Dynamic tuning of scheduling parameters at a simulated container terminal

Combining a reinforcement learning algorithm based on Reward Backpropagation Prioritized Experience Replay and a Double Deep Q-Network with the Taguchi Method into a dynamic scheduling algorithm

by

Lieke Daniëls (SNR: 2026583)
B.Sc. Tilburg University 2021

A thesis submitted in partial fulfillment of the requirements for the degree of
Master of Science in Business Analytics and Operations Research

Tilburg School of Economics and Management
Tilburg University

Supervised by:
prof. dr. ir. H.A.M. Daniels
dr. ir. ing. M.J.P. Peeters

Date: August 11, 2022

Abstract

This thesis addresses the design and implementation of a dynamic scheduling algorithm in the container terminal simulation software of TBA, with the goal of increasing terminal productivity. A reinforcement learning algorithm that combines the algorithmic choices of Reward Backpropagation Prioritized Experience Replay and a Double Deep Q-Network is discussed and implemented. This algorithm derives scheduling decisions for each of the 144 defined states of the simulated container terminal and the resulting decisions are merged with the existing scheduling algorithm of TBA. The Taguchi Method is applied to determine desirable scheduling parameter settings for each of the defined states and the resulting dynamic scheduling algorithm yields statistically significant increases in quay crane productivity for 6 out of 9 conducted experiments. Statistically significant increased quay crane productivities are mostly found in experiments with 3 and 3.5 horizontal transport vehicles - vehicles responsible for the transport of containers from quay cranes to the storage yard and vice versa - per quay crane. Furthermore, the resulting algorithm particularly increases the productivity of quay crane twin load moves, in which quay cranes load two containers simultaneously from the quay onto a vessel. Additional experiments are conducted to test the robustness of the dynamic scheduling algorithm to a reduction in daily terminal volume and yield a statistically significant increase in quay crane productivity for the off-peak experiments with 3 and 3.5 horizontal transport vehicles per quay crane.

Acknowledgements

First of all, I would like to thank my colleagues and supervisors at TBA for sharing their knowledge and for giving me a delightful six months as an intern at TBA. Mostly, I would like to thank Pim van Leeuwen for his dedication and support on a regular basis. He provided the perfect balance between supervision and independence and was always reachable for a clear explanation or to think along with me.
Furthermore, I would like to thank my supervisors at Tilburg University for their valuable insights and for taking the time to read a relatively lengthy thesis that does not address the most straight-forward methods nor context.
Lastly, I would like to thank my family and friends for giving me the opportunity and support to pursue a second career.

Lieke Daniels
Tilburg, August 2022

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

AGV	Automated Guided Vehicle
ANN	Artificial Neural Network
ASC	Autonomous Straddle Carrier
B&B	Branch and Bound
BSP	Berth Scheduling Problem
DP	Dynamic Programming
DSA	Dynamic Scheduling Algorithm
DQN	Deep Q-Network
GA	Genetic Algorithm
GRASP	Greedy Randomized Adaptive Search Procedure
GVS	Greedy Vehicle Search
HCT	Hadarom Container Terminal
HFS	Hybrid Flow Shop
HTV	Horizontal transport vehicle
ISO	International Organization for Standardization
JSP	Job-Shop Problem
KPI	Key Performance Indicator
MCF	Minimum Cost Flow
MDP	Markov Decision Process
NN	Neural Network
NSA	Network Simplex Algorithm
QC	Quay crane
QCSP	Quay Crane Scheduling Problem
QCSNIP	Quay Crane Scheduling with Non-Interference Constraints Problem
PER	Prioritized Experience Replay
POMDP	Partially Observable Markov Decision Process
RBPER	Reward Backpropagation Prioritized Experience Replay
RL	Reinforcement learning
RLA	Reinforcement Learning Algorithm
RNN	Recurrent Neural Network
ShC	Shuttle Carrier
SGD	Stochastic Gradient Descent
SMDP	Semi-Markov Decision Process
SN	Signal-to-noise
SPO	Sequential Parameter Optimization
TBAA	TBA's scheduling algorithm
TEU	Twenty-foot Equivalent Unit
TLBO	Teaching-Learning-Based Optimization
TOS	Terminal Operating System
TP	Transfer point
TS	Tabu Search
TT	Terminal Truck
UIMA	Universal Island-based Meta-heuristic Algorithm
YC	Yard crane

Chapter 1

Introduction

Over the last decades global maritime container trade has grown substantially in size and importance. In 1980 the total global amount of goods carried by containers was around 102 million metric tons and in 2017 this amount reached 1.83 billion metric tons (Nagurney, 2021). This stimulated a large increase in vessel capacities: the dead-weight tonnage (weight carrying capacity) of container ships has grown from a maximum of approximately 11 million metric tons in 1980 to over 275 million metric tons in 2020 (Placek, 2021). A growth rate of this magnitude demands solid adaptivity of international supply chains, container terminal operators and shipping lines. But global demand and supply does not always increase: during the financial crisis of 2008/2009 changes in consumption patterns reduced world container shipping demand by 12.4% and the COVID-19 crisis in 2020 marked the beginning of a sequence of supply chain disruptions all over the world, leading to a variety of operational challenges for all parties involved (Notteboom et al., 2020). Combined with the strong bargaining position shipping lines have - due to the large number of container terminals to choose from - this results in a highly competitive market environment for container terminals. Increasing efforts of container terminal operators is required to improve the productivity and efficiency of their container terminal, at a minimal cost. Productivity of a container terminal is often measured by the turnaround time of vessels (by taking into account the load and size of the vessel) or by yearly container throughput in TEU (Twenty-foot Equivalent Unit) (Doerr and Sánchez, 2006). The majority of containers on a typical container terminal arrives and/or leaves by vessel and therefore has to pass by the quay of the terminal at least once. Containers at the quay are handled by a quay crane (QC). QCs serve vessels and barges: they discharge containers from ships onto internal transportation modes and load containers from these internal transportation modes onto ships. A QC is very expensive and needs a lot of (costly) space. Therefore, QC productivity is of high importance; it is the major Key Performance Indicator (KPI) for most container terminal operators. One of the operations on the terminal that influences QC productivity is scheduling: the scheduler assigns jobs (container handling or container moves) to various equipment types. A job could be, for example, picking up container A at QC 1 and bringing it to a specific place in the yard. If a container is not at the QC at the time it is required for loading, or if too many discharged containers are waiting at a QC to be picked up, QC productivity is negatively affected. The goal of this research is to investigate and implement possible improvements in the container terminal scheduling algorithm in order to increase QC productivity, commissioned by and therefore specifically designed for TBA Group in Rijswijk.

1.1 Company information

TBA Group is a globally operating consultancy and software company that uses simulation and emulation software to optimize the logistic processes of ports, container terminals, airports, manufacturing plants and other logistic systems. It is a subsidiary of Konecranes, a Finnish company specialized in manufacturing and service of cranes and lifting equipment. TBA was founded in 1988 in the Netherlands and currently has its headquarters in Rijswijk. Other offices are located in Leicester and Doncaster (United Kingdom), Satu Mare (Romania), and Düsseldorf (Germany). Local company representatives are active across North and South America and Oceania.
The services provided by TBA range from the early stages in the design phase of a terminal to live operations, both for new as well as existing terminals with a need for automation or expansion. Examples of these offered services are: Terminal Design, Trainings for Terminal Operators, Performance Improvement and Analysis and implementation of the Terminal Operating System (TOS).
In order to find efficient solutions for various types of terminals or warehouses, TBA develops simulation models for terminal performance studies under different strategies. These models can be adapted to fit the exact design of any terminal or warehouse and are used to test the performance of different strategies (like container placement, routing and job dispatching), equipment types or specifications. This thesis focusses on the scheduling algorithm that is used in one of the simulation programs of TBA: TIMESQUARE. The dynamic scheduling algorithm could later be implemented in the TBA Scheduler to be applied in real-life settings.

