one truck in the queue.

3.2.3 Gym interface

Section 2.4.3 introduced two toolkits that are often used to benchmark new algorithms. Arguably the most famous of these is the OpenAI gym toolkit. The environments in OpenAI gym follow a certain interface such that research is reproducible and environments are standardized (Brockman et al., 2016). To make the research of this project reproducible and easy to understand the environment follows the OpenAI gym interface. Thus, if someone is familiar with the

OpenAI gym interface, the environment created for this project can easily be used. Furthermore, many Python packages for Deep Reinforcement Learning such as *stable-baselines* (Hill et al., 2018) are compatible with environments that follow the OpenAI gym interface. Using stable-baselines does not only save time during development, but also during learning, as the package is already optimized for performance as well.

3.2.4 Reward function

The reward function is an essential part of the MDP. The goal of this project is to minimize the total costs of the truck to dock assignment. Therefore, the costs in the docking process are used as a reward for the agent. The costs in the inbound process at the DC are the following:

- Transport costs inside the DC: for transporting pallets from the staging lanes to the buffer locations of the products.
- Waiting costs: for trucks that have to wait before docking.
- Stockout costs: for each truck that carries products that are out of stock inside the DC.

These costs play a role in determining the priority of assigning trucks to staging lanes, especially when there is a queue. Specifically, the waiting costs are of interest as these are not known before arrival.

In existing literature similar cases can be found. Phan and Kim (2015) consider the case of assigning containers to trucks. They include the waiting time in the queue and early and late completion time in the cost function. However, they do not include stock-out costs. Herron and Hawley (1969) point out that it can be hard to estimate the costs of a stock-out on an item level. Next to a dollar penalty for each stock-out per time unit, they propose a combined costs function that includes the fraction of demand that cannot be satisfied. According to the authors, this provides more flexibility than determining costs on an item level. Additionally, Badinelli (1986) proposes that the costs of stock-outs are always subjective, and no cost function can truly capture the underlying costs. Therefore, it seems that the stock-out costs need to be determined in close collaboration with subject matter experts of large retailer X. Similarly, the waiting costs are dependent on contracts between large retailer X and suppliers. As such, these costs have to be determined together with subject matters experts of large retailer X as well.

First, the transport costs can be found by multiplying the number of pallets by the distance and the speed of travel inside the warehouse. Second, the waiting costs are only incurred for trucks that have to wait for more than 2 hours. In reality, there are no direct costs related to the waiting of trucks as these are not paid for by large retailer X. However, a truck can not be left waiting in the queue forever, as this would damage relations with suppliers. Therefore, in consultation with large retailer X, it was decided that costs are incurred after two hours of waiting. Third, the stockout costs could be found by multiplying the waiting time of a priority truck (as these represent the stockouts) and a cost parameter. However, as it is currently unknown how often stockouts occur this is left out of scope for this project. The environment does contain all functionality and logic to support modeling such priority trucks.

At the end of each episode the agent gets a reward that is equal to the negative mean costs per truck during that episode. This episodic reward can be represented by equation 3.2.1.

$$r_t(s,a) = -\frac{\sum_{i \in I} c_p^i + c_w^i}{n} \tag{3.2.1}$$

With c_p^i the transport costs inside the DC for truck i, c_w^i the waiting costs for truck i, and n the number of arrivals. In the case of this project one episode consists of one day (16 hours) of real-time simulation. This means that with an arrival rate of six trucks per hour the agent only gets

a reward after 96 actions (if the agent decides not to wait). The sparsity of this reward could be reduced by introducing an intermediate reward. Still, it is hard to define what constitutes a good action directly after each action is taken. An experiment was done with a small positive intermediate reward if the mean costs per truck go down after an action was taken. However, this experiment did not show better performance to just episodic rewards. In addition, the agent would be rewarded for greedy behavior, which is the baseline algorithm that the agent should outperform. Therefore, it was decided to only have an episodic reward.

3.3 Simulation

This section explains the simulation that is used to model the inbound operations of DC's of large retailer X. This simulation plays an important part in the environment needed to train the DRL agent. First, the reasoning for a discrete-event simulation is explained. Second, the discrete-event simulation and its elements are described. Although the elements of the simulation are described with care, some details are abstracted away for readability. If more details are required one is invited to view the code in the repository associated with this thesis.

Motivation discrete-event simulation

The literature review of Chapter 2 showed that scheduling and assignment problems are often modeled and solved by using a simulation based methodology (Chia and Lin, 2016; Zhou et al., 2018; Azab et al., 2020). In this project, a simulation model could aid in developing and comparing possible solution methods. Therefore, a simulation model of the inbound operations process of the DC's is developed. In literature multiple simulation methods can be identified. One such simulation method is a *discrete-event simulation*. Discrete-event simulations (DES) simulate the dynamics of the system on an event-by-event basis instead of checking at each time-step whether or not an event occurs. A DES focuses on the future events sorted on time with each "arriving" event updating the time of the simulation. Furthermore, Jahangirian et al. (2010) note that DES is convenient for resource utilization and queuing, which is of paramount importance in the modeling of the inbound operations. Although other methods exist, these are often focused on high-level modeling, which is not suitable for the detailed processes of the inbound operations in a DC. Therefore, DES seems to be the best method for this project.

3.3.1 Description discrete-event simulation

The simulation for the inbound operations consist of the following events: dock arrival and departure, staging lane arrival and departure, and delays. The events indicate when a truck arrives or departs at the docks, when the goods of a truck have arrived on the staging lane or when the staging lane is cleared, and when a truck is delayed. Next to events, the simulation consists of entities that carry information needed in the simulation. These entities are the following:

- Truck: truck entities are used as a unique identifier for each arrival. The truck entities contain the following information: the arrival time, number of pallets, priority of the truck, and the weighted Manhattan distance from the docks to the buffer locations of the products inside the DC.
- Dock and staging lane: these two entities keep track of the availability of the docks and staging lanes in the simulation.
- Queue: the queue is used to keep track of waiting trucks and their characteristics, this queue is therefore filled with truck entities. A separate queue is maintained for the staging

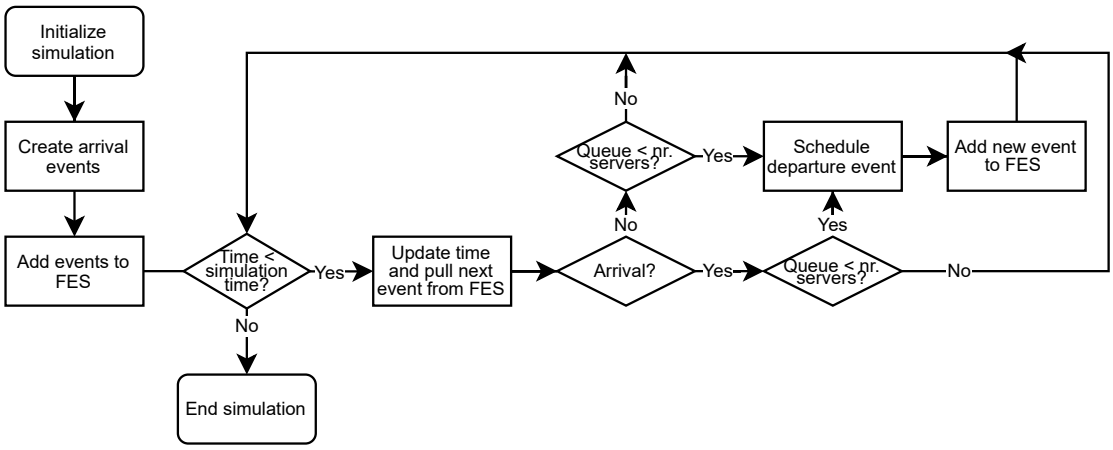

Figure 3.3.1: Simplified flowchart of the discrete-event simulation with the Future Event Set represented by FES.

lanes and the docks as these are distinct queues. A queue at the staging lanes occurs when the staging lane is ready to be cleared but operators are not yet available to clear this lane.

Fig. 3.3.1 shows the process of a simplified version of the discrete-event simulation, some of the logic used in the simulation is not displayed for readability purposes. It is important to note that, the simulation is implemented in Python with special care to ensure minimal dependency on third party packages. The only dependencies outside the Python standard library are SciPy and NumPy. SciPy is used to create and sample from distribution needed for the arrivals. NumPy is used to speed up the simulation with high performance array objects instead of Python lists.

3.3.2 Simulation framework

The previous section gave an introduction to a discrete-event simulation and the assumptions that are made for the simulation of this project. This section dives deeper into the details associated with the simulation for the truck to dock assignment problem. The framework of the simulation gives a high-level overview of the simulation, as can be seen in Pseudocode 1. The framework shows the general steps in the simulation for simulating 16 hours of operations, which will be referred to as a *run* or an *episode*.

Although the simulation is not strictly the same as the environment, the simulation is central to the environment in which the agent learns. Therefore, it is important to explain some basic functionality that is required for the OpenAI Gym interface. The gym interface requires at least the following functions:

- Reset: to reset the environment to an initial state
- Render: to show the current results of the agent
- Step: to simulate one action in the environment

Furthermore, the environment — or in this case the simulation — should return specific information to the agent after it has taken an action. This information consists of four parts: an observation of the state of the environment, a reward, info (in this project this is left empty), and a flag indicating the episode or simulation has ended.

Pseudocode 1: high-level overview of the step function in the simulation

```
Initialize simulation with parameters                                   ▷ See Section 3.3.3
Reset environment                                                       ▷ See Section 3.3.4
while Number of future events > 0 do
    Action ← greedy or agent action selection                           ▷ See Section 3.3.5
    if Action is not wait then
        Handle action                                                   ▷ See Section 3.3.6
    end if
    if previous event is dock departure then
        Schedule dock departure                                         ▷ See Section 3.3.7
    end if
    if previous event is dock arrival, delayed dock arrival, or staging lane departure then
        if dock departure is allowed then
            Schedule dock departure                                     ▷ See Section 3.3.7
            if staging lane operator available then
                Schedule lane departure                                 ▷ See Section 3.3.8
            end if
        end if
    end if
    if Dock queue length > 0 and number of future events < 1 then
        Handle remaining trucks                                         ▷ See Section 3.3.9
    end if
    Set event ← first event from future event set
    time ← time at which current event takes place
    Register current queue length at docks and staging lanes
    Sort queues
    if Event is dock arrival or delayed dock arrival then
        Handle arrival event                                            ▷ See Section 3.3.10
    else if Event is dock departure then
        Handle dock departure event                                     ▷ See Section 3.3.11
    else if Event is staging lane departure then
        Handle staging lane departure event                             ▷ See Section 3.3.12
    end if
end while
Return key performance indicators                                       ▷ See Section 4.4
```

3.3.3 Initialization

The simulation starts with an initialization. For the simulation to work well, several distributions of events are needed as input. The most important is the distribution of the arrivals. Other parameters include the number of docks, the number of dock operators, the number of staging lane operators, the time needed to unload a carrier from a truck, the setup time for unloading a truck (i.e. docking, opening the truck, and administrative tasks), and the simulation time. The distributions used to initialize the simulation can be any distribution that is supported by the scientific package SciPy (Virtanen et al., 2020b). The standard parameter settings for the simulation are where possible derived from data provided by large retailer X. The derivation of these parameters is discussed in the next chapter. To sum up, the following parameters are needed to initialize the simulation:

- Number of docks
- Distances between docks and products
- Number of pallets and products per trucks
- Simulation time
- Distribution of arrivals
- Number of operators at the docks
- Number of operators at the staging lanes
- Time to unload one carrier from a truck
- Setup time for unloading a truck
- Travel speed inside the warehouse

Additionally, there are parameters associated with the waiting costs and the costs of traveling inside the DC. These costs are used to calculate the key performance indicators needed to evaluate the methods. Furthermore, these costs serve an important role in training as they make up the reward of the DRL agent during training. After the parameters are passed to the initialization, the data on the distributions of the number of pallets per trucks and the distances between docks and products are loaded. Next, the Future Event Set (FES) is filled with the arrivals that are expected for that simulation run. For each expected arrival a truck entity is generated. The arrival time of each truck is determined by sampling from the distribution of the interarrival times of trucks that is specified in the initialization function. This interarrival time is added to the arrival time of the previous truck that was generated, now the arrival time is known. The number of pallets and products on these pallets are sampled randomly from available data from a DC. The number of generated events is proportional to the simulation time parameter, which means if the simulation time increases, the number of arrivals increase as well. Subsequently, the generated trucks are used to create arrival events with which the FES is filled. Finally, the events in the FES are sorted on arrival time using a heap queue algorithm.

3.3.4 Environment reset

In order to sequentially simulate one day of operations, the simulation is reset after each run. With the reset the queues are emptied, the time is reset to zero, the FES is filled with events, and additional internal parameters are reset. Additionally, the entity that keeps track of the current state of the system is reset. Although not all parts of the reset are required when a greedy

algorithm is used, this is necessary for the DRL agent. For example, the DRL agent relies on the state for training whereas the greedy algorithm does not. Next to that, the reset function is required by the OpenAI Gym interace.

3.3.5 Action selection

For the DRL approach, the agent can pick any of the first ten trucks in the queue to dock next (if available), or wait. In contrast, the greedy approach always chooses to dock the first truck in the queue, which corresponds to a first-come-first-serve (FCFS) policy. For more details on the action space please refer to Section 3.2.2.

3.3.6 Handling actions

Handling actions might seem straightforward but this is actually quite complex. First, a check is done to see if the action was to wait, if so, there is no need to take further action. If the action is is to dock a truck, the action is handled. Since it could be that the agent chooses to wait multiple times it is frequently checked if a departure is allowed at either the docks or the staging lanes, this is reflected in Pseudocode 2, which describes the process of handling actions.

Pseudocode 2: handling actions

```
Truck ← first truck from queue, or truck selected by agent
if Dock and staging lane combination available then
    Dock number ← dock with shortest weighted distance to product location of goods in truck
    Lane position ← left if available, else right
    Dock truck at selected dock
    Set availability staging lane and dock to occupied
    Remove truck from queue
end if
if previous event is dock departure then
    Schedule dock departure                                   ▷ See Section 3.3.7
end if
if previous event is dock arrival, delayed dock arrival, or staging lane departure then
    if dock departure is allowed then
        Schedule dock departure                               ▷ See Section 3.3.7
        if staging lane operator available then
            Schedule lane departure                           ▷ See Section 3.3.8
        end if
    end if
end if
```

3.3.7 Scheduling dock departures

To schedule a dock departure, a dock departure event has to be created and added to the FES. The most important part is determining the time at which the departure event should occur. This is determined by adding the time to empty the truck to the current time in the simulation. The time needed to empty a truck is determined by a setup time and a time needed to unload all carriers in the truck. Pseudocode 3 summarized this process.

Pseudocode 3: scheduling dock departure events

Truck ← Truck that is scheduled to depart
Departure time ← current time + setup time + time to process a carrier times the number of carriers
Create departure event
Add departure event to FES
Register waiting time of departing truck

3.3.8 Scheduling staging lane departures

Pseudocode 4 describes the process of scheduling a staging lane departure. It is important to note that a "Truck" refers to a truck entity in the simulation which contains information on products, not necessarily a real truck. Furthermore, the travel time is based on the weighted distance between product locations and docks which allows for easy computation.

Pseudocode 4: scheduling staging lane departure events

Truck ← Truck that is scheduled to depart
Dock number ← Dock associated to the staging lane at which the departing truck is located
Travel time ← Time needed to travel from dock to storage of product times the number of carriers
Departure time ← current time + travel time
Create departure event
Add departure event to FES
Register travel time
Register waiting time of departing truck

3.3.9 Handling remaining trucks

The agent is allowed to choose the wait action, this means that at some point all trucks could have arrived while there is still a queue. If this is the case wait action is masked such that deadlocks are prevented. The handling of the remaining trucks is similar to the process described in Pseudocode 2 with the wait action always masked. Empirical results show that the agent quickly learns not to postpone the docking of trucks until the end since this will result in sub-optimal decisions with respect to the waiting and transportation costs.

3.3.10 Handling arrivals

Pseudocode 5 shows how arrivals are handled. As mentioned earlier the environment returns four items (an observation of the state of the environment, a reward, info (in this project this is left empty), and a flag indicating the episode or simulation has ended) each time the agent has to take an action. As the events influence the state of the system they can trigger a situation in which the agent has to decide on an action. Specifically, if operators at the docks are available, an arriving truck can be scheduled to depart right after its arrival.

3.3.11 Handling dock departures

The process of handling dock departures can be seen as a reversed version of the way arrivals are handled. Pseudocode 6 shows how dock departures are handled. Please note that the processing of the next currently docked truck is done during the handling of actions, therefore this is not included in the pseudocode for handling dock departures.

Pseudocode 5: handling arrival events

```
Add truck to dock queue
if dock departure allowed then
    if dock and staging lane combination is available then
        State ← current state of environment
        Return state, reward, done, info          ▷ Required by the Gym interface
    end if
end if
```

Pseudocode 6: handling dock departure events

```
Set availability of dock where departing truck was docked
if dock departure allowed then
    if dock and staging lane combination is available then
        State ← current state of environment
        Return state, reward, done, info          ▷ Required by the Gym interface
    end if
end if
```

3.3.12 Handling staging lane departures

Although one might suspect the staging lane departures to be similar to the dock departures, it is slightly different. The staging lane departures are not departures of trucks, but the final "departure" of the last goods on the staging lane. Pseudocode 7 shows how staging lane departures are handled. A departure at the staging lanes can trigger an opportunity to dock a new truck since a new dock and staging lane combination can become available. If this is the case the simulation returns relevant information to the agent such that it can decide on its next action.

Pseudocode 7: handling staging lane departure events

```
Dock number ← Dock associated to the staging lane that is cleared
Set availability of staging lane where products were located
if a staging lane is full and an operator is available then
    Next staging lane ← Next staging lane in queue that is not currently being cleared
    Schedule staging lane departure                ▷ See Section 3.3.8
end if
if dock departure allowed then
    if dock and staging lane combination is available then
        State ← current state of environment
        Return state, reward, done, info          ▷ Required by the Gym interface
    end if
end if
```

3.3.13 Special events

The basic discrete-event simulation only contains arrival and departure events. However, in reality there are other events that impact the inbound operations. For example, trucks that are scheduled to arrive at a certain time might be delayed due to traffic. A delay event can be introduced that occurs according to a certain distribution. This delay event reschedules one expected arrival in the FES by changing its arrival time.

Another special event that can occur is a stock out inside the warehouse. If a stock out occurs and a scheduled outbound truck needs this product before it can supply a supermarket, the out-

bound truck could wait before it can deliver its goods. In practice the trucks leave to the store without these goods, which is also undesirable. Similarly, if the employees inside the supermarket have to wait for a delivery truck before the shelves can be stocked, extra costs are incurred. These stock outs are modeled by adding a priority parameter to each truck entity. If the truck has priority the extra costs are incurred on top of regular waiting costs. As mentioned earlier, priority trucks are not considered in this thesis project, however, these events are part of the simulation and environment should they be needed.

3.4 Benchmark algorithms

Heuristics are used to set a baseline performance for assigning trucks to docks. The DRL solution can be compared to those heuristics to see its relative performance. The current policy of large retailer X for assigning trucks to docks is close to a greedy policy, which is discussed in this section.

3.4.1 Greedy algorithm

A greedy algorithm is an algorithm that achieves a locally optimal solution. In the case of this project, a greedy algorithm can be used to assign truck to docks. The algorithm chooses to dock the next truck in the queue at the dock that minimizes the distance between the dock and the location of the products in the warehouse. This process is described in Pseudocode 8.

Pseudocode 8: Greedy algorithm

```
Truck ← Truck that is selected by agent or first truck in the queue
Available docks ← Docks that are currently available
for each dock in Available docks do
    Find distance between dock and product locations inside DC
end for
Dock number ← Available dock with lowest distance to product locations
```

This solution works reasonably well since the transportation costs inside the warehouse are locally minimized. However, as discussed in Section 1.2 the greedy algorithm falls short when capacity is dire. Specifically in these scenarios it is expected that a learning algorithm can outperform a greedy algorithm. It should be noted that the greedy algorithm does not have the option to wait, as it always chooses a locally optimal solution. Furthermore, it is assumed that a greedy policy is close to the actual policy that large retailer X uses to assign truck to docks. However, in reality there might be a slightly different ad-hoc prioritization.

3.4.2 Random policy

Next to the greedy approach and the DRL solution, a random policy is considered. In this policy a random truck is chosen from the trucks in the queue to be docked next. Subsequently, the selected truck is assigned to a dock greedily with respect to the distance between the docks and the weighted product locations in the DC. This policy could be used to determine the effectiveness of the DRL approach compared to random choosing a truck from the queue. Results showed this approach on average performs three times worse in terms of costs, therefore, the the random policy was excluded from further experiments.

3.5 Deep Reinforcement Learning algorithm

This section discusses the algorithm used to train the DRL agent and its optimizations. Furthermore, hyperparameter tuning of the proposed DRL algorithm is discussed. For more details on (deep) reinforcement learning, please refer to Section 2.3 and Section 2.4 of the literature review.

3.5.1 Proximal Policy Optimization

Proximal Policy Optimization — or PPO — is used to train the Deep Reinforcement Learning agent. As discussed in section 2.4 this model has been used extensively in literature. Although PPO is much less sample efficient compared to Q-learning methods it is significantly faster in terms of wall-time. Additionally, PPO uses a clipped surrogate objective which makes training relatively stable compared to other algorithms. Recently, the Phasic Policy Gradient (PPG) algorithm was introduced by Cobbe et al. (2021), which was discussed in the literature review in Section 2.3.1. PPG is more sample efficient compared to PPO. This is done by separating the training of the value function and the policy. However, this algorithm introduces additional parameters that need to be tuned and it is much less prevalent in literature, therefore, PPO is preferred for this project.

PPO is available in many Deep Reinforcement Learning packages such as stable-baslines, rllib, and Tensorforce. Using an implementation from an existing package saves time on implementation details and wall-time as this code is already optimized. For example, the PPO implementation of stable-baselines3 supports multi-processing. Additionally, this contributes to the flexibility of the implementation as with stable-baselines3 a different algorithm can be implemented with ease if that would be necessary in the future, for instance, when more advanced algorithms are available. Based on the above arguments, this project uses PPO. More specifically, this project uses the *MaskablePPO*[1] implementation of sb-contrib3, which is an experimental version of PPO from stable-baselines3 that allows invalid action-masking.

3.5.2 Optimizations

Compared to the PPO algorithm as described in the original paper by Schulman et al. (2017), the PPO algorithm used in this project has additional optimizations. These optimizations aid in stability during the training phase of PPO. Work by Andrychowicz et al. (2020) showed that these optimizations are indeed effective. The optimizations are the following:

- Mini-batch updates: in the original PPO paper the policy gets updated using an complete batch of data, with mini-batch the updates are done in a step-by-step manner.
- Normalizing Advantage: for each mini-batch, after the generalized advantage estimation (Schulman et al., 2015b), the advantages are normalized by subtracting their mean and by dividing by its standard deviation.
- Normalizing Observations: similar to the advantages the observations are normalized. This is done by keeping a running mean and standard deviation. The running mean is subtracted for each observation, subsequently this observation is divided by the running variance. This leads to an observation with a mean of approximately zero and a standard deviation of one. Additionally, the original observations are clipped to $[-10, 10]$.
- Rewards scaling: during training the rewards are scaled. This is done by by dividing the rewards by the variance of the discounted returns. Subsequently, the rewards are clipped to a range of $[-10, 10]$.

[1] Link to code repository of MaskablePPO implementation used for this project

- Global gradient clipping: the gradients of the neural network are clipped before feeding it to the Adam optimizer. This way, the global L2 norm does not exceed 0.5.

3.5.3 Hyperparameter tuning

There are many parameters that can be set in the PPO algorithm. However, as Andrychowicz et al. (2020) pointed out not all parameters have a high influence on the performance of policy gradient methods. To get an insight which parameters are the most important for this particular environment a test is done with *Optuna* (Akiba et al., 2019). Optuna is a hyperparameter optimization framework that can be used to find the optimal hyperparameters for various algorithms, including DRL algorithms. Compared to manual tuning hyperparameters, random search, or grid search, Optuna is significantly faster in finding optimal parameters. Random and grid searches waste valuable time due to continuing evaluating unpromising parts of the search space whereas Optuna uses previous evaluations to further explore promising parts of the search space.

For this thesis, Optuna is used with a Tree-structures Parzen Estimator (TPE) sampler. During each trial in which hyperparameters are tested, TPE first fits a Gaussian Mixture Model (GMM) on the set of most promising hyperparameters and second fits a GMM on the remaining parameters. Next, a parameter value is chosen that maximizes the ratio between the two GMM's. Since it is known that certain parameters for neural networks can have similar effects (see for example Smith et al. (2017)) a multivariate version of TPE is used. This version can account for the depedency of certain hyperparameters.

3.5.4 Policy analysis

A policy analysis can be used to get insight into the policy learned by the agent. In this project, the policy analysis serves two purposes. The first purpose is to learn what information in the state is important for the agent to decide on taking a certain action. The second purpose is to extract heuristic rules from the policy of the agent, these rules can give further insight into the policy and allow for easier adaptation of the solution in practice.

3.5.5 Feature importance

The literature review on DRL interpretation methods in Section 2.5 discussed various ways to determine which features are important in the decision-making process of the trained agent. To determine which features of the state were important for the agent to choose a certain action a post-hoc local method can be used, such as a tree-based model. Although these methods are inherently unable to capture the same understanding as the trained agent, they can provide insight into what features are locally correlated with the action.

Recent work by Liessner et al. (2021) shows how SHAP values can be used to interpret the decisions made by an RL agent. The SHAP values can be used to identify which features of a state have positive or negative effect on the action selected by the agent as well as which features have little or no effect. Furthermore, the SHAP values can be used to create insight in the global feature importance. In contrast with tree-based methods, SHAP can use the actual predictions of the agent to generate explanations, which makes is much more reliable. It seems that SHAP is able to fulfill the requirements for the first purpose of the policy analysis; creating insight into what information in the state is important in the decision making process of the agent.

3.5.6 Extracting heuristic rules

In recent years a lot of research showed how tree-based models can be used to explain the decisions of RL agents (Liu et al., 2019; Coppens et al., 2019; Bewley and Lawry, 2020). This research mainly focuses on explaining how features correlate with actions taken by the agent. Although tree-based models cannot be used to capture the true decision-making process of an RL agent they can be used to create heuristic rules that approximate the behavior of the agent. Specifically, decision trees can be used since these are inherently interpretable models from which the decision rules can be extracted.

3.6 Reducing state space

The state space for this project is manually constructed in consultation with subject matter experts. However, as Section 2.6 described, some elements of the RL pipeline could be automated. Previous research showed it might be possible to reduce the state space while maintaining similar performance. This could alleviate problems concerning the curse of dimensionality. Furthermore, Refaei Afshar et al. (2020) showed decreasing the state space by aggregating features can speed up training.

In this project SHAP values are used to determine which features have a large influence on the agent's decision-making. Based on the SHAP values, features can be excluded from the state space if they are not important in the decision-making process. This approach is novel and thus this project could make a contribution to the field of AutoRL by linking evaluations of SHAP values to state space reduction.

Chapter 4

Experimental setup

This chapter introduces the experiments of this project. The goal of these experiments is to derive insight into the current approach of assigning trucks to docks and compare this with a DRL approach. First, the simulation parameters are discussed. Second, different simulation scenarios used to evaluate the methods are explained. Third, the DRL training setup is discussed. Fourth, the objectives and performance metrics used to evaluate different methods are discussed. Finally, the setup for the policy analysis is explained. The results of all experiments are discussed in Chapter 5.

4.1 Simulation parameters

In the previous chapter Section 3.3.3 introduced the parameters of the simulation that can be set by the user. This section explains how the parameters for the simulation were derived from data (if possible), or how they were estimated. The data used to derive the parameters is based on one week (01-11-2021 to 07-11-2021) of data collected from one DC of large retailer X unless specified otherwise. First, the number of docks and the simulation time are discussed. Second, the arrival rate is derived from actual data. Third, the number of operators at both the docks and staging lanes is discussed. Fourth, the time needed to unload a truck is derived from actual data. Fifth and last, the speed of transport inside the DC is discussed.

4.1.1 Number of docks and simulation time

The number of docks and the simulation time play an important role in the simulation. The number of docks has a large influence on the waiting costs that are incurred for trucks waiting to dock. The simulation time has an influence on the density of the reward for the DRL agent. With the current episodic reward, if the number of arrivals per hour remains constant and if the simulation time increases, the rewards become sparser. The sparseness of the rewards has an impact on the complexity of learning a good policy.

The number of docks used for the inbound process is not fixed. As there is not exact data available of what docks are available for which process at what time, this has to be estimated. In collaboration with large retailer X, it was decided that 10 inbound docks would be close to simulating the real DC.

The simulation time is set to 16 hours. Most DC's of large retailer X operate 24/7 with the exception of a weekly maintenance window. During this maintenance window, which occurs

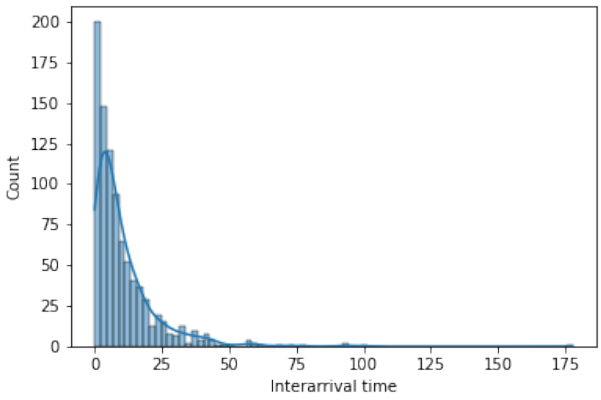

Figure 4.1.1: Interarrival time of trucks for one week of data

every Sunday afternoon, the DC is not operational and as such, trucks cannot be docked or emptied during this time. It is expected that simulating longer than one day would make the rewards for the agent too sparse to learn anything useful. Therefore, it was decided to only simulate for one day and subsequently reset the simulation. Although the operations are more or less continuous, the nights are much quieter compared to daytime. During the night the staging lanes that are still full can be emptied to start the day with nearly all docks and staging lanes empty. Thus, the assumption of resetting the simulation daily should not be too far from the real operations of the DC's.

4.1.2 Arrival rate of trucks

The arrival rate determines how many, and when, trucks arrive at the DC. As this is a key parameter in the simulation this is derived from data provided by large retailer X. In the literature review Section 2.1 and Section 2.2 showed that a Poisson process is often used to model arrivals, as such, it is expected the arrivals of trucks at DC's can also be modeled using a Poisson process. It should be pointed out that trucks related to cross-docking are excluded from the data as this is out of scope for this project. First, an initial check is done to see if the interarrival time of the trucks comes close to a Poisson process. In a Poisson arrival process the interarrival time follows an exponential distribution. Fig. 4.1.1 shows the interarrival time of trucks of one DC for one week of data without any further processing. It can be seen that the shape of the plot has quite long tails. The suspicion arised that outliers are present, perhaps due to trucks arriving during the maintenance period.

As there is a suspicion of outliers the data is transformed into a normal distribution using a box-cox transformation. This transformation is needed to allow the use of outlier tests that assume the data is normally distributed. Fig. 4.1.2 and Fig. 4.1.3 show a histogram and a QQ-plot of the transformed data respectively. Both figures show the data is close to a normal distribution, however, the QQ plot reveals some anomalous observations in the tails. Since the data is approximately normally distributed an outlier test can be performed. Tukey's method is used to remove outliers as it is easy to implement and understand. In short, Tukey's method marks values as potential outliers if they are not in within the range $[-1.5 \cdot IQR, 1.5 \cdot IQR]$ where IQR is the inter quartile range. Tukey's method identifies five potential outliers. This might seem strange at first, however, there are multiple arrivals on Sundays during the maintanance period when the DC is closed for production. A closer inspection of the potential outliers shows that these are trucks that arrive late (in the beginning of the maintanance period) and those that arrive at the

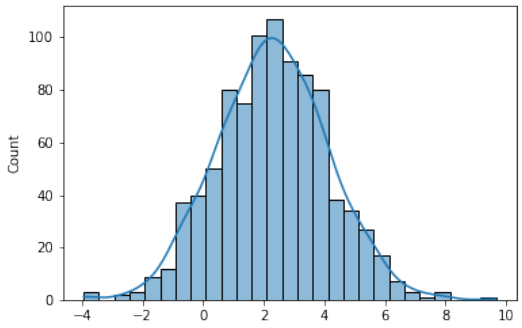

Figure 4.1.2: Histogram of interarrival time after Box Cox transform

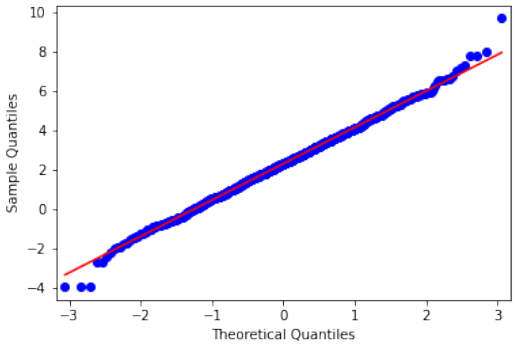

Figure 4.1.3: QQ-plot interarrival time after Box Cox transform

end of the maintenance period. As such, the values identified by Tukey's method as outliers are excluded from the data. Since Tukey's method assumes the data comes from a normal distribution this has to be tested. To test for normality an omnibus test is used that combines a test for kurtosis and skewness. This method circumvents issues regarding ties in the data which would have to be checked when using a Shapiro-Wilk or Anderson-Darling test for normality. The hypothesis H_0: *the data is normally distributed* is tested and a p-value of 0.37 is found with test statistic $k^2 + s^2 = 1.99$. This means the null hypothesis cannot be rejected and thus the assumptions of Tukey's method are met.