1.2 Problem description

The current scheduling algorithm used by TBA is a relatively straightforward approach with a scoring mechanism. Central planning generates a list of orders: an order indicates for a specific container where it is currently located and where it has to go to. An example for an **order** could be:

- *Container A that is located in stack B must be loaded at quay crane C.*

An order can be divided into multiple **moves**. In the above example the corresponding moves could be:

- *A Stack Crane must move container A from stacking location B onto a Terminal Truck.*
- *Container A must be transported from location B to location C by a Terminal Truck.*

The scheduler is supplied with the complete list of moves and their (approximate) due times and assigns **jobs** to the various equipment types. A job consists of an instruction to one specific machine, for example:

- *Get container A at location B and drop it at location C.*

The first step of the scheduling algorithm is to match the equipment type to each of the moves it can theoretically make, resulting in a job list with nxm jobs, where n is the number of machines on the terminal of the equipment type that is currently

under consideration and m is the number of moves that require a machine of that type.
The next step of the algorithm excludes all jobs that should not be taken into consideration. For example, machines that are currently performing another job cannot be assigned to the next job.
After reducing the size of the job list only the useful jobs remain and the scoring algorithm starts: based on k variables x_i and parameters α_i each job receives a score with value $\sum_{i=1}^{k} \alpha_i x_i$. Variable x_1 could be the *driving distance in meters* and its corresponding parameter α_1 could have the value 2. Due times of the jobs - based on the time an order is required to be completed - are also included in the variables x_i. Note that $\alpha_i \in \mathbb{R}$, which results in an infinite search space of parameter values, and that different equipment types can have different variables and parameters. The variables x_i can attain binary or continuous values. In the above example of *driving distance in meters* $x_1 \in \mathbb{R}^+$, but the variable *isdischarging* can only be 0 (if the machine is not discharging) or 1 (if it is). Therefore, a change in parameter values α_i has a different effect on the total score for different variables x_i.
In the last step of the algorithm the job with the highest score is scheduled as the next job and the algorithm starts again. The scheduling algorithm is depicted in Figure 1.1.

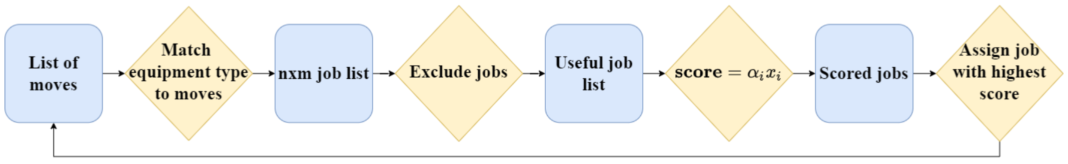


Figure 1.1: Scheduling algorithm TBA.

The objective of the scheduler is to schedule as many useful jobs from the job list as possible within a given time frame, in order to maximize terminal productivity.
The parameter values α_i $(i = 1, ..., k)$ have a large impact on the performance of the terminal and are currently tuned based on experience of simulation engineers and extensive experiments. They are, however, set in a static range, which means that the same parameter settings are used throughout a full range of experiments. TBA expects that a dynamic parameter setting, in which different scoring parameter values are considered for different circumstances, could benefit terminal productivity. The aim of this research is therefore to adapt the current scheduling algorithm and implement it in TIMESQUARE such that the parameter settings are tuned dynamically. This requires a thorough investigation of different situations that might occur during the scheduling cycle that would require different parameter settings. On top of that, a procedure that reduces the infinitely large search space of parameter settings is needed to ultimately find the parameter values that provide the largest increase in terminal productivity for each situation.

1.3 Research questions

The main topic of this research is to investigate the effect of implementing dynamic scheduling parameter settings on the productivity of the container terminal. Due to their location at the quay, high operational costs and large size, QCs are important bottlenecks of a container terminal and therefore QC productivity is one of the main drivers of container terminal productivity. We can therefore measure the performance of the adapted scheduling algorithm by investigating the effect on QC productivity and formulate the following research question (RQ):

Can QC productivity on container terminals be improved by adapting a static scheduling parameter setting to a dynamic parameter setting, and thereby ultimately result in a more productive container terminal?

Five sub questions (SQ) have been formulated to provide guidance in answering the main research question:

1. Which models and methods have previously been used to optimize container terminal scheduling and could these be effective in this setting?
2. Which circumstances during the process of container handling on a container terminal might require a change of parameter setting to improve equipment productivity?
3. How can the infinite search space of parameter values be reduced without loosing valuable parameter settings?
4. Is the effect of the improved scheduling algorithm on the computational time of the scheduler acceptable and/or can this be decreased?
5. What is the expected improvement in QC productivity when the dynamic parameter setting is implemented?

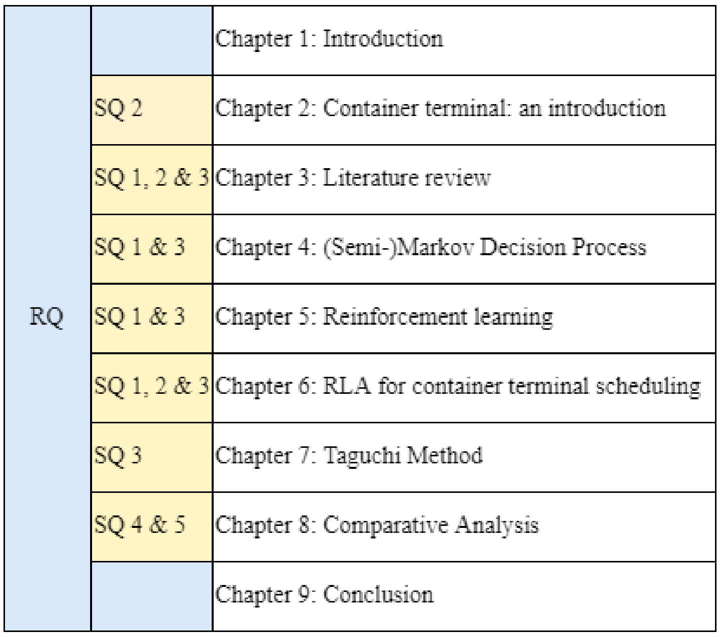
1.4 Strategy and expected results

The current scheduling algorithm is an uncoordinated, greedy algorithm that is based on penalties and rewards. Throughout this thesis we refer to this algorithm as TBAA (TBA's Algorithm).
In an uncoordinated scheduling algorithm there is no coordination between different equipment types, which could lead to a decrease in terminal productivity. Note that this approach decouples the scheduling of different equipment types, but moves that belong to a specific order always have to be performed according to a predefined sequence. Therefore, jobs will only be assignable to a certain type of equipment once the previous equipment type has completed the job that concerns the same container.
A former intern at TBA has investigated the possibility of integrating a coordinated scheduling algorithm (Jonker, 2018) by scheduling complete orders instead of separate jobs that belong to the same order. A Simulated Annealing algorithm was implemented, but did not result in a significant improvement in QC productivity. Furthermore, the algorithm was computationally too expensive, compared to TBAA, which is a greedy algorithm that decides for every equipment type on the

next move one-at-a-time.
This research therefore does not result in the implementation of a complete new scheduling algorithm, but mainly aims to enhance TBAA by implementing a dynamic strategy and improving the existing parameter settings.
Previous research (discussed in Chapter 3) shows promising results of reinforcement learning (RL) algorithms when applied to scheduling problems in a wide variety of logistic operations. RL is an area of machine learning in which digital agents are trained in an environment to take specific actions with the goal of maximizing some notion of cumulative reward.
After defining a suitable environment, actions and rewards for the setting in this thesis, a reinforcement learning algorithm that decides on the job scheduling at the container terminal will be implemented. In this thesis, we refer to this algorithm as RLA (Reinforcement Learning Algorithm). The effect of the adapted schedules on QC productivity will be assessed and the schedules will undergo a thorough analysis. Since integration of RLA - and any machine learning based algorithm - in the simulation program of TBA is computationally too exhaustive, parameter settings will be derived from the actions that are taken by the reinforcement agent in specific situations. The Taguchi Method will be used to efficiently assess the performance of different parameter settings and to derive the optimal setting. Afterwards, the dynamic parameter settings will be implemented in TBA's simulation model and the effect on QC productivity and computational time will be analysed. The final scheduling algorithm algorithm is referred to as DSA (Dynamic Scheduling Algorithm).
We expect the performance of RLA to be highly dependent on the defined state and action space. A complex state and action space will presumable improve the current TBAA schedules, but make the final implementation in TIMESQUARE more demanding. A smaller state and action space captures less information of the simulation model and might therefore decrease QC productivity when compared to TBAA. The computational complexity of the algorithm will, however, decrease and the interpretation of the results will be more straight-forward. A balance between the two situations is therefore the most desirable methodology. Deriving suitable parameter settings and implementing these in the current algorithm results in an educated combination of TBAA and RLA and is therefore expected to increase QC productivity when compared to the currently used schedules, as created by TBAA. Hence, due to its dynamic nature and the fact that the current parameters are set by trial-and-error we expect an improvement in QC productivity over TBAA after the implementation of DSA.
Since it is of large interest for TBA that the adapted algorithm does not increase computational time substantially, the choice was made to implement the dynamic parameter settings and the situations they apply to and not RLA into their software. The final result then still is a greedy algorithm that will not be computationally too exhaustive. A slight increase in computational time will be caused by the dynamic implementation: depending on the number of situations in which a parameter setting has to be revised, the parameter settings have to be updated. It is anticipated that this is computationally not demanding and that these efforts will be surpassed by the benefits of an increased QC productivity.