After removing the outliers the data is transformed back to its original shape. Fig. 4.1.4 and Fig. 4.1.5 show the distribution of the interarrival time and a sample from an exponential distribution respectively. It can be seen that, although the tail of the actual arrival time is slightly longer, these two plots look similar. A Lilliefors test, which is based on the Kolmogorov-Smirnov test, is done to confirm the belief that the data of interarrival times comes from an exponential distribution. The hypothesis H_0: *the data is exponentially distributed* is tested and a p-value of 0.08 with a test statistic of $KS = 0.035$ is found. Although the evidence is not very strong the null hypothesis that the data comes from an exponential distribution cannot be rejected. This means the arrivals can be modeled with a Poisson process. The mean arrival rate can directly be estimated from the available data and is found to be 6.05 trucks per hour. Therefore, the simulation can sample the interarrival times from an exponential distribution with $\lambda = 6.05$ to simulate a Poisson arrival process.

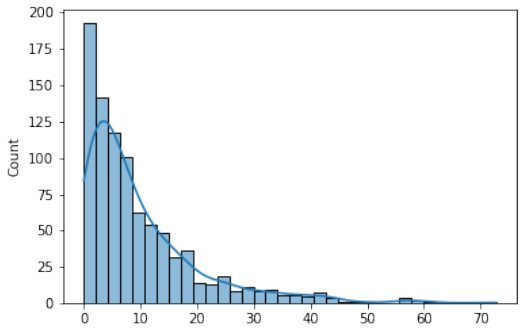

Figure 4.1.4: Interarrival time of trucks for one week of data without outliers

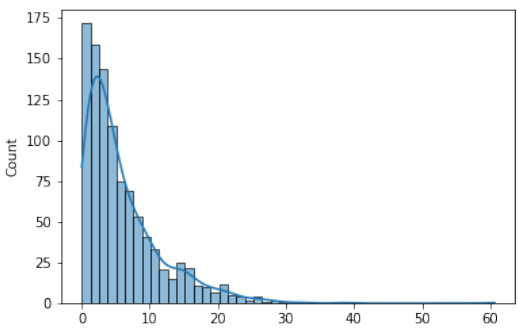

Figure 4.1.5: Sampled data from an exponential distribution $\lambda = 6$

4.1.3 Number of operators

The number of operators working in the DC have a large influence on the total capacity of the DC. If there are more operators, the inbound trucks can be emptied faster and the clearing of the staging lanes takes less time in total. However, there is a limit to the number of operators, for example due to physical restrictions in the number of reach trucks that can operate simultaneously inside the DC.

Unfortunately, there is no detailed data available on the number of operators available for the tasks that are in the scope of this thesis. Operators can have multiple tasks, including tasks related to outbound operations. Therefore, it is hard to exactly determine the number of operators inside the DC. According to experts, queues regularly form at the docks. The speed of emptying a truck and clearing a staging lane *can* be derived from data. Using this data trial runs of the simulation can point out which number of operators is realistic. From the trial runs in the simulation, it seems that the ideal distribution of operators is one operator dedicated to the dock operations (emptying trucks) and three operators dedicated to clearing staging lanes. If the number of operators in these locations decrease, the queue becomes unreasonably large. In contrast, if the number of operators increases, the queue disappears.

Finally, it should be pointed out that operators take breaks and change shifts. This means the operators are not always working. Since the number of operators has to be estimated, the frequency and actual effect of these situations is complex to derive. Therefore, this level of detail is out of the scope of this project.

4.1.4 Unloading trucks

Large retailer X has provided data that can be used to derive the time needed to unload a truck. For each arrival, data is available on the time at which the truck docked and the time at which the truck left the dock. Additionally, there is information available on the number of carriers inside each truck. This way, the processing time can be corrected for the actual load of the truck.

The data shows that there are trucks with very long processing times (over 17 days), which indicates potential outliers, as can be seen in Fig. 4.1.6. A difference in processing time is expected due to breaks and changes of shift. Although it is unclear where the outliers begin, an estimate can be made. It is estimated that for processing times above 5000 seconds (83 minutes) per carrier there is definitely some other mechanism that plays a role. Even breaks or changes in shift do not influence the processing time with this significance. It seems this threshold might still be quite high, however, to prevent excluding valid data the threshold is set at this level.

After the outliers are removed an estimate of the processing time per truck can be made. It is suspected that the processing time consists of a setup time and a processing time per carrier. The setup time is for example the time needed to connect the truck to the actual dock and the time needed to complete administrative tasks. A regression analysis is done to derive the bias (setup time) and the slope (time per carrier) of the data. The regression analysis shows an intercept of 414 seconds and a slope of 7.24 seconds.

4.1.5 Transport inside the DC

The rate of transport inside the DC is directly related to the time required to empty a staging lane. As the distance that needs to be covered decreases or the speed of travel increases, the speed at which the lanes can be emptied increases as well. The speed at which a reach truck can move inside the DC is assumed to be 10 kilometers per hour. The time needed to pick up or put away products is assumed to be 30 seconds per carrier (i.e. in total 60 seconds for picking up and

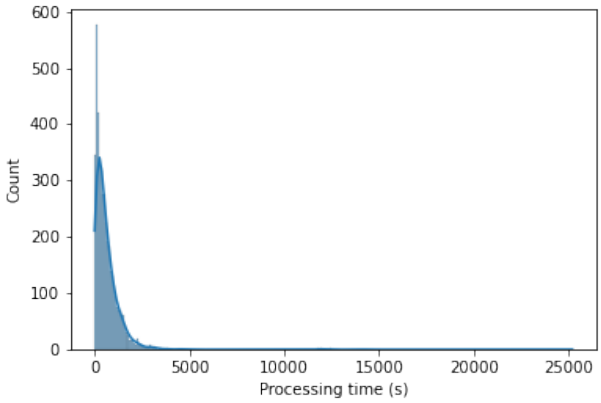

Figure 4.1.6: Countplot of processing time per truck at the docks

putting away).

The remaining factor, the distance between the staging lanes and the products, can be derived from data. large retailer X has provided data that contains the coordinates of the docks and the products inside the DC. Furthermore, coordinates of entry points of the aisles where the products are located are included. Given these coordinates, the distance between the docks and the products can be calculated. The distance function used to calculate the distance is the Manhattan distance, also known as the taxicab distance or city block distance. The Manhattan distance function measures the distance along two axes with right angles. The distance can be found using Eq. 4.1.1, where x and y represent the coordinates of the docks and the product. The final distance is found by adding the distance from the dock to the entry points of the aisle closest to the product and the distance from this entry point to the product location. Although this might not always be the most optimal way to the product locations, it is assumed this route is used by reach truck drivers. For each truck, the distance is weighted by the number of load carriers of each product such that the distance between the docks and products can be given by one vector of with a length equal to the number of docks.

$$d = |x_1 - x_2| + |y_1 - y_2| \tag{4.1.1}$$

Number of pallets per truck

The number of pallets or carriers per truck influences the time needed to unload a truck and the time needed to clear a staging lane. Furthermore, the number of pallets per product is relevant. The number of pallets is related to the type of products, as some products are more frequently delivered. In order to minimize assumptions, the distribution of products per truck is left unchanged. This means trucks with their distribution of product type are directly sampled from the data. Subsequently, these distributions can be matched with a specific number of pallets that is independently sampled from data. This way, the original data set that is made up of over 1000 arrivals at one DC, becomes much larger. The sampling for both the products and the number of pallets is done randomly without replacement.

4.1.6 Delays

Data shows that trucks often do not arrive at the expected time as specified in the advance shipping notice. In general, two cases of delays can be identified. In the first case, trucks arrives

slightly before or after the expected arrival time due to slight variations in breaks or traffic. In the second case, trucks are significantly delayed, for example due to a traffic jam or accident. Data on the arrivals times and expected arrival times shows patterns that align with these two cases. It is complex to find a distribution that can be fitted on this data, therefore the delays are modeled using the two cases. The first case is modeled by adding noise as described in Section 3.2, this includes trucks that arrive slightly early. The second case is modeled by randomly picking a future arrival and delaying it by a minimum of 30 minutes and a maximum of 2 hours. In practice it rarely happens that a truck is more than 15 minutes early, therefore there are no "negative delays". In the simulation the delay is sampled from a uniform distribution $X \sim U(0.5, 2)$, in which the bounds represent the time in hours. These delays are modeled as special events that arrived according to a Poisson process with an interarrival rate of $\lambda = 0.5$.

4.2 Simulation scenarios

The methods described in Chapter 3 are evaluated on different scenarios. The two main scenarios will be referred to as the *large* and *small* case. The difference between the large and small case are the number of docks on the simulation time per episode. The large case is closest to the real operations of the DC, the small case is a simplified version.

Large and small case

In the large case the distribution center is assumed to have 10 docks and 16 hours of operations per day, whereas the small case has 4 docks and 4 hours of simulation time. All other parameters are kept the same. It is important to note that, the docks used in the simulation are evenly spread out over the DC of large retailer X. se two scenarios each serve their own purpose. The large case is closest to the real operations of the DC of large retailer X and can thus be used evaluate the value of a DRL approach over the currently used greedy algorithm. The small case is mainly used to enable faster evaluation and interpretation of the results of the trained agent. Because the easy scenario has less docks and less simulation time the evaluation is much faster ($\approx 4\times$). Furthermore, training the agent is more straightforward as the rewards become much denser. Finally, it is expected that an analysis of the policy of a trained agent is more manageable for the small case as the state space is substantial for the large case.

Other and implicit scenarios

Next to the easy and the large case, other scenarios based on the large case are introduced. These scenarios have slightly different parameter settings, for instance the arrival rate will be slightly lower or higher compared to the training setup. The goal is to use these scenarios to test how well different approaches can handle changes in the environment.

Next to the explicit scenarios the simulation also has implicit scenarios. As discussed in Chapter 1 the greedy assignment of trucks to docks is expected to fall short in scenarios where the capacity of the docks is limited. Because the agent can choose *which* truck to dock next, it is expected that the agent outperforms greedy in these scenarios. As the trucks arrive according to a Poisson process there are simulation runs with fewer and more arrivals. This implies that the agent has to learn to handle both the scenario where capacity is limited as well as the scenario where plenty capacity is available.

It should be pointed out that the difficulty of the problem could be decreased by explicitly creating additional scenarios. These scenarios could, for example, include a warm-up period such that the capacity would already be somewhat limited at the start of the actual simulation. However, in collaboration with large retailer X it was chosen not to pursue this idea to ensure the

simulation stays close to the actual operations.

4.2.1 Number of runs

In order to evaluate the methods in a fair way a certain level of accuracy needs to be guaranteed. The Central Limit Theorem (CLT) can be used to determine the number of runs that are required to achieve a predetermined level of accuracy. In order to determine the number of runs an estimation of the standard deviation σ is required. Given an estimation of σ the number of runs has to satisfy the following inequality:

$$n > (\frac{z_{\alpha/2} \cdot \sigma}{\varepsilon \cdot \bar{x}_0})^2 \tag{4.2.1}$$

The mean costs for processing a truck are deemed the most important performance indicator, therefore, the σ is taken for this value. Performance indicators are discussed in more detail at the end of this chapter. Following a initial run of 500 days of simulation with a trained agent $\sigma = 34.3$ and a mean $\bar{x}_0 = 63.8$ is found for the mean costs. Although the standard deviation is quite high, this makes sense since there is some variation in the number of arrivals. With $\sigma = 34.3$ and a mean $\bar{x}_0 = 63.8$ a total of approximately 455 runs is needed to get a 95% confidence interval.

4.3 Training setup

Section 3.5 previously describe the motivation to use PPO to train the DRL agent. The training starts with initializing the environment. During this initialization the simulation is initialized as described in Section 3.3.3. Subsequently, the algorithm interacts with the environment until the number of training steps has been reached. For a more detailed explanation of the DRL and the PPO algorithm please see Section 2.3.1 and Section 2.4.

Because hyperparameter optimization is computationally expensive Optuna is not used to find the optimal hyperparameters but to give an insight into what parameters are of importance to this problem setting. After running an experiment where all parameters were allowed to be changed Optuna showed that mainly the learning rate and number of steps per update had a high influence on the reward. Together these parameters explained around 85% of the variance in the reward. To still get a near optimal result these two parameters were optimized further. For the large case, the learning rate η was varied on between $5e^{-5}$ and $3e^{-4}$ whereas the number of steps per update was either 2048 or 4096. The model was trained for 2 million time steps for each parameter setting. Additionally, models with the same parameters were evaluated multiple times, due to the randomness during training the same parameters can achieve different results. It seems that some seeds strike a favorable balance between different possible implicit scenarios in the simulation. As the state space is quite large — 322 values for the large case — the training can be unstable. Experiments show that the steps per update and the learning rate influence the training stability and thus the final reward. The model with the best reward used 4096 steps per update with a learning rate of $8e^{-5}$. For the simple case, a similar approach is successful. All experiments are run locally, on CPU, on a machine with an AMD Ryzen 7 4800U with a baseclock of 1.8 GHz and a boost of 4.2 GHz.

The other parameters are the same as the original PPO paper by Schulman et al. (2017). This means a fully connected neural network with two layers of 64 units and a hyperbolic tangent activation function was used for both the value and policy network. The discount factor was set to 0.99 and the clipping was kept at 0.2. Adam was used as the optimizer to update the network weights. Furthermore, the batchsize was set to 64 and the number of epochs per update

to 10. Other parameters introduced in the optimization section in the previous chapter include the clipping parameters, these are set as proposed in Section 3.5.2.

4.4 Performance metrics

The objective of the agent is to minimize the mean total costs per truck, this is also the main performance metric to evaluate all models. As described in Section 3.2.4 the costs of docking a truck consist of the transportation costs c_p and the waiting costs c_w. However, not only the costs are of importance, one of the principal reasons for this Master's Thesis project is the limited capacity at the DC's. Fortunately, part of the cost for docking a truck can be used as a proxy for capacity. If travel costs are minimized, the time needed to clear a staging lane is also minimized. In addition to costs, the following metrics are tracked during evaluation of each model for each episode:

- The number of trucks
- The mean queue length at both the docks and the staging lanes
- The mean waiting time at the docks and staging lanes
- The mean and maximum travel time

As discussed in Section 3.2.4 the costs are directly related to the waiting and travel time. In consultation with large retailer X the costs are fixed to the 25 per hour for operators inside the warehouse, whereas waiting costs are set to 30 per hour. Should the agent perform better compared to the greedy approach, the potential savings could be easily derived given these costs. Since the objective is to lower the costs, it is expected that compared to the current greedy approach the agent is able have a lower queue length and waiting time at both the docks and the staging lanes. Furthermore, the mean and maximum travel time are expected to decrease as well.

4.5 Policy analysis

To get insight into the learned (optimal) policy of the agent 100,000 trajectories (i.e. state and action pairs) of the trained agent with the lowest mean costs per truck are collected. Subsequently, the collected trajectories are used in a supervised learning setting to mimic the behavior of the agent. To check if the model is overfitting, the decision tree classifier is fitted on the training data (state represents the features, actions the labels) and subsequently the fitted model is used to make predictions for the data in the test set. If the scores are similar it can be concluded that the model is valid. Some other details regarding the implementation of the decision tree classifier are explained in the next chapter.

Furthermore, SHAP values are generated with this data in order to provide local and global explanations of the influence of features on the actions of the agent. After collecting the trajectories, the states where the agent is restricted to only one action (the action to wait) are excluded from further processing. Subsequently, the `KernelExplainer` from the SHAP package is used in combination with the trained RL agent to find the SHAP values. To find the SHAP values 100 data samples are used as background.

4.6 Reducing state space

After the policy analysis it should be clear which features influence the agent's decision-making process. The SHAP values are used to determine the feature importances. At this point, it is hard

to determine a threshold for features that should or should not be included in the state space. In this project the features are selected manually. However, this process could be automated by setting a threshold or excluding the top n features that influence the model outcome the least. The reason this is not automated in this project has to do with the limited time available for developing automated solutions.

Chapter 5

Results

This chapter discusses the results of the experiment as presented in Chapter 4. First, the learning behavior of the agent is discussed. Subsequently, the results of the large case are discussed followed by the small case. Next, a sensitivity analysis is performed to see how various factors influence the performance of the agent. After the sensitivity analysis a policy analysis is done, this shows what features were important for the agent in making a decision. Please note that the ticks on the y-axis of figures in this chapter are excluded for confidentiality. Additionally, the results in the tables are scaled for confidentiality.

5.1 Learning behavior

Before diving into the results of the methods it is important to look at the learning behavior of the agents. Two distinct agents were trained, one for the easy scenario and one for the hard scenario. Coincidentally, the hyperparameters for the models with the best rewards during training are the same in both scenarios. It should be noted that for the small case different parameter settings nearly achieved the same results whereas this does not hold true for the large case.

Fig. 5.1.1 and Fig. 5.1.2 show the reward achieved during training. It can be seen that the learning curves for both the hard and small case seem to stabilize after a certain period of training, this indicates the agent reached an optimal solution. Although at first sight the curve for the small case seems less stable compared to the large case, this is mainly due to difference in range of the y-axis. Additionally, it can be observed that the rewards for the small case stabilize much earlier in the training process compared to the large case. This can be explained by the denser rewards for the small case, which allows the agent to learn faster. Although it has not been tested, it is expected that the masking of invalid actions benefited the agent significantly, both in terms of speed and stability during training.

Empirical results show that learning a good policy is much more complex in the hard scenario. Not only did the agent seem to be more sensitive to hyperparameter settings, the seed of the environment also had a great impact on training. It is speculated that this is due the implicit scenarios in the simulation which the seed influences. It is suspected that some order of scenarios during training can greatly influence the agents learning behavior, in both a good and bad way. The best model for the large case was found after evaluating on five different seeds whereas for the small case only three seeds were tested. Please note that, although different seeds were tested during training, the models were all evaluated on different seeds to avoid overfitting on specific patterns.

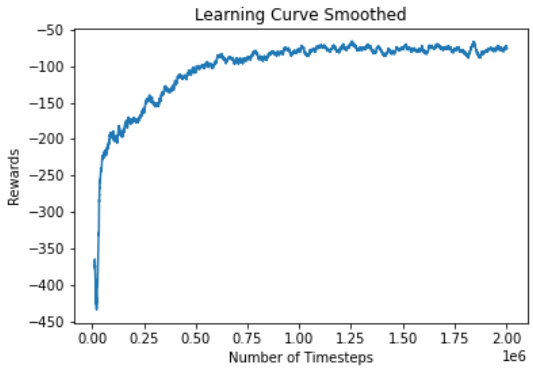

Figure 5.1.1: Smoothed learning curve (large case)

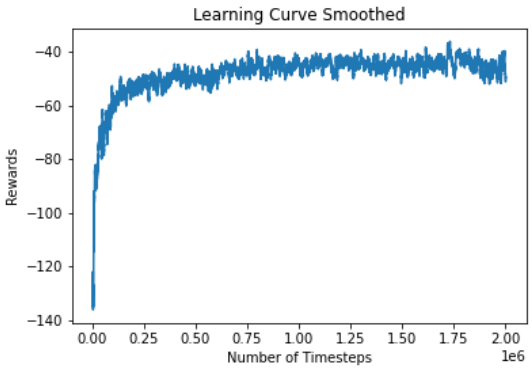

Figure 5.1.2: Smoothed learning curve (small case)

In addition to the learning curve the distribution of actions can provide information about the behavior of the agent. It is expected that the agent chooses different actions compared to the greedy approach, for example, the wait actions. Fig. 5.1.3 shows the action distribution of the agent over 50,000 interactions with the environment for the large case. The actions zero through nine correspond to the positions of the trucks in the queue, with zero being the first truck in the queue and nine the truck in the 10th position in the queue. The wait action is always action 10. As can be observed, the agent chooses to wait (action 10) most frequently. However, in some scenarios the only option the agent has is to wait, thus the distribution might be skewed. If the actions to wait are omitted when this was the only valid action the distribution of 49,813 interaction remain. The figure shows the wait action is still the most chosen action. This is because the simulation always returns the state when there is a truck in the queue and there is a dock available. So once the agent decides to wait, the agent gets the opportunity to dock much more frequently compared to a scenario where waiting is not possible. Furthermore, it is suspected the agent learns that waiting does not have any costs initially and therefore chooses to wait fairly often. Additionally, it can be observed that after the wait action the agent chooses the first truck in queue the most often (12457 times), the second truck in queue is chosen 3608 times. The third truck in queue is chosen only 528 times and the fourth (4 times), fifth (1 times), sixth truck (1 times), and seventh truck (7) are chosen even less frequently. For action four, five, six, and seven the frequency is so low that it is not visible in Fig. 5.1.3. This makes sense since these actions are invalid during most time steps.

From the distribution of actions it follows that part of the state space is largely unused by the agent. As the agent never chooses, or never has the option to choose, certain actions, it seems that this information would be redundant. A similar reasoning can be made for the future events. Empirical results show that indeed, the agent does not rely on the information about future arrivals as much. This implies that it might be possible to achieve similar results with a significantly smaller state space. A smaller version of the state space is tested for the policy analysis.

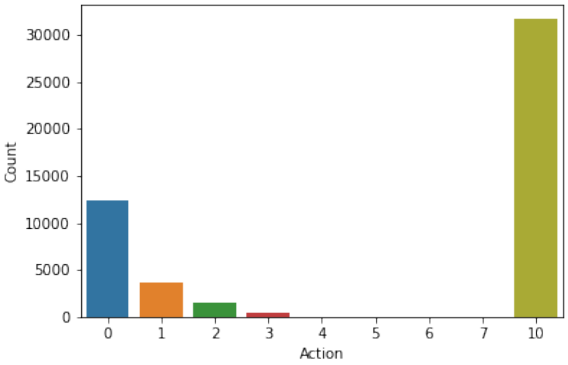

Figure 5.1.3: Action distribution trained agent (large case) for 50,000 interactions

5.2 Large case

For the large case the greedy approach is compared with a trained DRL agent. Both methods are evaluated on the same seeds to ensure the performance comparison is fair. As discussed in Section 4.2 the evaluation should be done for at least 455 episodes to get a 95% CI. For the sake of simplicity this is rounded up to 500 episodes. Please not that the ticks on the y-axis are removed in this section due to confidentiality.

First, the main performance indicator *mean costs per truck* is compared. To recap, these costs are composed of the transportation costs inside the DC and the waiting costs for trucks that cannot dock upon arrival. Compared to the greedy method the agent reduces the costs with 26.9% on average. Fig. B1 shows a lineplot of the mean costs per truck over 500 episodes for the greedy approach and the trained agent. It can be seen that for nearly every episode the mean cost per truck is lower when the agent is used. Further analysis shows that in cases where greedy is better than the agent, it is on average only 2% better. However, over 500 episodes the agent shows a significant improvement in mean costs per truck compared to the greedy method. Next to decreasing the costs, the variance is decreased significantly as can be seen in the boxplot in Fig. B2 which shows much shorter whiskers for the agent compared to the greedy method.

The overall costs can be broken down into transportation and waiting costs. Fig. B3 and Fig. B4 respectively show the total transportation and waiting cost per episode in a boxplot. The waiting costs can be lower for the agent because the transportation costs are minimized by the agent which increases the capacity of the DC. Although both costs are lower for the agent, the difference in the transportation costs is more noticeable. Specifically, in Fig. B3 it can be observed that the box of the agent has almost no overlap with the of the greedy method. It is expected this difference is caused by the sub-optimal decisions of the greedy approach when there is a queue. Additionally, the box is much more condensed, indicating less variance in the transportation costs over different episodes.

Table 5.2.1 shows key descriptive statistics for most KPI's for both the trained agent as well as the greedy method. The values in bold indicate the KPI's where greedy performed better than the agent, for all other KPI's the agent is better. First, it can be seen that the mean performance of the agent is better in every aspect except the mean queue length at the docks. This makes sense because the agent has the option to let trucks wait whereas the greedy method never lets a truck wait if there is a possibility to dock. Even though the mean queue length at the docks is higher, the mean waiting time is lower for the agent. This could be due to the agent making smart decisions in what truck to dock next. Another notable difference is the waiting time and the queue length at the staging lanes, the agent clearly outperforms the greedy method for these

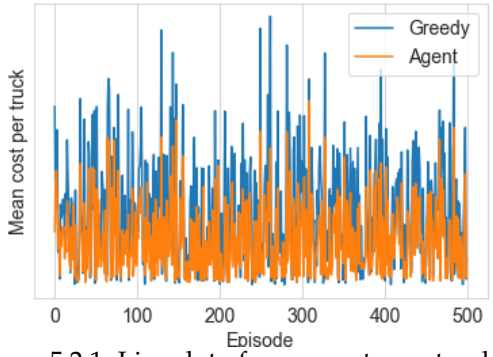

Figure 5.2.1: Lineplot of mean costs per truck

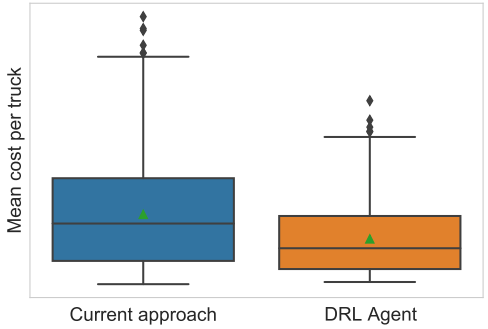

Figure 5.2.2: Boxplot of mean costs per truck

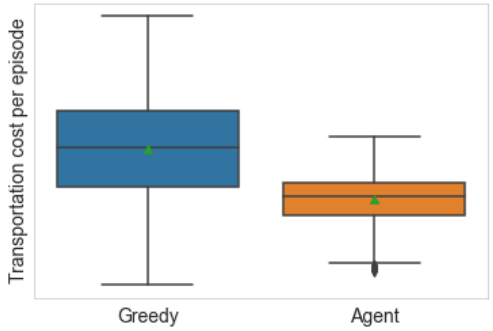

Figure 5.2.3: Boxplot of transportation costs per episode

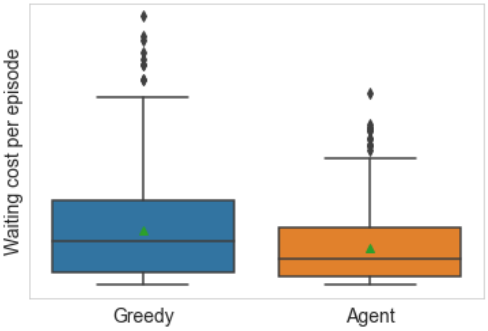

Figure 5.2.4: Boxplot of waiting costs per episode

		$\bar{x}$	σ	Min	Max
Trained Agent	Total cost	285.38	180.80	79.07	1052.52
	Transportation costs	108.15	13.29	72.38	140.23
	Waiting costs	177.19	169.71	0.00	912.25
	Mean queue length dock	2.99	1.80	0.44	9.55
	Mean waiting time dock (hrs)	0.79	0.44	0.11	2.60
	Mean queue length lane	4.66	0.70	2.07	5.90
	Mean waiting time lane (hrs)	1.54	0.572	0.26	3.34
Greedy	Total cost	396.20	272.10	**65.21**	1483.02
	Transportation costs	133.50	269.28	**65.21**	200.60
	Waiting costs	262.72	247.41	0.00	1282.38
	Mean queue length dock	**2.82**	1.89	**0.35**	9.42
	Mean waiting time dock (hrs)	0.92	0.70	**0.009**	3.52
	Mean queue length lane	5.90	1.06	**1.80**	7.61
	Mean waiting time lane (hrs)	2.24	1.10	**0.09**	5.41

Table 5.2.1: Large case: KPI comparison per episode between agent and greedy over 500 episodes

KPI's. This can be explained by the fact that the travel time between the staging lanes and the product locations is much lower for the agent, hence the staging lanes can be cleared in less time. In terms of standard deviation the agent outperforms the greedy method for all KPI's, this means the agent is more consistent for all KPI's. Interestingly, the greedy method has better minimum scores for all KPI's. It is expected that these score were achieved during an episode that did not have many arrivals as in this case the greedy method is expected to work well. Additionally, it can be concluded that the agent has not yet learned an optimal strategy for each scenario.

5.3 Small case

Similar to the large case the greedy approach and the trained DRL agent can be compared. The mean costs per truck are 12.98 for the agent compared to 17.20 for the greedy approach, which is a difference of 24.5%. Although this the difference is slightly less pronounced compared to the large case it is in the same order of magnitude. Overall, in terms of costs, the results show similar differences as for the large case, the detailed plots can be found in the appendix.

Table 5.3.1 shows an overview of the different KPI's for both the greedy approach and the trained agent. It can be observed that the agent outperform the greedy approach in all KPI's except the mean queue length at the docks. Again, this makes sense since the agent sometimes chooses to let a truck wait. In contrast with the large case, the minimum values for the KPI's are also better for the trained agent. It is suspected this is due to the relatively limited capacity of the DC in the easy scenario compared to the hard scenario. This can be explained by the fact that the number of arrivals stay the same for this scenario, even though there are less docks available. Compared to the large case, another remarkable result is the distribution between transportation and waiting costs. For the small case the waiting costs are reduced with over 50.8% whereas this is only 32.5% for the large case. Again, it is suspected that this is caused by the relatively many arrivals for the size of the DC compared to the large case.

		$\bar{x}$	σ	Min	Max
Trained Agent	Total cost	7.90	5.28	1.90	43.76
	Transportation costs	6.28	1.67	1.90	10.88
	Waiting costs	1.61	4.07	0.00	33.16
	Mean queue length dock	0.23	0.13	0.03	0.78
	Mean waiting time dock (hrs)	0.07	0.04	0.02	0.33
	Mean queue length lane	4.23	0.09	0.15	0.58
	Mean waiting time lane (hrs)	0.10	0.05	0.02	0.36
Greedy	Total cost	10.70	7.58	2.05	52.68
	Transportation costs	7.42	2.18	2.05	15.45
	Waiting costs	3.29	5.68	0.00	37.46
	Mean queue length dock	**0.21**	0.15	0.02	0.89
	Mean waiting time dock (hrs)	0.06	0.06	0.00	0.32
	Mean queue length lane	0.48	0.11	0.11	0.67
	Mean waiting time lane (hrs)	0.12	0.08	0.00	0.42

Table 5.3.1: Small case: KPI comparison per episode between agent and greedy over 500 episodes

5.4 Sensitivity analysis

A sensitivity analysis is performed to see how various factors influence the performance of the agent with respect to the greedy approach. Specifically, the parameters that are estimated, or

that are expected to have a high influence on the performance are analyzed. This analysis also gives insight into the generalizability of the model. The sensitivity analysis is performed on the large case as this is the environment closest to the actual operations. All analyses are performed without retraining the agent.

Arrival rate

It is expected that the arrival rate is important for the costs of the truck docking operations of the DC. As the arrival rate increases the waiting time at the docks increases which in turn can lead to higher waiting costs. As the greedy approach is expected to make sub-optimal decisions during scenarios were capacity is insufficient, the difference in costs between the agent and the greedy approach is expected to increase. Additionally, the contrast between the the models is expected to decrease when the arrival rate decreases. First, the mean arrival rate is increased from 6.05 per hour to 8.05 per hour. Second, the mean arrival rate is decreased to 4.05.

The results show the absolute difference between the two approaches increases as the arrival rate increases, however, relatively the difference remains similar. For most other metrics similar results can be observed, the details of these results can be found in Appendix B. It is interesting that the greedy approach no longer achieves the best performance for the lowest costs per episode. This can be explained by the fact that there are less episodes with few arrivals, which is where the greedy approach performs relatively well. Furthermore, the maximum costs are nearly 50% lower for the trained agent compared to the greedy approach. This makes sense, since the agent seems to handle scenarios with many arrivals much better.

When the arrival rate is decreased to a mean of 4.05 arrivals per day the greedy approach outperforms the agent on nearly all KPI's concerning the mean, as can be seen in the results in Appendix B. The agent seems to have a sub-optimal strategy in this scenario, which can be explained by the characteristics of the environment the agent was trained on. Although the state contains information on the remaining arrivals the agent seems to choose to wait in scenarios where this is not beneficial. Additionally, it is expected that a greedy approach works well in scenarios where capacity is abundant. Lastly, the agent still outperforms that greedy approach in terms of standard deviation and minimum values of the KPI's.

Number of operators

Next to the arrival rate the number of operators at the docks and staging lanes are expected to influence the costs of the inbound operations. It is expected that decreasing the number of operators at the staging lanes has a similar effect as increasing the arrival rate. In this analysis the number of operators at the staging lanes are changed as these seem to have the highest influence on the performance. However, when the number of operators is set too high the docks become the limiting factor, not the staging lanes.

The results show that with two operators (instead of three) the agent still outperforms the greedy approach on average, as can be seen in the results in Appendix B. However, the difference between the two methods becomes less prevalent. This could be explained by the increased pressure on the operations due to the decreased number of operators. At some point, the agent can also no longer handle the number of arrivals with the available resources. It is interesting to note that the difference in transportation costs seems to remain similar, in contrast, the difference in waiting costs seems to become smaller.

When the number of operators increases to four (instead of three) the difference between the agent and the greedy approach is apparent. The mean cost per episode of the agent compared to the greedy approach are much lower, as can be seen in the results in Appendix C. The difference

in mean costs increases from 26.9% to 66.6% in favor of the agent. The difference seems to be mostly due to the waiting costs and extreme cases. The standard deviation on the total costs is a factor 10 lower for the trained agent which indicates the greedy approach performs poorly in some scenarios. This is also reflected in the maximum for the KPI's. It can be concluded that the number of operators does have a large impact on the KPI's.

5.5 Policy analysis

Understanding the reasons for the agent's decisions is critical in determining trust in the model, which is important when one intends to act on predictions or when deciding on the deployment of a new model. This section discusses the results of the policy analysis, which give insight into the features in the state that influence the agent's decision-making process. Next to that, the heuristic rules that are extracted from a decision tree classifier are discussed.

5.5.1 Feature importance

As discussed earlier in this chapter it seems to be possible to decrease the state space for the large case. Since the small case and the large case are fairly similar the assumption is made that this also holds true for the small case. This becomes relevant in the policy analysis as the smaller the state space, the easier it might be to understand and interpret the behavior of the agent. Therefore, the agent is retrained on the simple case (i.e. 4 docks, 4 hours of simulation) with a significantly smaller state space of only 45 values instead of 192. The information on trucks in the queue is limited to three and the information on future events is omitted. All other settings and parameters remain unchanged. Results show a similar performance with this small state space compared to the large state space. A mean cost of 12.94 per truck is found for 500 days of simulation compared to 12.98 for the large state space. It can be concluded the additional information of the large state space is not needed to achieve similar performance. After training, 100,000 trajectories (i.e. state and action pairs) of the agent are collected for further analysis.