1.5 Outline

This introductory chapter concludes with an outline of this thesis: Chapter 2 consists of an introduction to the design of and operations on a container terminal, in order to provide sufficient support to understand the context and scope of this thesis. Chapter 3 contains a literature study on the topic of this thesis, divided into three parts: planning and scheduling algorithms for container terminals, reinforcement learning algorithms that were applied in comparable settings and various methods that concern (hyper-)parameter tuning. In Chapter 4 we provide the theoretical background for the processes that form the basis for reinforcement learning algorithms: Markov Decision Processes. In Chapter 5 the connection with reinforcement learning is made: we gradually build towards the existing algorithms that are combined in the implementation of RLA. The defined state space, action space and reward function of RLA are specified in Chapter 6, along with the details of the implementation of the algorithm. Chapter 7 introduces the Taguchi Method and discusses its application in the context of this thesis with the goal of merging TBAA with RLA into the final algorithm: DSA. Chapter 8 provides an extensive analysis of the results of both RLA as well as DSA and compares these to TBAA. Additional experiments that test the performance of DSA on off-peak simulation days are discussed in this chapter as well. The final chapter of this thesis, Chapter 9 starts with a brief summary, followed by concluding remarks and recommendations for future research. Figure 1.2 indicates which research sub questions (SQs) are addressed in which chapter of this thesis.



		Chapter 1: Introduction
	SQ 2	Chapter 2: Container terminal: an introduction
	SQ 1, 2 & 3	Chapter 3: Literature review
	SQ 1 & 3	Chapter 4: (Semi-)Markov Decision Process
RQ	SQ 1 & 3	Chapter 5: Reinforcement learning
	SQ 1, 2 & 3	Chapter 6: RLA for container terminal scheduling
	SQ 3	Chapter 7: Taguchi Method
	SQ 4 & 5	Chapter 8: Comparative Analysis
		Chapter 9: Conclusion

Figure 1.2: Thesis outline with specification of the research question (RQ) and sub questions (SQ) that are addressed in each of the chapters.

Chapter 2

Container terminal: an introduction

In this chapter we introduce the reader to the most important operations that take place on a container terminal, as well as discuss the terminal design and different equipment types that can be used. This way, sufficient support is provided to understand the details and scope of this thesis. The knowledge was acquired by following TBA's internal container terminal course. Section 2.1 discusses terminal design, operations and trends and Section 2.2 addresses the different types of containers that are commonly used. Section 2.3 introduces the processes at the waterside of a container terminal and Section 2.4 discusses the function of horizontal transport vehicles and the specific type that is used in the experiments of this thesis: the Shuttle Carrier. Section 2.5 addresses the layout and function of the storage yard and Section 2.6 briefly discusses the operations that take place at the landside of a container terminal. The chapter concludes with Section 2.7 by addressing the main subject of this thesis: the scheduling decisions on a container terminal, along with the challenges that a scheduler faces.

2.1 Design, operations and trends

Marine container terminals form an important link in intercontinental container flows, by providing transshipment from large deep-sea vessels to smaller barges for inland waterways, trucks, trains and vice versa. Furthermore, container terminals provide valuable storage space to hold containers both long- as well as short-term, which allows for the decoupling of intercontinental and continental container flows in time. Direct transshipment - without the storage capacity - would be too complex and just-in-time delivery of containers barely possible (van Zijderveld, 1995). Container terminals sometimes provide additional services: repairs, cleaning, inspection and/or cargo consolidation (when the contents of one container do not share the same origin or destination).
Figure 2.1 provides a simplified, schematic side view of the general design of a container terminal. From left to right the operations consist of the following:

- Vessels and/or barges berth at the waterside (also called quayside or seaside). Generally, multiple vessels can berth at one terminal simultaneously and sometimes barges are handled at a specific barge facility instead of the deep-sea quay.

- Quay cranes (QCs) discharge containers from vessels and barges onto horizontal transport vehicles (HTVs, also called internal transportation modes) and load containers from these HTVs onto the vessel.
- HTVs transport containers from the (storage) yard to the QCs and vice versa. Depending on the type, HTVs are able to pick up and place containers themselves or need to wait for an available crane to assist them in these procedures.
- Yard cranes (YCs) load and discharge containers onto and from the HTVs or place them in buffers (small areas where containers can wait for a short time period), where they can be picked up by HTVs.
- Containers are (temporarily) stored in the yard (also called storage yard or stack). They can be placed on top of each other and are handled by YCs.
- On the landside of the terminal YCs load and discharge containers onto and from HTVs or directly on external trucks.
- Trains are loaded and discharged by cranes and HTVs transport these containers to the yard.
- Trucks arrive through the gate to pick-up and/or drop-off containers at the landside of the terminal.

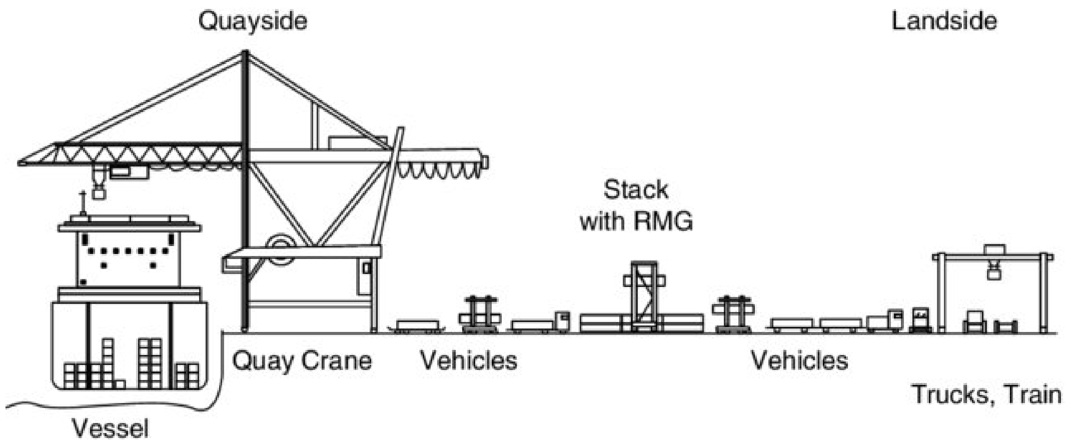


Figure 2.1: Side view general container terminal, not true to size (Steenken et al., 2004).

Note that these operations do not need to take place in the order they were discussed, depending on the origin and destination of a container. We distinguish four types of container flows:

- Import: containers arrive by vessel and leave by truck, train or barge.
- Export: containers arrive by truck, train or barge and leave by vessel.
- Transshipment: containers arrive by vessel and leave by vessel.
- Domestic: containers arrive by truck, train or barge and leave by truck, train or barge.

Some container terminals are devoted to one specific type of container flow, while others are capable of performing each of the discussed operations in both directions.