After collecting the trajectories, the `KernelExplainer` from the SHAP package is used in combination with the trained DRL agent to find the SHAP values. Next, the SHAP values can be plotted to show how each feature influences the decisions of the agent. Fig. 5.5.1 shows the global feature importance for the small case. Alternatively, Fig. C1 in Appendix C shows a summary of influence of the features on the actions of the agent for the small case. The names of the features in the figures are abbreviated for readability purposes. It can be seen the availability features of the docks and lanes are the most important in the agent's decision. Furthermore, the distance feature (starting with "Dist.") can have a large influence on the decision. The first integer in the distance features indicates the number of the dock, whereas "pos. in queue" is an abbreviation of *position in queue*, which indicates the position of the truck in the queue. It seems the agent is able to learn a connection between distance and the reward. Additionally, the waiting time is an important feature in the decision to dock the second truck in the queue. Even though there is only an episodic reward, which might make learning the effect of waiting time on the reward difficult, the agent still appears to be able to learn the relationship between waiting time and the episodic reward.

In general, it seems the agent often chooses a truck that is closest to dock 1 and 2. This focus can be explained by the central location of these docks, which makes these the most interesting places to dock for most trucks. It can also be seen that some other features have little influence on the agent's decision such as the time or the features related to the third truck in the queue. Furthermore, it seems the agent is not influenced by the current time in the simulation. This feature was added as a reference for the estimated departure time features. However, it is suspected that the normalization of the state might influence the training behavior of the agent. Because the nor-

malization uses a rolling mean the relation between the estimated departure time and the time in the simulation is lost. Therefore, it makes sense that the time feature barely has any influence.

Fig. 5.5.2 shows the local explanations for each state and action pair for one episode. The hue in this plot is linked to the SHAP value of that feature at that time step, which helps to indicate the influence of that feature at that time step. As can be seen there is a lot of information in this figure, too much to discuss it all, therefore only interesting results are highlighted.

First, the feature that keeps track of the remaining arrivals during an episode shows a positive influence on the agent's decision (red hue) at the beginning of an episode. This influence means it pushes the agent to a higher output value from the baseline, which is the wait action. At the end of the episode there are less arrivals and the influence is negative, which means it pushes the agent towards docking a truck from the queue. This behavior corresponds to the predicted behavior.

The availability features show that when the values of the features are high (= occupied) the wait action is preferred. Interestingly, dock 4 is only used once during an episode, this could be due to an underlying mechanic in the simulation or due to the location of that dock. If another dock is available that is closed to the product locations the greedy algorithm will always choose this dock. Furthermore, it can be seen the agent is not influenced a lot by the waiting time of the first truck in the queue. In contrast, the waiting time of the second truck in the queue shows high influence on the decision making process. It is unclear why there is a specific focus on the second truck in the queue. It is speculated that this might be due to the increased waiting time that could occur if there is a second truck in the queue compared to just one truck. The SHAP values for the waiting time of the second truck in the queue match with the expected behavior of the agent.

As expected the distance features also show a high influence on the agent's decision-making process. It can be seen the SHAP values are high (red hue) when the distance between the dock and products is large, whereas they are low when the distances are low (blue hue). When the distance between the dock and products is large the agent might prefer to take a different action instead of docking that truck in the queue (depending on availability), this corresponds to the expectations. The features related to the third truck in the queue have little impact on the decisions of the agent. This can be explained by the fact that this spot in the queue is often empty, therefore the agent does not always have consider it in its decision.

This analysis with SHAP values gives insights into the importance of features in the decision making process of a DRL agent. It shows there might be an opportunity to omit certain features from the state space, such as the features related to the third truck in the queue as well as the time feature. It seems there is a possibility for AutoRL methods to include the use of methods such as SHAP to automatically select features to aid in developing a state space.

5.5.2 Extracting heuristic rules

The heuristic rules are extracted from a decision tree classifier fitted on collected trajectories. Similar to the feature importance, the trajectories are for the small case. Before fitting the decision tree classifier it is important to look at the distribution of the labels. Fig. 5.5.3 shows the distribution of the actions, which are used as labels. As expected there is an imbalance in the data. It can be seen that the majority of the actions are the wait actions, followed by the action 0 (dock first truck in the queue) and action 1 (dock second truck in queue).

To check if the model is overfitting, the decision tree classifier is fitted on the training data and subsequently the fitted model is used to make predictions for the data in the test set. If the scores are similar it can be concluded that the model is valid. In the decision tree a trade-off has to be made between interpretability, variance, bias, and error of the model. Fig. 5.5.5 shows

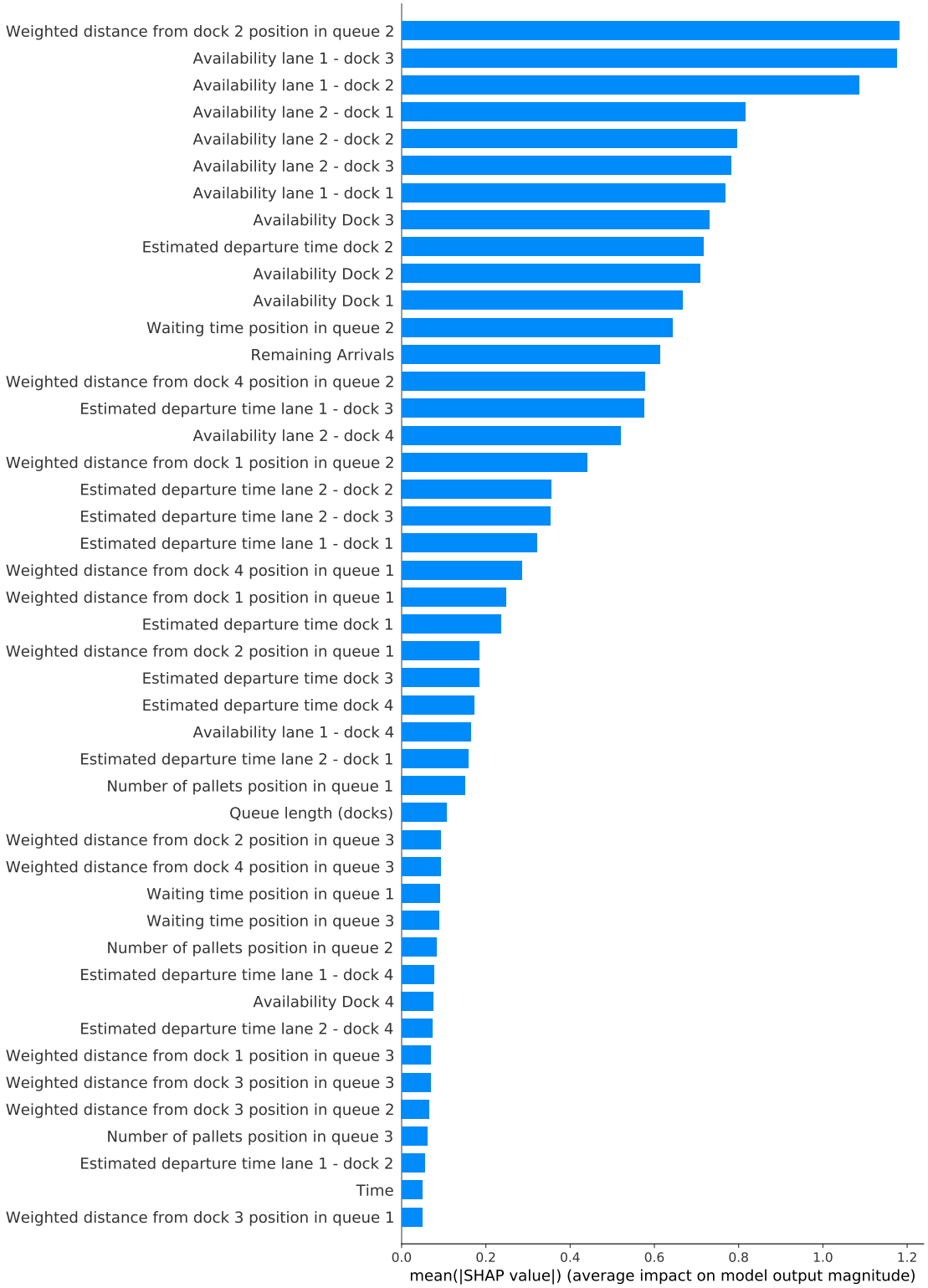

Figure 5.5.1: Global feature importance of the small case

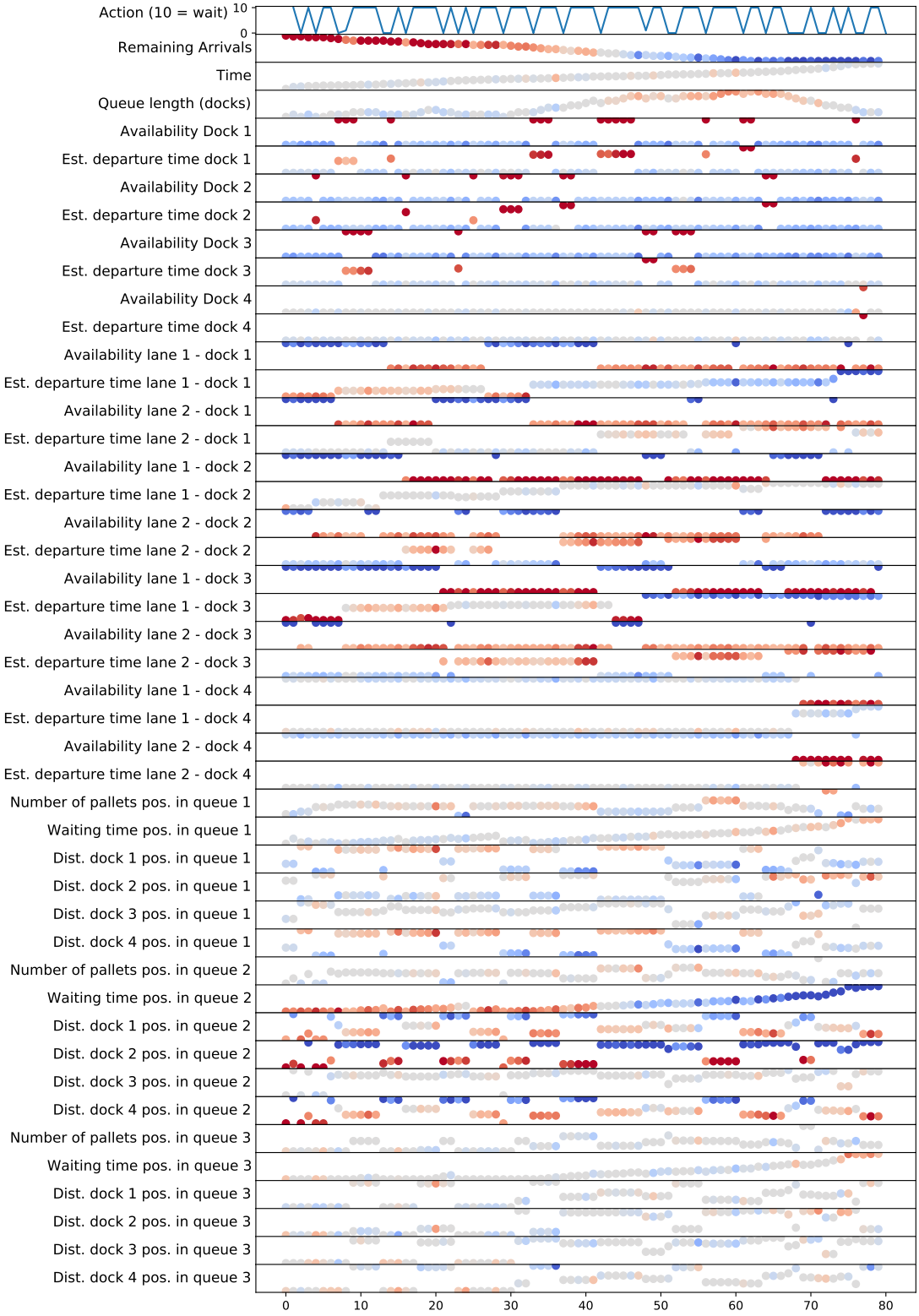

Figure 5.5.2: Local explanations for one episode with the color hue representing the SHAP values

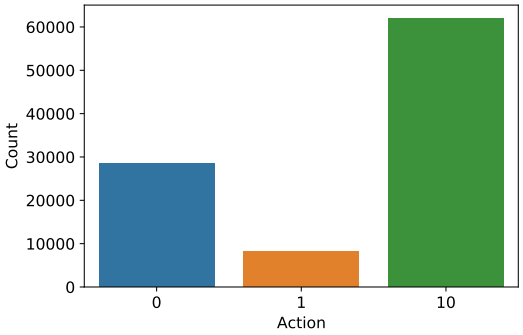

Figure 5.5.3: Action distribution of the input data for the policy analysis

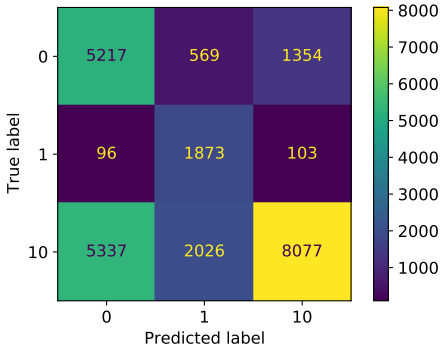

Figure 5.5.4: Confusion matrix of the decision tree classifier on the test set

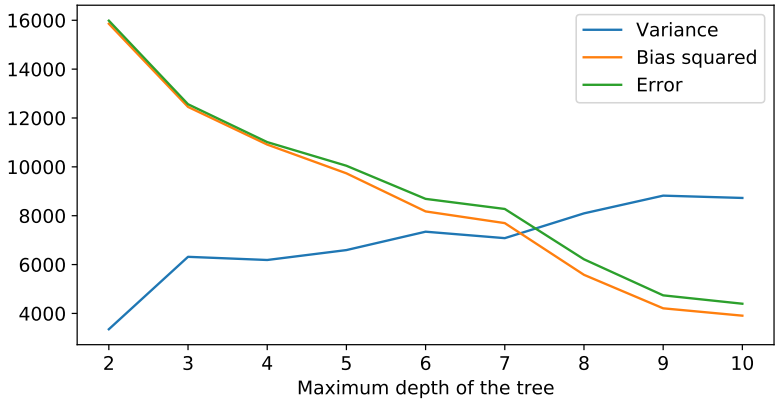

Figure 5.5.5: Bias-Variance trade-off of the decision tree classifier

how the variance, bias, and error change with the maximum depth of the tree. The figure shows that the variance starts the increase at the after a depth of 7 trees. Therefore, it seems that the maximum depth of the tree should be set to 7 to avoid overfitting. However, when a depth of 7 trees is chosen there are 156 nodes, which makes the model hard to interpret. To increase the interpretability of the decision rules the maximum depth of the tree is set to 3, which leads to only 13 nodes in the tree. The effect of the imbalance in the classes is reduced by adjusting the class weights in the classifier, these are inversely proportional to the class frequencies found in the input data. Furthermore, the data is standardized by removing the mean and scaling to unit variance. The scaling is done using the pipeline feature of scikit-learn, which avoids information from the test set leaking into the train set (Pedregosa et al., 2011). A stratified 5-fold split is used for cross-validation.

With the decision tree classifier from the scikit-learn package and the parameters as described above an accuracy of 62% is found for both the train and test set. Thus, it can be concluded that the model is not overfitting. As the maximum depth of the tree is limited, the model is probably underfitting. Fig. 5.5.4 shows the confusion matrix the labels predicted by the model versus the true labels. It can be seen that the label 10 (wait action) has only few false positives compared to the other labels. The model has difficulty predicting the 0 (dock first truck in queue) and 1 (dock second truck in queue) accurately, it is suspected this might be caused by the imbalance in the data set. Furthermore, it can be observed that label 0 and especially label 1 have few false negatives compared to label 10.

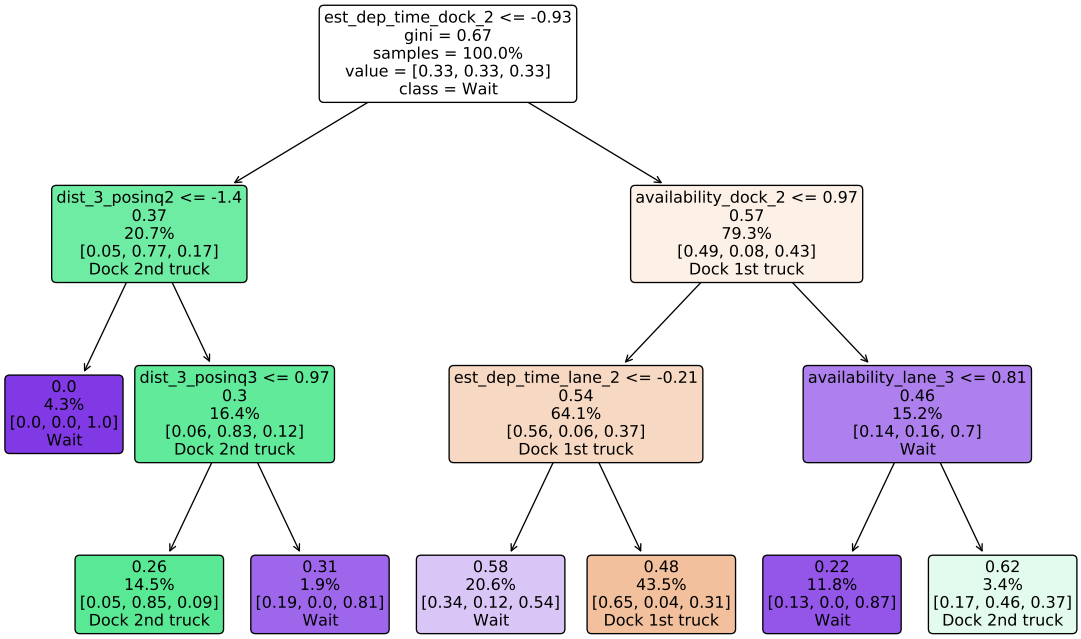

Figure 5.5.6: Visualization of the decision tree used to extract heuristic rules. The root node indicates the labels for the values in each in each nodes, where the label "value" indicates the class distribution in that node. The color of the node indicates the majority class in that node.

Now that the decision tree model has been fitted the tree can be visualized in a figure, as can be seen in Fig. 5.5.6. The colors in this figure indicate the majority class in that node, the "values" indicate the class distributions within a node. It can be seen that the first split is made on the estimated departure time for the truck at dock 2. This is interesting since it is not expected to be such an important feature. However, when the there is no truck at this dock, this value is equal to the current time. Hence, it might be that the first split is actually based on the time feature. It could indeed be that the current time in the simulation indicates what action the agent takes. The simulation always starts with empty docks, therefore the decisions could change depending on the time in the simulation. The second dock might also be the best dock for many trucks due to its central location. Furthermore, the second dock seems to have large influence on the decision to dock the first truck. In the orange colored nodes, the label "dock the first truck in the queue" is the largest. If the depth of the tree would be larger, this leaf node could probably be further split since it now represents 43.5% of the samples whereas it is not the most prevalent class in the data. It can be seen that the decisions before the leaf node are only based on features related to dock 2. In the left branch of the tree it can be seen that the label "dock second truck in queue" is dominant. It seems that the distances for dock 3 are most important in this decision. The decision rules as displayed in the decision tree can be extracted and used to classify new instances. As the model is currently underfitting due to the restriction of the depth of the tree, the model could be retrained if more accuracy is required. However, the interpretability of the model might decrease owing to the increased number of nodes.

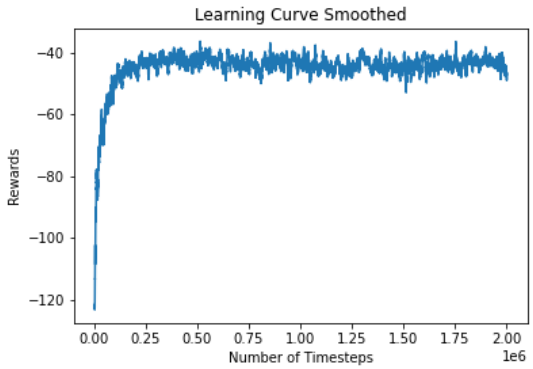

Figure 5.6.1: Smoothed learning curve reduced (35 features)

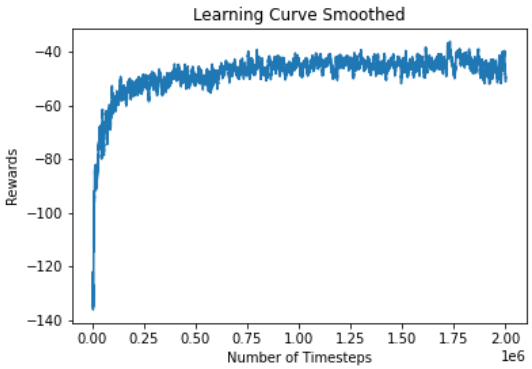

Figure 5.6.2: Smoothed learning curve original (45 features)

5.6 Reducing state space

The SHAP values of the policy analysis show certain features have a trivial influence on the model output. Specifically, Fig. 5.5.1 can be used to identify these features. A distinction can be made between features that are related to certain docks, staging lanes, trucks in the queue, and those that are not. Specifically, the "time" and "queue length" are of low importance and not related to features mentioned above. Therefore, these features are excluded from the state space. Other trivial features are often related to the third truck in the queue. Therefore, these features are excluded as well.

By excluding these features the state space reduces to 35 features compared to the original 45 at the start of the policy analysis. The agent is now retrained with the same parameters as before, on the exact same seed, such that the performance can be compared fairly. Fig. 5.6.1 shows the learning curve of the agent with the reduced state space of 35 features. It can be observed that the performance achieves a reward of -40 in less time steps compared to the agent trained on the original state space of 45 features, as shown in Fig 5.6.5. Evaluating both methods on the mean costs per truck over 500 episodes shows the agent with the reduced state space achieves a mean cost of 12.94 per truck whereas the agent trained on 45 features achieved a mean cost per truck of 12.94. It thus seems an agent can achieve the same performance on a reduced state space. Fig. 5.6.3 shows a barplot of the feature importances for the reduced state space with 35 features. The order of the importances is fairly similar to the original state space. However, it seems the features related to dock 2 become even more relevant.

Based on the feature importances as shown in Fig. 5.6.3, it seems there might be opportunities to reduce the state space even further. Specifically, it seems the agent's decision are not influenced a lot by the the distance between dock 1, 3, and 4 and the product locations in the warehouse. Similarly, the estimated departure time of these docks and their associated staging lanes not important (with some exceptions). It can also be observed from Fig. 5.6.3 that the waiting time does not influence the agent's decision a lot, except for the waiting time for the second truck in the queue. Similar observations can be made in Fig. 5.5.1. An experiment is done where all the features mentioned above are excluded, which decreases the state space to just 17 features. This reduced state space now comprises the availability features for all docks and staging lanes, the distances between dock 2 and the weighted product locations, and the estimated departure times for dock 2 and its staging lanes. After retraining the agent with the same parameters and seed as before a learning curve can be plotted. Fig. 5.6.4 shows the learning curve for the reduced state space with only 17 features. Along side this figure the learning curve or the original state space with 45 features is plotted in Fig. 5.6.5 to make comparing the curve easy. It can be seen

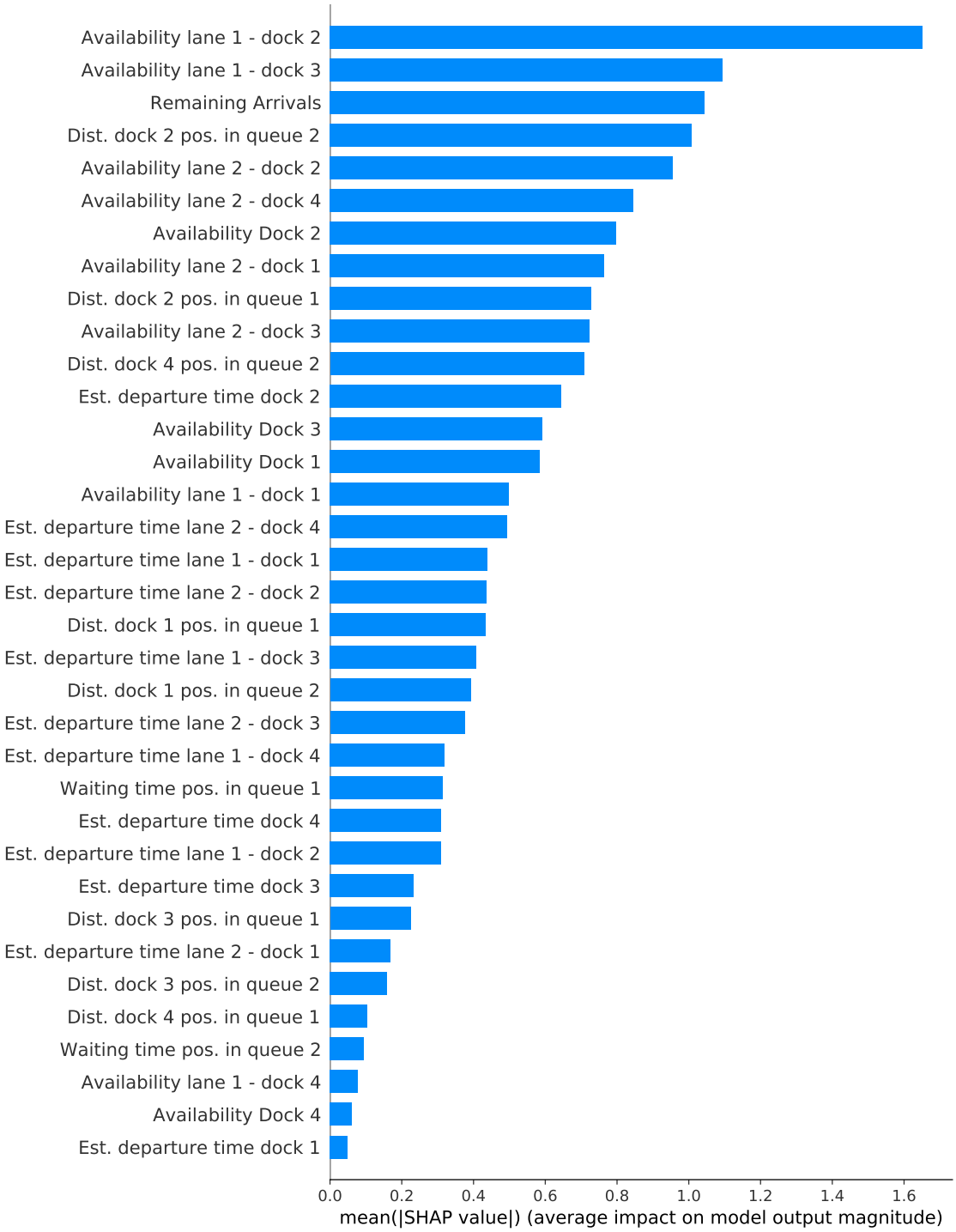

Figure 5.6.3: Global feature importance of the small case reduced state space with 35 features

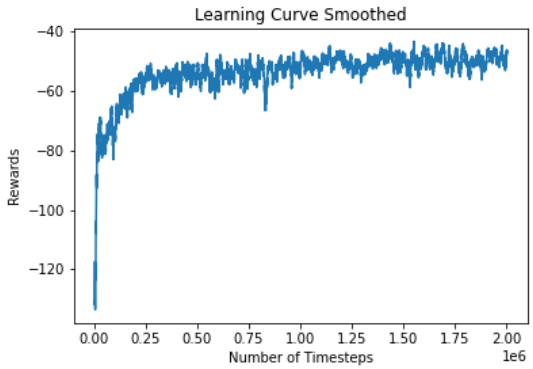

Figure 5.6.4: Smoothed learning curve small case reduced (17 features)

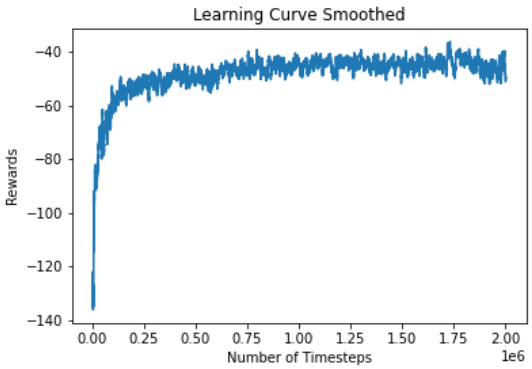

Figure 5.6.5: Smoothed learning curve small case original (45 features)

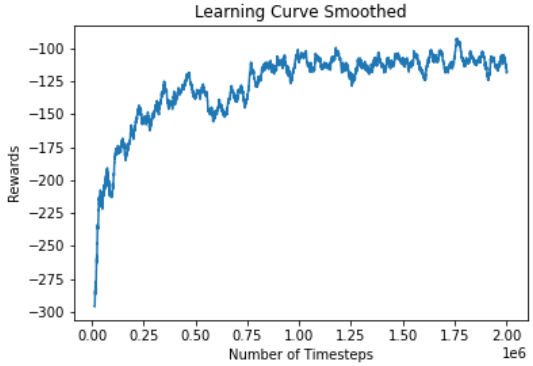

Figure 5.6.6: Smoothed learning curve large case reduced (99 features)

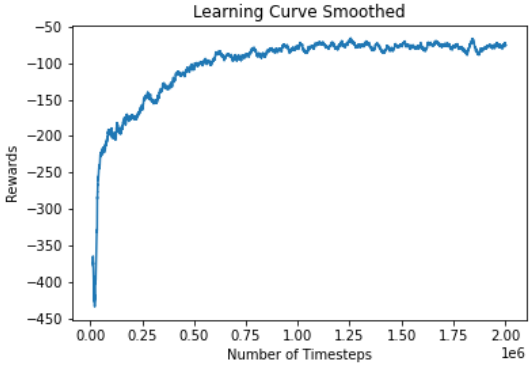

Figure 5.6.7: Smoothed learning curve large case original (322 features)

the curves in Fig. 5.6.4 and Fig. 5.6.5 are similar, however, the curve for the agent trained on the reduced state space of 17 features seems more volatile. Furthermore, it seems the agent trained on 17 features is not able to achieve the same reward during training as the agent trained on 45 features. Evaluating both agents on 500 episodes shows the agent trained on 17 features achieves a mean cost per truck of 21.12, whereas the agent trained on 45 features was able to achieve a mean cost per truck of 12.94. It seems the agent does need additional information to achieve similar performance.

Similar to the small case the state space can be reduced. As shown in Fig. 5.1.3 the majority of the actions are to dock one of the first four trucks in the queue, possibly because the queue is not that long most of the time. Therefore, only the features related to the first three trucks in the queue are kept. This reduces the state space from 322 to 99 features. The agent is retrained with the same parameters on the same seed as the agent with the original features. Fig. 5.6.6 shows the learning curve of the agent trained on the subset of 99 features, this can be compared to the agent trained on the original 322 features in Fig. 5.6.7. It can be seen the agent trained on the reduced state space of 99 features is unable to achieve the same reward as the agent trained on the original state space. Evaluation on 500 episodes confirms this; the mean costs per truck for the agent trained on 99 features are 40.77 compared to 28.12 for the agent trained on the original 322 features. Furthermore, it seems the learning curve is much less smooth for the agent trained on the reduced state space. It seems that additional hyperparameter tuning might be needed as the change in state is quite large from the original to the reduced state space.

Chapter 6

Conclusion

It can be concluded that the master's thesis project filled five distinct gaps in literature. First, the problem context and the stochasticity in the system make the assignment problem in this project is unique. Second, although Deep Reinforcement Learning (DRL) has been applied to similar problems, the application of DRL to this setting, but also to the closely related stochastic sequential assingment problems (SSAPs), is novel. Third, the application of interpretable and explainable DRL methods on practical problems adds to the existing body of literature. Fourth, the thesis project provides an environment in accordance with the OpenAI gym interface that could be used by any researcher or practitioner to explore the application of DRL to assignment problems. Finally, the project uses SHAP values to reduce the state space by a significant amount without compromising on performance of the model. In addition, the study also has practical relevance. The study revealed weaknesses of the current greedy policy used by large retailer X. Additionally, a DRL solution that surpasses the performance of the greedy approach in nearly all KPI's was developed. With the results of this study the main research question can be answered but first the sub questions will be answered.

What is the current process of assigning trucks to docks and how can this be modeled?

Currently, the inbound trucks from suppliers are assigned greedily with respect to the distance between the staging lane and the location of the products inside the DC on a first-come-first-serve (FCFS) basis in most cases. The current greedy policy of assigning trucks to docks does not work well when the capacity at the docks or staging lanes is pressing. When all dock-staging lane combinations are occupied, an inbound truck cannot immediately dock after arriving. Currently, the next truck in the queue is assigned to a dock when this dock becomes available. However, it might be that another truck in the queue is a better fit for that dock with respect to the distance between the dock and the products inside the DC. This forms the basis for an improved policy.

What is Deep Reinforcement Learning and how can it be used to solve the truck to dock assignment problem?

Deep Reinforcement Learning (DRL) combines deep neural networks with reinforcement learning. In contrast with other Machine Learning (ML) methods, RL is concentrated on learning from interactions. The goal of RL is to map situations to actions, or learn an optimal behavior policy for an agent interacting with an environment, to maximize a certain reward signal. As reinforcement learning is centered around sequential decision-making it could be used to optimize the truck to dock assignment problem concerning the inbound operations of large retailer X. In order to do so, an environment is needed with which the DRL agent can be trained. The

environment should resemble the actual process of the inbound operations.

What are methods to interpret and explain the behavior of a Deep Reinforcement Learning agent?