As stated in the introductory chapter, increasing demand for global maritime container trade has led to large increases in vessel capacities and demands solid adaptivity of container terminal operators. Furthermore, the highly competitive market demands a large degree of innovation, while costs must be kept minimal (Notteboom et al., 2020). As a consequence, an important trend in container terminal design is automation: ranging from fully-automated terminals to (partly) automated terminal equipment. Integrating automation is a complex task, not in the least with regards to safety of people working on or interacting with a (semi-)automated terminal. This requires thorough testing, preferably outside of the real terminal or - even better - before the terminal is built. Simulation provides a safe, thorough and relatively cheap alternative to real-life testing, by helping container terminal operators make educated decisions on terminal design, equipment, expansion and capacity. Furthermore, simulation tools are essential in determining the robustness of a proposed terminal design to possible future changes in global and regional economy, equipment types, environmental regulations and demand fluctuations (Saanen, 2004). Since the average process time from the first ideas until commissioning takes six to seven years (Saanen, 2004) and requires a significant amount of money, the design must account for future scenarios in the best way possible.

2.2 Containers

The intermodal containers that we know today were first introduced in the 1950s and are ISO standardized steel boxes constructed to transport cargo and to be efficiently stacked on top of each other. Twistlocks at each of the eight corners of a (regular) container provide the opportunity to safely secure containers that are stacked on top of each other. On vessels, these twist locks are combined with lashing rods to keep the containers in place (see Figure 2.2).

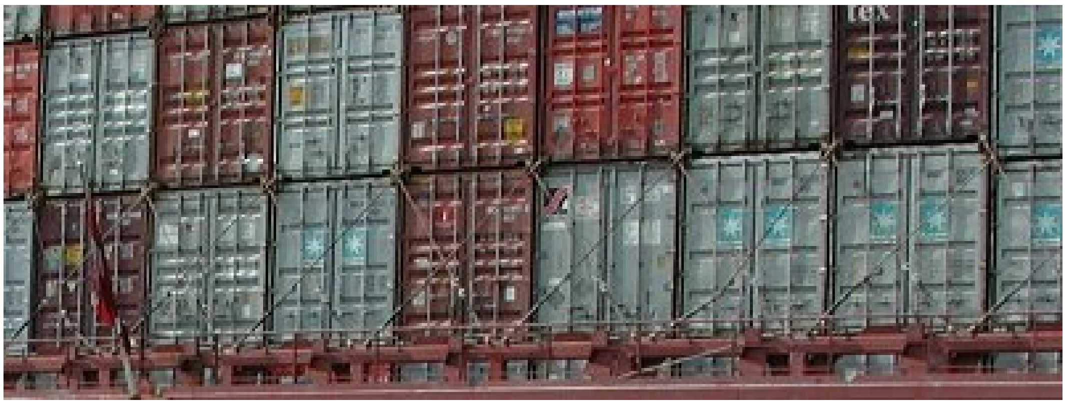


Figure 2.2: Standard containers secured with lashing rods (TBA Training Portal).

The ISO standardization allows for a global stacking efficiency, safety and generalization in the specifications of various equipment types all over the world. Different types of containers exist, each for specific types of cargo. An overview of the most commonly used container types is given in Figure 2.3. Most come in two sizes: 20

feet and 40 feet, with external dimensions of (length × width × height in feet'inches) 20'0 × 8'0 × 8'6 and 40'0 × 8'0 × 8'6, respectively. Container capacity and volume is measured in Twenty-Foot-Equivalent Units (TEUs), expressing the equivalence with the number of 20 ft containers. Due to the different-sized containers it is unclear what is meant by a vessel with a capacity of 1000 containers; holding only 20 ft containers (a TEU-factor of 1) would require a significantly smaller vessel than one with only 40 ft containers (a TEU-factor of 2). Therefore, TEU is used for all capacity and volume related container descriptions.

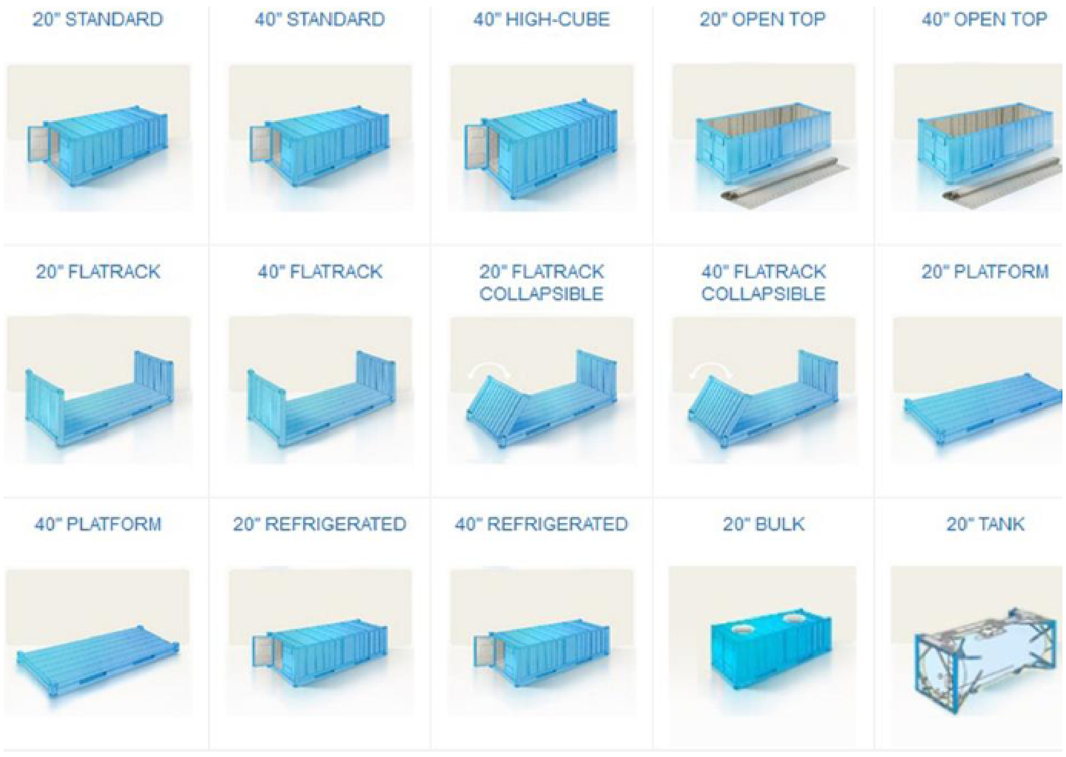


Figure 2.3: Different types of intermodal containers (TBA Training Portal).

We briefly discuss the different container types that are depicted in Figure 2.3:

- Standard containers are the most commonly used containers. If cargo permits, these container types are preferred.
- High-cube containers are standard containers with a length of 40 feet and an increased height of 9 feet and 6 inches.
- Open top containers have a larger capacity than standard containers, because the cargo can stick out the top of the container. This is especially useful for large machinery or construction materials. Lashing rings are used to keep the cargo stable.
- Flatrack containers only have walls on the short side of the container and therefore allow the cargo to stick out the side of the container. These containers are used to transport trucks and boats that do not fit in regular containers. The collapsible version can be shipped more efficiently when empty, but can endure slightly smaller top loads than the fixed end flatrack containers.

- Platform containers do not have any side walls and are used for extremely large and/or heavy cargo such as airplane parts and heavy, odd-shaped machinery.
- Refrigerated containers (also called reefers) possess an interal refrigeration unit and are therefore capable of transporting temperature-sensitive, perishable cargo, like meat or fruits. Reefers rely on external power and therefore often have specifically assigned places on vessels and container terminal yards. Reefers do have ISO standardized dimensions (mostly 40 feet) and can therefore be handled by regular container terminal equipment types.
- Bulk containers are only produced with a length of 20 feet and are designed for the transport of bulk goods, such as corn, pet food and grain. The containers have manholes on the top and hatches on the doors to load and unload the bulk goods.
- Tank containers transport liquid, gases and powders and consist of a vessel made of stainless steel, surrounded by a protective layer and a steel frame. The containers have a manhole and at least one valve on the top and another valve on the bottom. Both hazardous as well as non-hazardous goods can be transported with tank containers. Dangerous goods should always be reported and clearly marked on the outside of the container, and will generally be handled and stowed separately.

2.3 Waterside

At the waterside (also called quayside or seaside) vessels and barges berth alongside the terminal. Vessels transport containers overseas and barges transport containers through inland waterways. The size and the number of vessels that can be serviced simultaneously is bounded by the size of the quay, by the QCs that load and discharge containers onto and from vessels and by the water depth. Shipping lines generally want to minimize the time their vessels spend at a container terminal, making it crucial for terminal operators to minimize the turnaround time of vessels (the time frame between the arrival and departure) and to avoid congestion of vessels that are waiting to be serviced. To save costly space at the quay, barges are sometimes handled at a specific barge facility, where the water depth or handling equipment is not suited for large deep-sea vessels.

QCs are among the most expensive equipment types on a container terminal and require a lot of space alongside the quay. Since QCs are responsible for loading and discharging vessels and minimizing the turnaround time of vessels has a high priority for container terminal operators, QC productivity needs to be as high as possible. A QC can only discharge a container from a vessel when there is a place to drop the container at the quay or onto another type of equipment, and can only load a container on a vessel when this container is ready to be picked up. Loading QCs need to follow the bayplan of the vessel: a plan that describes which container should be placed at which location on the vessel.

Some vessels have on-board cranes that are capable of loading and discharging the vessel without the need for appropriate handling equipment at the quay. Furthermore, Mobile Harbour Cranes - cost-attractive, smaller cranes that are easily repositioned and capable of handling all types of cargo - can be used to replace or complement regular QCs. In this thesis we focus solely on QCs that can lift one

(single-lift) or two (twin-lift) containers at the same time. These twin-lift QCs carry two 20 feet containers as if they are one 40 feet container (end-to-end). Obviously, a QC that is performing twin-lift operations can achieve a higher productivity when measured in the number of containers handled, compared to a QC that performs single-lift operations. QCs with a tandem-lift function can carry two (20 or 40 feet) containers side by side, but these are not considered in the experiments of this thesis.
Each of the QCs places the containers directly in a QC transfer point (TP) that is either located right underneath the QC or at the back of the QC (backreach). The difference between these two placements is indicated in Figure 2.4, where a screenshot of TIMESQUARE depicts three QCs. HTVs arrive through the green rectangles (QC buffers) and wait there until the desired container is at the QC TP. They then drop/pick-up containers underneath or at the back of the QC at the TP, and leave through the blue rectangles.

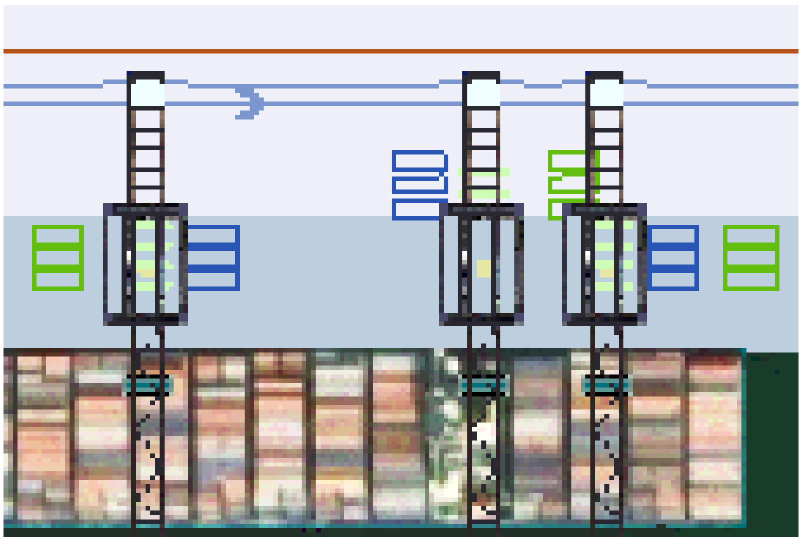


Figure 2.4: Three QCs (in black, with a vessel below and the yard above). The two outer QCs have a TP underneath, with buffers closer to the vessel. The middle QC performs backreach moves with a TP and buffers closer to the yard (Screenshot TBA Simulation model TIMESQUARE).

A QC with backreach needs more time to move a container from the vessel to the buffer and vice versa, and can therefore achieve a slightly lower productivity than a QC with a TP closer to the vessel. On the other hand, HTV driving distance decreases when QC buffers are located closer to the yard. This has a smaller impact on terminal productivity than QC productivity but cannot be ignored.

2.4 Horizontal transport vehicles

HTVs are responsible for container transport from the quay to the storage yard and vice versa. Depending on the terminal layout and functionality they can also be deployed at the landside of the terminal to - for example - transport containers from the storage yard to the train facility. In this thesis we do not include a train facility and the only area where this internal transportation mode is considered is between the quay and the storage yard.

HTV types vary considerably, ranging from fully-automated to human-operated equipment with different functionalities. All HTV types are capable of transporting containers horizontally over the container terminal, while only some of them can also perform (twin-)lifting operations. HTVs that are not equipped with a spreader cannot lift containers themselves and always need another type of handling equipment to assist them.
In this thesis we consider Shuttle Carriers (ShCs) (see Figure 2.5): relatively fast vehicles that are able to pick-up and drop containers autonomously. Both manual-operated as well as fully-automated ShCs exist. ShCs can carry 1-over-1 (or - equivalently - 2 high), meaning that they can carry one container over another container. They can also stack one container on top of one other container and therefore do not need additional equipment to assist them in picking up or dropping containers at QC TPs, in the storage yard or even from and to road trucks.

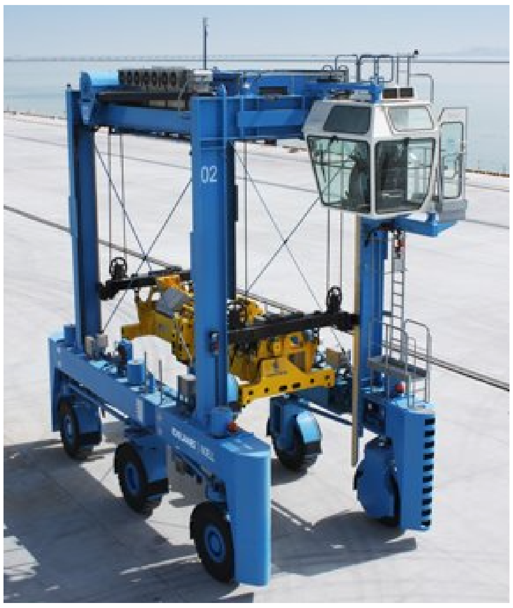


Figure 2.5: Shuttle Carrier (ShC) (TBA Training Portal).

2.5 Yard

The storage yard consists of a number of rectangular stack modules or blocks, separated from each other by roads or small land openings for terminal equipment. The precise location of a container within a module is called a slot and a number of slots that is located on top of each other is called a pile. Each slot is specified by three dimensions: a bay, row and tier, as becomes clear from Figure 2.6. One could address the number of bays as the length, the number of rows as the width and the number of tiers as the height of the module.
The layout of the storage yard is generally either in parallel (the stack modules and quay are parallel) or perpendicular (a 90° angle between the stack modules and quay). The difference becomes clear from Figures 2.7 and 2.8. A clear separation between the waterside and the landside of the yard is made by the perpendicular layout, while the parallel layout allows HTVs to drive to both sides of the terminal.