The main distinctions between different interpretation methods are the time of application, the scope, and the model. Post-hoc local methods can be used to explain which features influenced the agent's decision to take a certain action. In contrast, post-hoc global methods can be used to explain which time step was critical in achieving the final episodic reward. To obtain a deep understanding of the agent and to create acceptance of a DRL agent interpretation methods could be useful in the truck docking problem of this project. For example, it could be interesting to use post-hoc local methods to derive features importances for the DRL agent. Deriving these feature importances can be done by using Shapley Additive Explanations (SHAP), which is a unified measure of feature importance. SHAP is based on cooperative game theory and can explain any machine learning model by assigning credit for the model's output to each input feature. Another option are tree-based methods, which are self-interpretable models that can be used to mimic the behavior of a DRL agent. Once the DRL agent is mimicked the interpretation can be derived from the self-interpretable model. Post-hoc global methods could be applied as well, however, it is not expected that this can provide new insights since experts do not expect one specific time step to be crucial for achieving the episodic reward.

What information about the inbound operations is required for a Deep Reinforcement Learning agent to perform well?

The availability features of the docks and lanes are the most important in the agent's decision. Furthermore, the distance features have a large influence on the decisions. It seems the agent is able to learn a connection between distance and the reward. Additionally, the waiting time is an important feature for the second truck in the queue. Even though there is only an episodic reward, which might make learning the effect of waiting time on the reward difficult, the agent still seems to be able to learn a connection between waiting time and the episodic reward.

In general, it seems the agent has a focus on dock 1 and 2. This focus can be explained by the central location of these docks, which makes these the most interesting places to dock for most trucks. Some other features have little influence on the agent's decision, such as the time or the features related to the third truck in the queue. Furthermore, it seems the agent is not influenced by the current time in the simulation.

Is it possible to reduce the state space based on SHAP values without diminishing performance?

The SHAP values of the policy analysis showed that certain features have a trivial influence on the model output. For the small case, ten features were manually selected to be excluded from the state space such that it was reduced to 35 features. Results showed that with less features the model is able to reach an optimal reward in less time steps compared to the original state space of 45 features. A further reduction to 17 features showed a decrease in performance as well as stability during training. Applying the same approach to the large case proved to be more complex. The model trained on the reduced state space for the large case was unable to achieve the same performance as the agent trained on the orignal state space.

How can large retailer X use deep reinforcement learning to optimize the assignment of inbound trucks to docks such that the waiting and transportation costs are minimized?

To use DRL solution an MDP has to be formulated. A discrete-event simulation forms the basis for the environment with which the agent interacts. The agent can choose which truck to dock next, subsequently it is assigned greedily to the dock that minimizes the distance to the weighted product location inside the warehouse.

Compared to the greedy method the DRL agent on average reduces the costs with 26.9% in a simulated environment. Additionally, results showed that the mean performance of the agent is better than the greedy approach in every aspect except the mean queue length at the docks. The sensitivity analysis showed the agent outperforms the greedy approach when the arrival rate is increased. However, if the arrival rate decreases the greedy approach performs marginally better. This is consistent with the belief that the greedy method works well when the DC is not busy.

6.1 Limitations and future research

This study considered Deep Reinforcement Learning as a solution to optimize the truck to dock assignment process of inbound trucks at distribution centers of the large retailer X. Some assumptions were made in order to complete the study within the available time frame. For a situation with static docks the DRL solution surpasses the current greedy approach in a simulated environment. In reality, the docks are more flexible with outbound docks becoming available for inbound operations during some parts of the day. In order for the solution to be practically useful, the dynamic nature of docks needs to be taken into account.

Furthermore, the current setup does not take seasonality into account. It seems seasonality especially has a large effect on the operations of a DC, as holidays such as Easter, Sinterklaas, and Christmas are periods in which sales increase. There are weekly and daily patterns, which are not taken into account in the simulation. For future research it could be interesting to add scenarios that include seasonal patterns such as holidays. Additionally, the agent could be trained specifically on scenarios with few arrivals, or many arrivals. This allows the agent to learn an optimal solution for each scenario, which results show to be hard in the current setup.

Next to comparing the DRL solution with the current approach of large retailer X it could be interesting to compare it to additional benchmarks. For example, solving a problem instance optimally could show how close the DRL solution is to an optimal solution. This could strengthen the proposition to use a DRL solution.

For this study, the operations are assumed to shut down during the night, with a time-window of 16 hours per day during which trucks can arrive. In reality the operations are closer to a continuous operation with less arrivals during the night and a weekly service window. However, if the simulation would simulate a full week the rewards for the RL agent are expected to be too sparse to learn anything meaningful. It could be interesting to find ways to extend the episode length such that it represents one week of real time operations. It is suspected that a more comprehensive reward function could aid in this attempt.

In the current setup the agent can choose *which* truck from the queue to dock, not *where* this truck should be docked. It seems there is an opportunity to expand the action space such that it also includes the decision *where* to dock the chosen truck. Although the complexity of the action space increases, it could further improve the results of the DRL agent.

It seems the simulation model could be improved by including the outbound operations and order picking as this will allow modeling of dynamic docks. There are opportunities to optimize the entire DC operations once this is included in the simulation model. The multi-agent reinforcement learning (MARL) field is rapidly expanding, with a more comprehensive simulation model opportunities arise to use multi-agent reinforcement learning (MARL) to optimize the sub-parts of the DC operations. For example, one agent could focus on the assignment of inbound truck, another agent on the order picking process, and a third agent on the outbound process. For future research, it suggested to look into the potential of MARL for optimizing the DC operations. The environment developed in this study provides a starting point for doing so.

Finally, more research could be done in automating feature selection using SHAP values. This research showed that to some degree features can be manually removed based on SHAP values. Automating this process could further add to the existing body of literature on AutoRL.

6.2 Recommendations

This study has shown how DRL can be used to improve the truck to dock assignment problem in a simulated environment. As the research in the field of DRL is rapidly developing it seems that the future will only hold more opportunities to apply DRL.

This study also showed that there is a lot of dependency between processes in the DC. For example, the algorithm that optimizes the order picking process relies on the outbound dock planning. It would be valuable to consider the business case for modeling the complete operations of the DC in an interactive environment such that the operations in the DC can be optimized simultaneously. Finally, if the decision is made to use a DRL agent it is advised to:

- Combine the DRL agent with existing methods. Perhaps some parts of the problem can be optimized with existing methods, such as the algorithms that are currently used to optimize the order picking process. This would save time during the early stages of developing an environment.
- Make the environments modular. If the environments are made modular it becomes easy to expand the environment such that it contain larger or more complex parts of the problem.
- It is recommended to develop the environments in alignment with the OpenAI gym interface since this currently is the de facto standard. This research showed that state-of-the-art DRL models can be easily implemented when an environment follows this interface.
- Closely follow the development in the field of DRL. The field of DRL is quite new and rapidly developing. Besides the recommendation to follow the OpenAI gym standard, the "best practices" in the field are not very clear, however, this could change in the future.

Bibliography

Adadi, A. and Berrada, M. (2018). Peeking inside the black-box: a survey on explainable artificial intelligence (xai). *IEEE access*, 6:52138–52160.

Afshar, R. R., Zhang, Y., Vanschoren, J., and Kaymak, U. (2022). Automated reinforcement learning: An overview. *arXiv preprint arXiv:2201.05000*.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pages 2623–2631.

Albright, S. C. (1974a). A markov-decision-chain approach to a stochastic assignment problem. *Operations research*, 22(1):61–64.

Albright, S. C. (1974b). Optimal sequential assignments with random arrival times. *Management Science*, 21(1):60–67.

Alharin, A., Doan, T. N., and Sartipi, M. (2020). Reinforcement learning interpretation methods: A survey. *IEEE Access*, 8:171058–171077.

Anderson, C. W. (1989). Learning to control an inverted pendulum using neural networks. *IEEE Control Systems Magazine*, 9(3):31–37.

Andrychowicz, M., Raichuk, A., Stańczyk, P., Orsini, M., Girgin, S., Marinier, R., Hussenot, L., Geist, M., Pietquin, O., Michalski, M., et al. (2020). What matters in on-policy reinforcement learning? a large-scale empirical study. *arXiv preprint arXiv:2006.05990*.

Avis, D. and Devroye, L. (1985). An analysis of a decomposition heuristic for the assignment problem. *Operations Research Letters*, 3(6):279–283.

Azab, A., Karam, A., and Eltawil, A. (2020). A simulation-based optimization approach for external trucks appointment scheduling in container terminals. *International Journal of Modelling and Simulation*, 40(5):321–338.

Badinelli, R. D. (1986). Optimal safety-stock investment through subjective evaluation of stockout costs. *Decision Sciences*, 17(3):312–328.

Balaji, B., Bell-Masterson, J., Bilgin, E., Damianou, A., Garcia, P., Jain, A., Luo, R., Maggiar, A., Narayanaswamy, B., and ORL, C. Y. (2019). Reinforcement learning benchmarks for online stochastic optimization problems. arxiv, 2019. *URL http://arxiv. org/abs*.

Bastani, O., Kim, C., and Bastani, H. (2017). Interpretability via model extraction. *CoRR*, abs/1706.09773.

Baumann, M. and Buning, H. K. (2011). State aggregation by growing neural gas for reinforcement learning in continuous state spaces. In *2011 10th International Conference on Machine Learning and Applications and Workshops*, volume 1, pages 430–435. IEEE.

Beeks, M., Afshar, R. R., Zhang, Y., Dijkman, R., van Dorst, C., and de Looijer, S. (2022). Deep reinforcement learning based solution for a multi-objective online order batching problem. In *Proceedings of the 32nd International Conference on Automated Planning and Scheduling*.

Bewley, T. and Lawry, J. (2020). Tripletree: A versatile interpretable representation of black box agents and their environments. *CoRR, abs/2009.04743*.

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., and Zaremba, W. (2016). Openai gym.

Burkard, R. E. (1979). Travelling salesman and assignment problems: A survey. *Annals of Discrete Mathematics*, 4(C):193–215.

Cals, B., Zhang, Y., Dijkman, R., and van Dorst, C. (2021). Solving the online batching problem using deep reinforcement learning. *Computers & Industrial Engineering*, 156:107221.

Carvalho, D. V., Pereira, E. M., and Cardoso, J. S. (2019). Machine learning interpretability: A survey on methods and metrics. *Electronics*, 8(8):832.

Chia, L. and Lin, W. (2016). Simulation study of patient arrivals and doctors scheduling in a children's emergency department. In *2016 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pages 321–325. IEEE.

Chollet, F. (2015). Keras. https://github.com/fchollet/keras.

Cobbe, K. W., Hilton, J., Klimov, O., and Schulman, J. (2021). Phasic policy gradient. In *International Conference on Machine Learning*, pages 2020–2027. PMLR.

Coppens, Y., Efthymiadis, K., Lenaerts, T., and Nowé, A. (2019). Distilling Deep Reinforcement Learning Policies in Soft Decision Trees. *Proceedings of the IJCAI 2019 Workshop on Explainable Artificial Intelligence*, pages 1–6.

Dantzig, G. B. (1957). Discrete-variable extremum problems. *Operations Research*, 5(2):266–277.

David, I. and Yechiali, U. (1985). A time-dependent stopping problem with application to live organ transplants. *Operations Research*, 33(3):491–504.

Derman, C., Lieberman, G. J., and Ross, S. M. (1975). A stochastic sequential allocation model. *Operations Research*, 23(6):1120–1130.

Derman, C.; Lieberman, G. J. R. S. M. (1972). A sequential stochastic assignment problem. *Management Science vol. 18 iss. 7*, 18.

Dervovic, D., Hassanzadeh, P., Assefa, S., and Reddy, P. (2021). Non-Parametric Stochastic Sequential Assignment With Random Arrival Times. pages 4214–4220.

Dessouky, M. M. and Kijowski, B. A. (1997). Production scheduling of single-stage multi-product batch chemical processes with fixed batch sizes. *IIE transactions*, 29(5):399–408.

Dinic, E. and Kronrod, M. (1969). An algorithm for the solution of the assignment problem. In *Soviet Math. Dokl*, volume 10, pages 1324–1326.

Dutta, A. and Dobe, M. (2016). Health sector reform: Time to introspect. *Journal of Public Health Policy*, 37(3):388–393.

Eberhardt, S. P., Daud, T., Kerns, D., Brown, T. X., and Thakoor, A. (1991). Competitive neural architecture for hardware solution to the assignment problem. *Neural networks*, 4(4):431–442.

Frosst, N. and Hinton, G. E. (2017). Distilling a neural network into a soft decision tree. *CoRR*, abs/1711.09784.

Garland, M. (2021). Amazon rushes to boost fulfillment capacity as it plays 'catch-up' to demand. https://www.supplychaindive.com/news/amazon-earnings-logistics-fulfillment-capacity-ecommerce/604161/. [Online; accessed 06-March-2022].

Grimmett, G. and Stirzaker, D. (2020). *Probability and random processes*. Oxford university press.

Herron, D. P. and Hawley, R. L. (1969). Establishing the optimum inventory size and stocking policy for a warehouse. *A I I E Transactions*, 1(1):75–80.

Hessel, M., Modayil, J., Van Hasselt, H., Schaul, T., Ostrovski, G., Dabney, W., Horgan, D., Piot, B., Azar, M., and Silver, D. (2018). Rainbow: Combining improvements in deep reinforcement learning. In *Thirty-second AAAI conference on artificial intelligence*.

Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y. (2018). Stable baselines. https://github.com/hill-a/stable-baselines.

Hillier, F. S. (2012). *Introduction to operations research*. Tata McGraw-Hill Education.

Huang, S. and Ontañón, S. (2020). A closer look at invalid action masking in policy gradient algorithms. *arXiv preprint arXiv:2006.14171*.

Hubbs, C. D., Perez, H. D., Sarwar, O., Sahinidis, N. V., Grossmann, I. E., and Wassick, J. M. (2020). Or-gym: A reinforcement learning library for operations research problems. *arXiv preprint arXiv:2008.06319*.

Jahangirian, M., Eldabi, T., Naseer, A., Stergioulas, L. K., and Young, T. (2010). Simulation in manufacturing and business: A review. *European Journal of Operational Research*, 203(1):1–13.

Kaizer, J. S. (2013). *Fundamental theory of scientific computer simulation review*. University of Maryland, College Park.

Kaizer, J. S., Heller, A. K., and Oberkampf, W. L. (2015). Scientific computer simulation review. *Reliability Engineering & System Safety*, 138:210–218.

Karakovskiy, S. and Togelius, J. (2012). The mario ai benchmark and competitions. *Computational Intelligence and AI in Games, IEEE Transactions on*, 4:55–67.

Kellner, M. I., Madachy, R. J., and Raffo, D. M. (1999). Software process simulation modeling: why? what? how? *Journal of Systems and Software*, 46(2-3):91–105.

Kennedy, D. (1986). Optimal sequential assignment. *Mathematics of Operations Research*, 11(4):619–626.

Kingman, J. F. C. (1992). *Poisson processes*, volume 3. Clarendon Press.

Kleinrock, L. (1975). Queueing systems. Technical report.

Kong, W., Sivakumar, D., Liaw, C., and Mehta, A. (2019). A new dog learns old tricks: RL finds Classic optimization algorithms. *7th International Conference on Learning Representations, ICLR 2019*, (2009):1–25.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97.

Kurtzberg, J. M. (1962). On approximation methods for the assignment problem. *Journal of the ACM (JACM)*, 9(4):419–439.

Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Goldberg, K., Gonzalez, J., Jordan, M., and Stoica, I. (2018). Rllib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning*, pages 3053–3062. PMLR.

Liessner, R., Dohmen, J., and Wiering, M. A. (2021). Explainable reinforcement learning for longitudinal control. In *ICAART (2)*, pages 874–881.

Lin, L.-J. (1992). *Reinforcement learning for robots using neural networks*. Carnegie Mellon University.

Liu, G., Schulte, O., Zhu, W., and Li, Q. (2019). Toward interpretable deep reinforcement learning with linear model u-trees. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11052 LNAI:414–429.

Lundberg, S. M., Erion, G., Chen, H., DeGrave, A., Prutkin, J. M., Nair, B., Katz, R., Himmelfarb, J., Bansal, N., and Lee, S.-I. (2020). From local explanations to global understanding with explainable ai for trees. *Nature machine intelligence*, 2(1):56–67.

Lundberg, S. M. and Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in neural information processing systems*, 30.

Mao, H., Alizadeh, M., Menache, I., and Kandula, S. (2016). Resource management with deep reinforcement learning. In *Proceedings of the 15th ACM workshop on hot topics in networks*, pages 50–56.

McKinney, W. et al. (2011). pandas: a foundational python library for data analysis and statistics. *Python for high performance and scientific computing*, 14(9):1–9.

McLay, L. A., Jacobson, S. H., and Nikolaev, A. G. (2009). A sequential stochastic passenger screening problem for aviation security. *IIE Transactions*, 41(6):575–591.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. (2015). Human-level control through deep reinforcement learning. *nature*, 518(7540):529–533.

Mourtzis, D., Doukas, M., and Bernidaki, D. (2014). Simulation in manufacturing: Review and challenges. *Procedia CIRP*, 25(C):213–229.