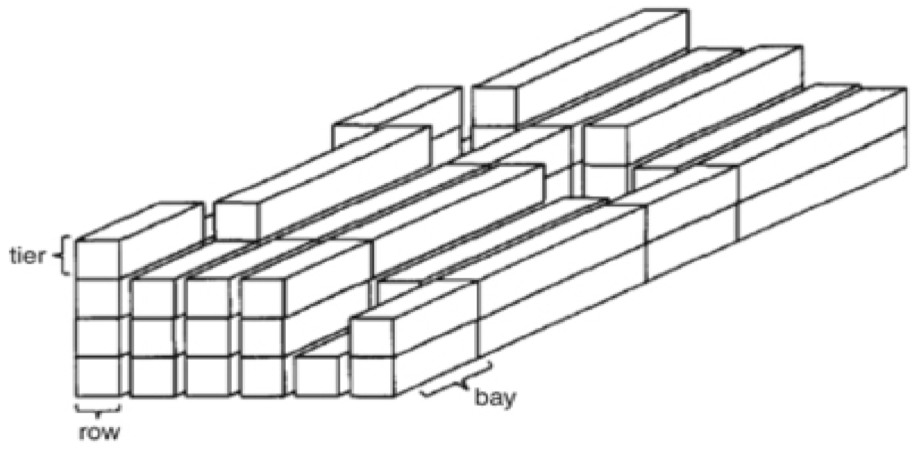


Figure 2.6: Structure of a stack module or block. Each slot is a unique combination of a bay, row and tier number (Luo et al., 2011).

There exists a wide variety of equipment types for container handling in the storage yard: yard cranes (YCs). We briefly discuss the two types in Figure 2.7 and Figure 2.8. Figure 2.7 shows a container terminal with Rubber Tyred Gantry Cranes (RTGs) and in Figure 2.8 Rail Mounted Gantry Cranes (RMGs) are used. Both types are capable of driving along the module; the first on wheels and the second on a specifically built railway. RTGs are cheaper than RMGs, but their productivity is typically slightly lower (10 containers per hour versus 15 containers per hour). In all experiments of this thesis RMGs are responsible for container handling in the storage yard. Depending on the layout of the terminal and equipment types containers are first placed in buffers - as in Figure 2.8 - or placed directly on the HTVs, either at the head of a module or at small transfer roads between modules.

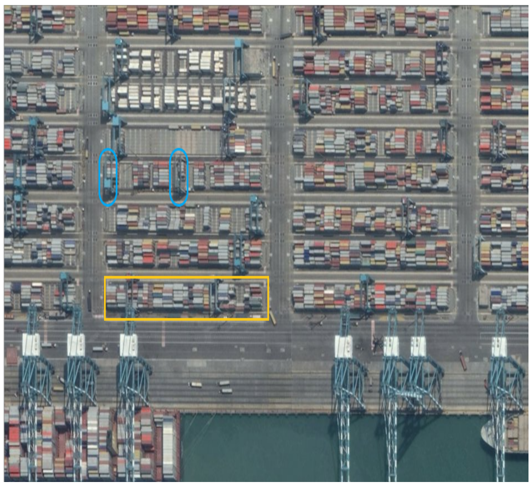


Figure 2.7:
PTP Tanjung Pelepas in Malaysia. Parallel yard and zero, one or two RTGs (blue) per module (yellow) (TBA Training Portal).

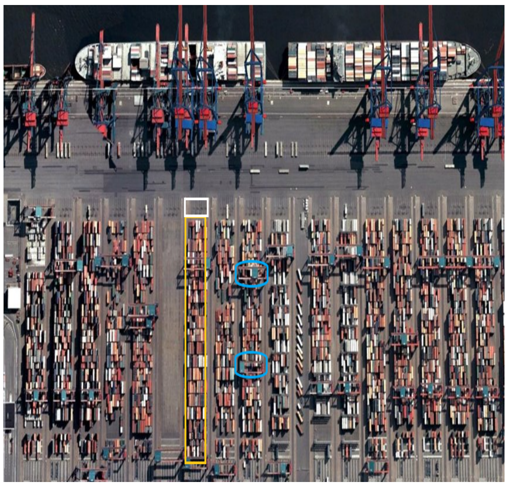


Figure 2.8: Container terminal in Hamburg. Perpendicular yard with two RMGs (blue) per module (yellow) and interchange buffers (white) (TBA Training Portal).

Each of the containers in the storage yard must be reachable by the yard equipment, either to be picked-up for transport, or to be reallocated to another spot in the yard. When a container that is not on the highest tier of its pile is needed for transport, the top container has to be moved to reach the needed container. This is called a rehandle or shuffle move. When containers are reallocated to different piles in order to anticipate leaving and arriving containers (usually off-peak) these moves are called housekeeping or prepositioning moves. The number of rehandle and housekeeping moves should be kept at a minimum, since no payment is received for any of them and it keeps the yard equipment from executing more profitable moves.
Containers are generally grouped together based on similar characteristics, like type, size, weight class and destination, to avoid rehandles as much as possible. For the same reason pile heights are to be kept as equal as possible. Sometimes - for empty containers - the shipping company is indifferent between containers; any empty container of the correct shipping line is allowed to board the vessel, hence grouping empty containers together results in zero rehandles for empty containers. Some container types require separate slots in the yard. Reefers, for example, need a power supply to keep them at the right temperature. Usually (part of) a separate module or a specific place in the yard is reserved for these container types. The same holds for odd-sized containers (like open top containers or flatrack containers, as introduced in Section 2.2) that are not suited for stacking. Tank containers and containers with dangerous or hazardous cargo are also treated with special care and on special yard slots. Depending on their degree of hazard these containers should be stored with a specified minimum distance between each other, to ensure safety on the terminal.

2.6 Landside

At the landside of the terminal trucks and trains arrive to drop-off and/or pick-up containers. Trains arrive at a special railway facility, where containers are loaded and discharged with the aid of cranes and/or HTVs. Trucks arrive through the gate, where various procedures have to take place before the truck driver can pick-up and/or drop-off a container. Most procedures concern (driver, chassis and container) identification, cargo and customs checks, cargo classification, empty container checks and damage detection in order to ensure safety and avoid illegal activities. At some container terminals these procedures can (almost) all be performed automatically, while other terminals perform manual checks. To provide acceptable service levels, congestion at the gate and truck service times must be kept minimal. Since this thesis addresses HTV scheduling at the waterside of the terminal, we assume an acceptable service level at the gate and sufficient hourly truck arrivals to prevent the yard from emptying by lack of drop-off trucks and from overflowing by lack of pick-up trucks. Furthermore, we do not include train arrivals in our experiments, and consider a perpendicular yard with a clear separation between the land- and waterside and can therefore exclude a deeper analysis of the landside processes.

2.7 Scheduling

The scheduler at a container terminal faces the important job of assigning container moves to equipment types at specific points in time with the goal of minimizing the number of rehandles, the turnaround time of vessels, the cost of (unnecessary) equipment use and the driving distance of HTVs, while maximizing the productivity of various equipment types and - ultimately - container terminal profit. Next to these objectives, the scheduler faces a large number of challenges due to uncertain arrival times of trucks and vessels, possible equipment breakouts and weather disruptions. Furthermore, rolling containers - containers of which the destination changes while already in the storage yard - can ruin schedules that were made in advance. It is therefore not possible to address the scheduling problem at a real-life container terminal in an off-line manner (where all job information is known beforehand): it has to be scheduled in a flexible, on-line way (where jobs arrive during the scheduling process).
It is under debate whether better solutions can be found when addressing the multiple scheduling problems of the container terminal scheduler simultaneously (coordinated scheduling) or separately (uncoordinated scheduling). This is further discussed in Chapter 3, but - as pointed out in Chapter 1.4 - the implementation of a coordinated approach by Jonker (2018) led to no significant increase in terminal productivity in the simulation models of TBA. Therefore this thesis addresses the on-line, uncoordinated scheduling problem of HTVs at the waterside of a perpendicular simulated container terminal. More specifically, we consider the scheduling of unload moves (from QCs to buffers located in the yard) and load moves (from buffers in the yard to QCs) on varying numbers of ShCs. In the isolated version of this scheduling problem it can be assumed that the RMGs always place the required container in the buffer before it is needed by a ShC and that container locations and destinations are pre-defined. Furthermore, QCs can only be productive when a slot is available in the QC TP (when the QC is discharging) or when the correct container is waiting for pick-up in the QC TP (when the QC is loading and needs to obey the vessel's bayplan). ShCs are responsible for both situations and therefore the scheduling decisions of ShCs have a direct effect on QC productivity and hereby container terminal productivity.

Chapter 3

Literature review

This chapter highlights previously done research in fields related to the topic of this thesis, divided into three parts: the first part in Section 3.1 concerns container terminal planning and scheduling for various equipment types. The effectiveness of simulation models is discussed, and uncoordinated as well as coordinated scheduling approaches are reviewed. Section 3.2 focusses on reinforcement learning algorithms that are used to improve logistic operations and the last part in Section 3.3 addresses the topic of parameter tuning: multiple methods and their applications are discussed.

3.1 Container terminal planning and scheduling

During the past decades, the literature on container terminals has grown substantially, simultaneously with the increasing importance of maritime container trade. Container terminal operators find themselves in a competitive market environment, where constant improvement is crucial for survival. This requires a thorough investigation of terminal requirements, constraints, operations and design, which in turn gives rise to a number of researchers that design, implement or optimize a wide variety of algorithms. Testing these algorithms on real terminals is very time-consuming, costly and sometimes even dangerous, which is why simulation techniques are often used.

3.1.1 Simulation

Carpenter and Ward (1990) are two of the first researchers to create a flexible model to reproduce most operations on a container terminal, including the effect of traffic interactions. This allows for thorough testing of important decisions in the terminal planning processes, and is used by various researchers and companies up until today. Gambardella et al. (1998) address the relevance of simulation, not only for the design of terminals before they are built, but also for operational decisions at container terminals. They point out that most scheduling decisions are usually solved by the terminal manager, only using his or her experience. Introducing new approaches becomes much more reliable when validation by means of a simulation model of the terminal is possible. Thus, the simulation tool also becomes a means at introducing new approaches into traditional settings. The simulation technique was constantly

improved over the years and enabled a number of researches to improve new and existing algorithms to increase terminal productivity and/or decrease costs: Said et al. (2014) achieve a 51% reduction in ship service time when compared to real data at El-Dekheilla port in Egypt and Leriche et al. (2016) reduce the cost of a new logistic system for Le Havre port in France by 16%.
Simulation models are extensively used to test the performance of newly created or adapted scheduling algorithms. Container terminal scheduling consists of four major components, that can be addressed simultaneously (coordinated) or separately (uncoordinated): Berth, QC, YC and HTV scheduling. We first discuss the uncoordinated approaches to solve these four scheduling challenges and then address the coordinated approaches to schedule one or more equipment types simultaneously.

3.1.2 Uncoordinated scheduling

The Berth Scheduling Problem (BSP) concerns the sequencing of vessels that are to be served at each berth, with the aim of minimizing the service time of vessels. Dulebenets (2018) discusses two evolutionary algorithms, based on the Genetic Algorithm (GA): one with a mutation operator that is adapted based on feedback from the search and one with a constant mutation operator. Both versions show promising results with an optimality gap that does not exceed 0.80%. The algorithm with an adapted mutation operator results in an average saving in terms of the total weighted vessel service cost of 5.4% and 8.5% for medium and large instances, respectively, without a significant increase in computational time. Kavoosi et al. (2019) elaborate on this evolutionary algorithm by introducing a Universal Island-based Meta-heuristic Algorithm (UIMA) that divides its population in four sub-populations (islands). Each island uses a different meta-heuristic algorithm to solve the BSP, which outperforms the UIMA approaches that uses the same algorithm on each island, and various single-solution-based meta-heuristic algorithms like Tabu Search and Simulated Annealing.
In the Quay Crane Scheduling Problem (QCSP) a sequence of handling jobs has to be assigned to a number of QCs with the goal of minimizing vessel handling times. The problem is shown to be NP-complete by Lee et al. (2008). Kim and Park (2004) implement a Branch and Bound algorithm (B&B) that is able to find optimal solutions and also perform a Greedy Randomized Adaptive Search Procedure (GRASP) that is computationally less exhaustive. The GRASP objective values do not exceed the B&B objective values by more than 10% and reduce the computational time to 3% of the B&B computational time for larger instances. Lee et al. (2008) propose an effective GA for the version of the QCSP where multiple QCs are allowed to work on one ship, but have to keep a distance of at least one ship-bay to prevent interference: the Quay Crane Scheduling with Non-Interference Constraints Problem (QCSNIP). They find near-optimal results with an average gap of 0.41% between the lower bounds and the solutions of GA.
YCs are responsible for container handling in the storage yard: they store, stack, relocate and retrieve containers in the storage yard and interchange containers by interacting with external trucks and HTVs. Due to their relatively slow operations they often form bottlenecks in terminals. Kim et al. (2003) formulate a problem that minimizes the service delay cost for arriving trucks and implement various sequencing methods: a modified Dynamic Programming (DP) approach, well-known

heuristic methods like first-come-first-served and shortest-processing-time and a RL approach that selects the most suitable sequencing rule at every decision point. Interestingly, the DP approach does not perform better than the straight-forward heuristics and the RL sequencing decisions outperform the heuristic methods when the container arrival locations do not follow a uniform distribution. Ng and Mak (2005) propose a B&B algorithm with improved lower and upper bounds that minimizes the sum of job waiting time and is able to solve the YC scheduling problem to optimality. According to Seyedalizadeh Ganji and Javanshir (2010) the B&B algorithm "has no sufficient efficiency to solve this models and becomes perfectly useless when the problem size increases." They therefore implement GA and compare its performance to B&B on small instances and find the same (optimal) solutions for both approaches with shorter run times for GA. GA is also applied on larger instances, but is not compared to B&B and therefore the solution quality remains unknown.
HTVs are the vehicles that transport containers to, from and within the yard and to/from different crane types. Generally, a large number of HTVs is driving around the container terminal and the scheduler aims to assign jobs to each of the vehicles with the objective of completing as many (useful) jobs as possible within a given time frame. Different terminal operators might have different expectations of HTV efficiency and productivity and an automated vehicle requires different constraints than manually operated HTVs. Therefore various objectives have been discussed in the literature, as well as a combination of those in what is called multi-objective optimization. Cai et al. (2013) focus on Autonomous Straddle Carriers (ASCs) and apply B&B combined with a column generation method to solve a weighted multi-objective function that consists of minimizing ASC travelling time, ASC waiting time and the finishing time of high-priority container-transferring jobs. The algorithm is able to solve small instances to optimality when minimizing the sum of weighted objectives, as well as when one objective at a time is considered. Hu et al. (2019) create clusters of containers and assign Terminal Trucks (TTs) to specific clusters by means of an adapted version of GA, where mutation and crossover operators are adapted based on the calculated fitness values. These fitness values are calculated based on a multi-objective function that consists of minimizing the non-loaded travelling distance and makespan of the TTs. Rashidi and Tsang (2011) define and formulate the scheduling problem of Automated Guided Vehicles (AGVs) as a Minimum Cost Flow (MCF) model and apply a state-of-the-art algorithm that is called the Network Simplex Plus Algorithm (NSA+). MCF models the scheduling problem as a connected network and NSA+ maintains a feasible spanning tree at each iteration until an optimal tree is found. Global optimal solutions are found for 3000 jobs and 10 million arcs within two minutes, which is significantly faster than the previously discussed versions of B&B. For larger instances or for when computational time is limited Rashidi and Tsang (2011) propose a Greedy Vehicle Search (GVS) method that assigns each job to a vehicle that minimizes a combination of waiting time, travelling time and job lateness. GVS is able to solve instances with 3000 jobs and 10 million arcs within five seconds, but with higher total costs than NSA+.