Oliphant, T. E. (2006). *A guide to NumPy*, volume 1. Trelgol Publishing USA.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., et al. (2019). Pytorch: An imperative style, high-performance deep learning library. *arXiv preprint arXiv:1912.01703*.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830.

Pentico, D. W. (2007). Assignment problems: A golden anniversary survey. *European Journal of Operational Research*, 176(2):774–793.

Phan, M.-H. and Kim, K. H. (2015). Negotiating truck arrival times among trucking companies and a container terminal. *Transportation Research Part E: Logistics and Transportation Review*, 75:132–144.

Puiutta, E. and Veith, E. M. (2020). *Explainable Reinforcement Learning: A Survey*, volume 12279 LNCS. Springer International Publishing.

Raffin, A., Hill, A., Ernestus, M., Gleave, A., Kanervisto, A., and Dormann, N. (2019). Stable baselines3. *GitHub repository*.

Refaei Afshar, R., Zhang, Y., Firat, M., and Kaymak, U. (2020). A state aggregation approach for solving knapsack problem with deep reinforcement learning. In Pan, S. J. and Sugiyama, M., editors, *Proceedings of The 12th Asian Conference on Machine Learning*, volume 129 of *Proceedings of Machine Learning Research*, pages 81–96. PMLR.

Ribeiro, M. T., Singh, S., and Guestrin, C. (2016). " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1135–1144.

Roth, A. M., Topin, N., Jamshidi, P., and Veloso, M. (2019). Conservative q-improvement: Reinforcement learning for an interpretable decision-tree policy. *arXiv preprint arXiv:1907.01180*.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *arXiv preprint arXiv:1511.05952*.

Schulman, J., Levine, S., Abbeel, P., Jordan, M., and Moritz, P. (2015a). Trust region policy optimization. In *International conference on machine learning*, pages 1889–1897. PMLR.

Schulman, J., Moritz, P., Levine, S., Jordan, M., and Abbeel, P. (2015b). High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438*.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Siebers, P.-O., Macal, C. M., Garnett, J., Buxton, D., and Pidd, M. (2010). Discrete-event simulation is dead, long live agent-based simulation! *Journal of Simulation*, 4(3):204–210.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., et al. (2016). Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., et al. (2017). Mastering the game of go without human knowledge. *nature*, 550(7676):354–359.

Smith, S. L., Kindermans, P.-J., Ying, C., and Le, Q. V. (2017). Don't decay the learning rate, increase the batch size. *arXiv preprint arXiv:1711.00489*.

Soumis, F., Ferland, J. A., and Rousseau, J.-M. (1980). A model for large-scale aircraft routing and scheduling problems. *Transportation Research Part B: Methodological*, 14(1-2):191–201.

Su, X. and Zenios, S. A. (2005). Patient choice in kidney allocation: A sequential stochastic assignment model. *Operations research*, 53(3):443–455.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction*. MIT press.

Tanno, R., Arulkumaran, K., Alexander, D., Criminisi, A., and Nori, A. (2019). Adaptive neural trees. In *International Conference on Machine Learning*, pages 6166–6175. PMLR.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 5026–5033. IEEE.

Van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Vinyals, O., Babuschkin, I., Czarnecki, W. M., Mathieu, M., Dudzik, A., Chung, J., Choi, D. H., Powell, R., Ewalds, T., Georgiev, P., et al. (2019). Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575(7782):350–354.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., et al. (2020a). Scipy 1.0: fundamental algorithms for scientific computing in python. *Nature methods*, 17(3):261–272.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., van Mulbregt, P., and SciPy 1.0 Contributors (2020b). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272.

Wang, J.-B. (2006). Single machine scheduling with common due date and controllable processing times. *Applied Mathematics and Computation*, 174(2):1245–1254.

Wang, Y., Estefen, S. F., Lourenço, M. I., and Hong, C. (2019). Optimal design and scheduling for offshore oil-field development. *Computers Chemical Engineering*, 123:300–316.

Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., and Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In *International conference on machine learning*, pages 1995–2003. PMLR.

Weng, L. (2018). Policy gradient algorithms. *lilianweng.github.io/lil-log*.

Williams, R. J. (1992). Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256.

Zenios, S. A., Chertow, G. M., and Wein, L. M. (2000). Dynamic allocation of kidneys to candidates on the transplant waiting list. *Operations Research*, 48(4):549–569.

Zhou, C., Li, H., Lee, B. K., and Qiu, Z. (2018). A simulation-based vessel-truck coordination strategy for lighterage terminals. *Transportation Research Part C: Emerging Technologies*, 95(July):149–164.

Appendix A

Results easy case

Results small case

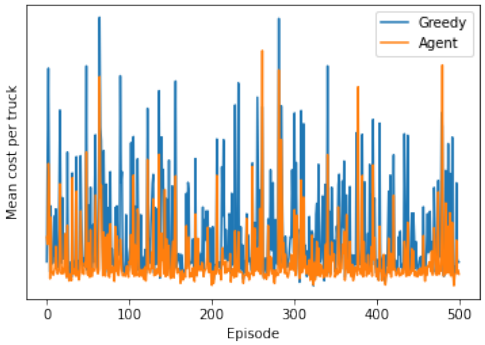

Figure A1: Lineplot of mean costs per truck (small case)

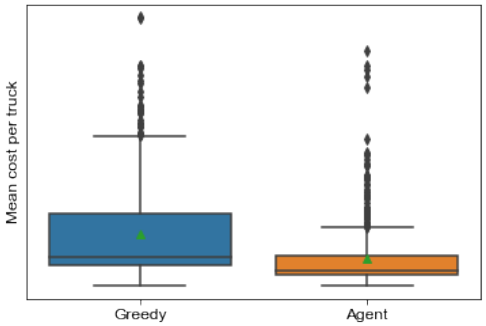

Figure A2: Boxplot of mean costs per truck (small case)

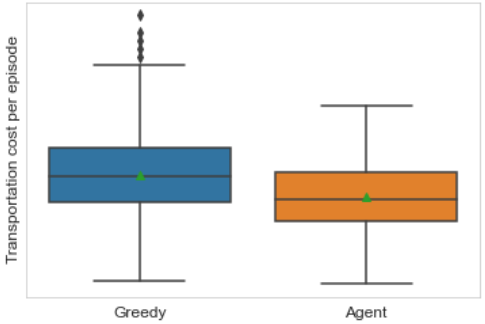

Figure A3: Boxplot of transportation costs per episode (small case)

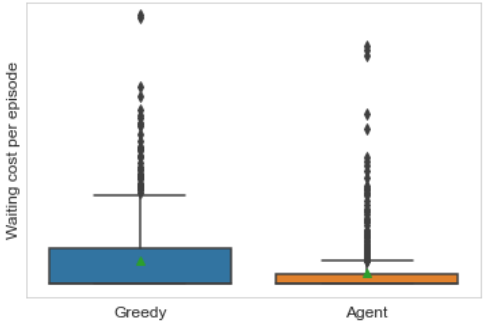

Figure A4: Boxplot of waiting costs per episode (small case)

Appendix B

Sensitivity analysis

Sensitivity analysis arrival rate 8.05 (hard case)

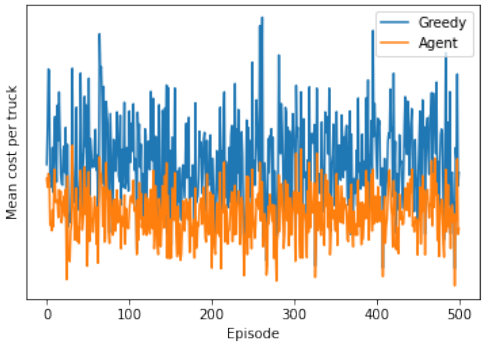

Figure B1: Lineplot of mean costs per truck — arrival rate 8.05

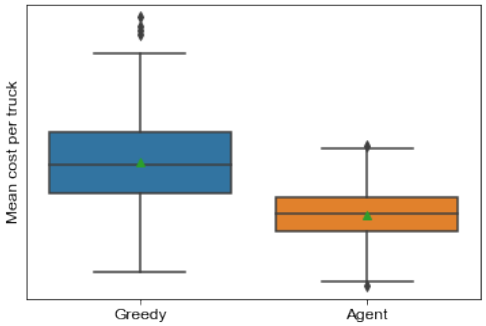

Figure B2: Boxplot of mean costs per truck — arrival rate 8.05

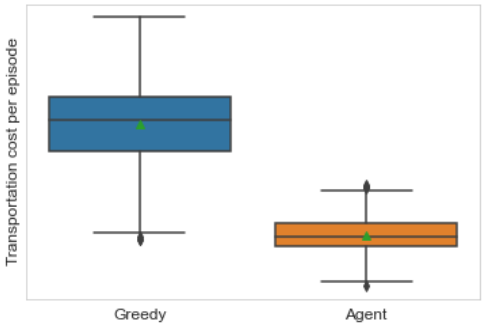

Figure B3: Boxplot of transportation costs per episode — arrival rate 8.05

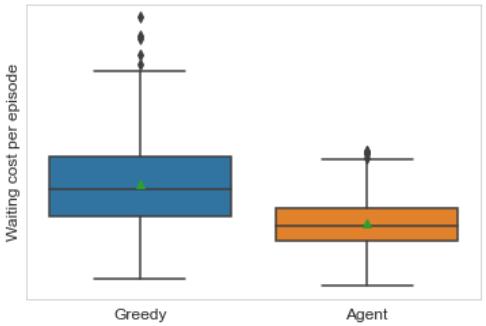

Figure B4: Boxplot of waiting costs per episode — arrival rate 8.05

Sensitivity analysis arrival rate 4.05 (hard case)

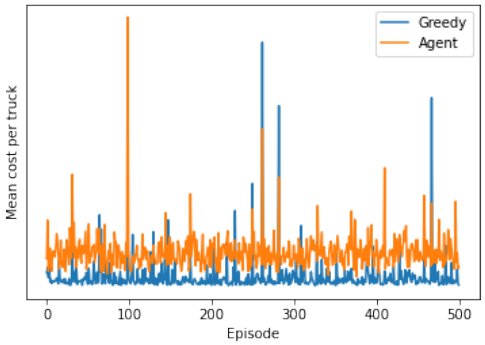

Figure B5: Lineplot of mean costs per truck — arrival rate 4.05

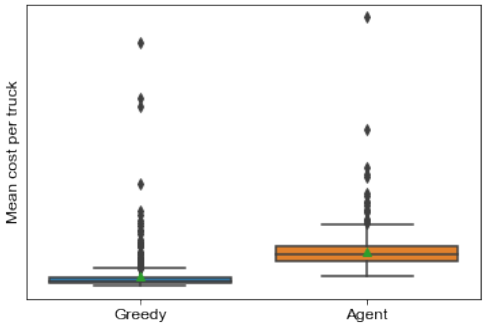

Figure B6: Boxplot of mean costs per truck — arrival rate 4.05

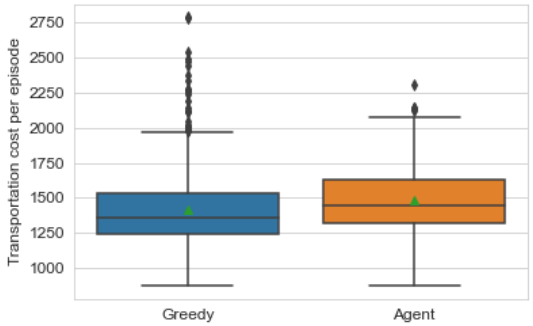

Figure B7: Boxplot of transportation costs per episode — arrival rate 4.05

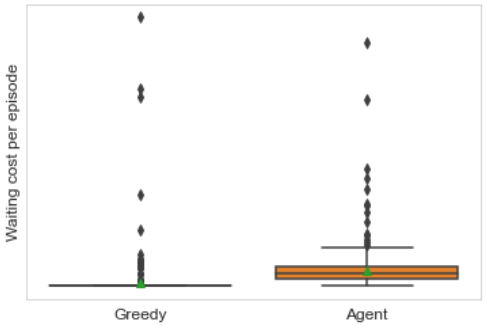

Figure B8: Boxplot of waiting costs per episode — arrival rate 4.05

Sensitivity analysis two operators staging lanes (hard case)

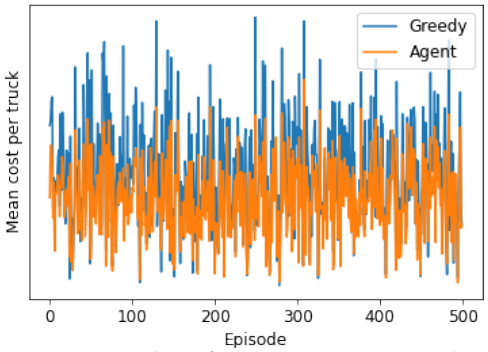

Figure B9: Lineplot of mean costs per truck — 2 operators staging lanes

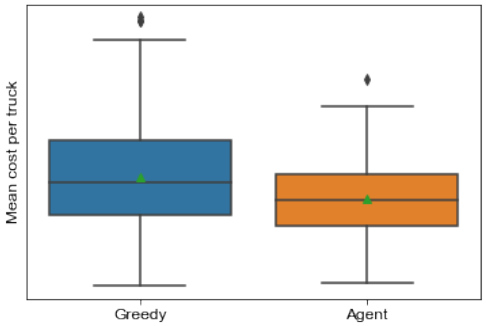

Figure B10: Boxplot of mean costs per truck — 2 operators staging lanes

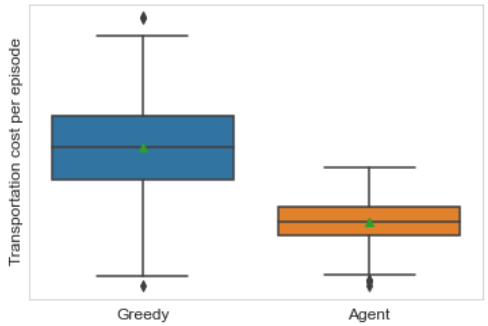

Figure B11: Boxplot of transportation costs per episode — 2 operators staging lanes

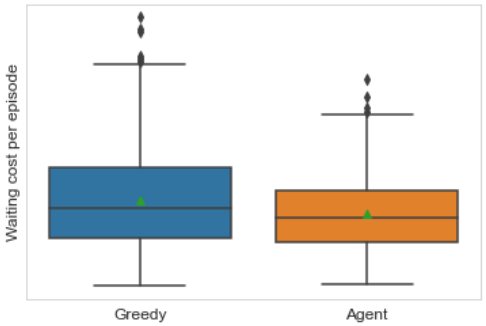

Figure B12: Boxplot of waiting costs per episode — 2 operators staging lanes

Sensitivity analysis four operators staging lanes (hard case)

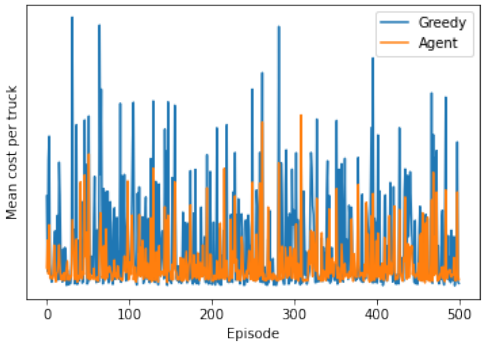

Figure B13: Lineplot of mean costs per truck — 4 operators staging lanes

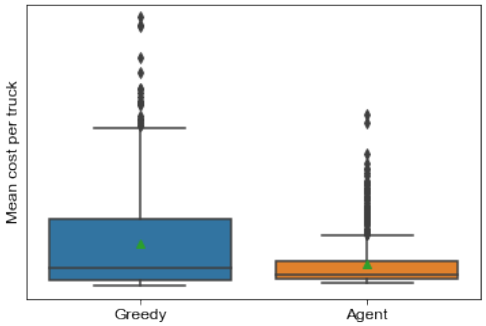

Figure B14: Boxplot of mean costs per truck — 4 operators staging lanes

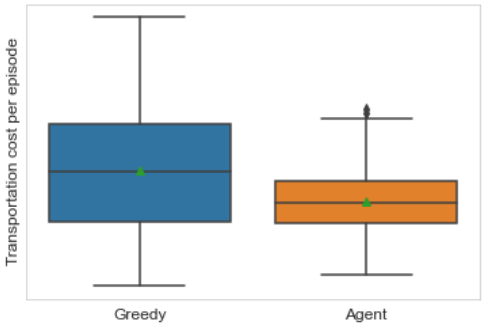

Figure B15: Boxplot of transportation costs per episode — 4 operators staging lanes

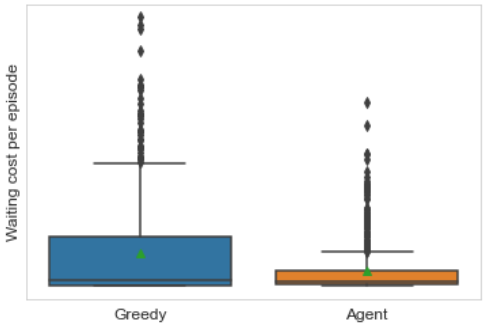

Figure B16: Boxplot of waiting costs per episode — 4 operators staging lanes

Appendix C

Policy analysis

Explaining the agent's behavior

Figure C1: Local explanation summary of the small case