3.1.3 Coordinated scheduling

So far we have discussed various methods that are used to assign jobs to specific equipment types at container terminals. These methods can be applied sequentially in an uncoordinated manner, which breaks down the complete set of terminal operations into a series of decisions but ignores existing interrelations between different levels. One might therefore want to consider all terminal operations within one optimization problem in a coordinated manner, but in practice this would result in models that are "*much too huge to be solvable at all*" (Bierwirth and Meisel, 2010). Therefore, various researchers address the integration of a smaller number of operations at a time. When applied to scheduling operations we refer to this integration as coordinated scheduling, but different terms can be found in the literature: synchronized scheduling (Ahmed et al., 2021) or - more generally applicable - integrated optimization (Geoffrion, 1999). The latter states that the integration of multiple problems can either be done by function or by applying deep integration. In functional integration a top-level decision is made that determines the sequence for solving base-level subproblems. This decision is based on interaction between the top- and base-levels. Deep integration merges two or more subproblems into one problem formulation and solves the problem simultaneously. Integration of assignment and scheduling problems can be a very efficient use of deep integration, as was shown by Diabat and Theodorou (2014). They formulate a model that assigns QCs to vessels and sequences the QC operations simultaneously and point out that the integrated approach leads to a significant reduction in handling time, especially for large instances. GA is used to solve the integrated problem and near-optimal solutions are found. Safaeian et al. (2021) add multiple constraints to the same problem, resulting in a more real-life-based representation of a container terminal. They implement a Teaching-Learning-Based-Optimization (TLBO) algorithm that seems especially efficient for large instances.
Cao et al. (2010) integrate the QC and Yard Truck scheduling problems and formulate the combined problem as a mixed integer programming model with the objective of minimizing the maximum makespan of all jobs. They implement GA and a Johnson's Rule-based heuristic algorithm and compare the objectives with a computed lower bound. The largest gap between the objective values and the lower bound is 10% and 44% for GA and the Johnson's Rule-based heuristic, respectively. The same problem is studied by Skaf et al. (2021) and GA is compared with the actual performance of the port of Tripoli-Lebanon. GA reduces the makespan of processing all containers by 13-18% when compared to the real data on the considered instances.
To conclude, the scheduling problems of QCs, YCs and TTs are integrated by Zeng and Yang (2009), Chen et al. (2007) and Chen et al. (2013), leading to more efficient solutions but larger computational times. Zeng and Yang (2009) formulate a Hybrid Flow Shop (HFS) scheduling problem that minimizes the maximum completion time of all jobs. GA combined with two heuristics that sort jobs on shortest and longest processing time, respectively, is implemented and a connection with simulation is made to assess the performance. The authors use machine learning to avoid long computational times: a Neural Network (NN) is trained to compute expected objective values and filter out bad solutions to prevent the simulation model from taking too long to provide results on the solution quality. Chen et al. (2007) implement a Tabu Search (TS) algorithm and show that a good initial solution is important for the efficiency of the algorithm. Chen et al. (2013) improve on the solution quality of TS by implementing a three-stage algorithm that first generates crane schedules,

then solves a multiple-truck routing problem and ultimately uses a disjunctive graph to construct a complete solution. The three stages are iteratively performed, which facilitates the search for a good solution. The authors claim that the results are "*suitable for large real-world problems*" (Chen et al., 2013).

3.2 Reinforcement learning in logistic operations

Reinforcement learning (RL) is a machine learning method that is based on learning from trial-and-error. An agent is trained to take actions in specific states of the world and investigates which actions eventually generate the best results, by trying instead of by being told (Wuest et al., 2016). The method is originally based on a Markov Decision Process (MDP) and therefore used to be limited to processes that can be modelled by discrete-time MDPs. Bradtke and Duff (1994) extend this domain to Semi-Markov Decision Processes (SMDPs). In a SMDP the time spent in each state of the world can follow any continuous distribution instead of the exponential distribution that is considered in MDPs. This extension resulted in a rising interest in RL from researchers in different fields of work.
Fotuhi et al. (2013) model YCs as reinforcement agents with the goal of solving the yard crane scheduling problem. They define the current state of the world by a combination of 10 numeric values that consists of the location of the yard crane and detailed information about the number of trucks that is waiting in specific blocks. In each state the agent can choose one of four actions: stay idle, serve the closest truck, serve the longest waiting truck that is close to the Yard Crane or serve the longest waiting truck that has exceeded the specified waiting time threshold. The reward value for an action is defined by the average truck waiting time of all previously served trucks and therefore changes with every truck that is served. The results show a significant decrease in average truck waiting time when compared to a method in which the YC always chooses the next truck that maximizes a specific utility function. Hirashima et al. (2004) and Hirashima (2018) apply RL to container marshalling: the process that rearranges container placements on the yard. The number of container movements is decreased, when compared to the use of conventional methods.
RL also shows to be effective in determining near-optimal inventory policies (Giannoccaro and Pontrandolfo, 2002) and in scheduling decisions in the field of supply chain management (Stockheim et al., 2003). The latter only include two possible actions for the reinforcement agents: accept or reject the current job offer in a state that depends on the current queue, the value of the job offer and the penalty cost for delivering the job after its due date.
Production scheduling that translates detailed process plans into a shop floor schedule is one of the most important processes in manufacturing systems and especially the Job-Shop Problem (JSP) has attracted a lot of attention (Wang and Usher, 2005). Several RL approaches have been implemented to solve different versions of this scheduling problem: Wang and Usher (2005) have a reinforcement agent decide on the best dispatching rule for a single machine and Wang and Usher (2007) extend this model to two and three machines, while still using a single agent to decide on the best dispatching rule for all machines.
Csáji et al. (2006) create a multi-agent model for the dynamic JSP, in which the

number of jobs and the time of entering the system is not known in advance. The authors construct a model with two types of agents: order agents and resource agents. Order agents are associated with scheduling one specific job and they can communicate with other order agents to create a sequence of operations (consisting of one or multiple jobs). The resource agents gather information about the possible costs of executing that sequence and can then bring out a bet. The agent with the best bet gets the job and both the resource agent as well as the order agent are rewarded. Zhao et al. (2021) use a single-agent model for the same problem, but extend the state characteristics in such a way that it contains all information about average machine utilization, average machine work load rates, completion rates of jobs and remaining process waiting rates of jobs at specific points in time. The action set of the agent consists of nine different heuristics (including a random job pick) and the reward for each action reflects the processing urgency in order to minimize delay time of the jobs. By using average values of machine utilization and work load rates for the state specification the number of states is not as large as when taking into account these values per machine, which is what Zhang et al. (2012) do: they include eight state features - each consisting of different values for a specific job and/or machine - which results in $3m + 8n$ state features in total, where m is the number of machines and n the number of jobs. Increasing the number of state features has an increasing effect on the learning time of the reinforcement agent, but does yield state-specific reliable results (Zhang et al., 2012).
Most of the discussed models apply a Q-learning algorithm (Fotuhi et al. (2013), Hirashima (2018), Stockheim et al. (2003), Wang and Usher (2005), Wang and Usher (2007), Zhang et al. (2012)), that approximates the state- (s) and action- (a) dependent value $Q(s, a)$ directly and updates it at each time step. At every decision moment a probability ϵ determines whether a random action is chosen or the action that minimizes (or maximizes, depending on the implementation) the value of $Q(s, a)$ (Li et al., 2012).
Zhao et al. (2021) implement a Deep Q-Network (DQN) algorithm, which combines the Q-learning algorithm with a Deep Neural Network in order to handle high-dimensionality and continuity of the defined states: DQN is able to effectively fit the value of $Q(s, a)$, even when the number of states s is very large.

3.3 Parameter tuning

Each model and algorithm has its own parameters and since they all are completely different, there is no best known way to generally tune parameters. Just like there is no 'best algorithm' that can solve each optimization problem, as stated in various "no-free-lunch" theorems (Wolpert and Macready, 1997). Parameters do have a common property: each parameter can take on multiple values and a set of parameters therefore can have many different possible settings. We discuss a number of techniques that are used to deal with and/or reduce the search space of parameter settings.
The "one-factor-at-a-time" method has been considered for a long time as the only correct way to perform experiments (Bartz-Beielstein and Markon, 2004). This method varies the factors (parameters) one at a time, while keeping all other factors constant. This immediately excludes interactions between the factors, which is not a realistic assumption in most applications. Box et al. (1978) discuss a 2^k full factorial design approach, in which the experimenter picks two levels for each factor and per-

forms the required optimization method for each of the 2^k factor level combinations. Bartz-Beielstein and Markon (2004) state that 2^{k-p} fractional factorial designs are sufficient, when k is large and/or when computational resources are limited. Rules for constructing these fractional factorial designs are presented in Box et al. (1978). Bartz-Beielstein and Markon (2004) also propose a tree-based regression method in which each branch of the tree corresponds to the level of one factor. Each node of the tree denotes the average objective or fitness value of all factor level combinations that are possible in that part of the tree. The advantage of this method is that it allows the experimenter to test whether changing a specific factor has a significant effect on the fitness values.

In machine learning algorithms the tuning of hyperparameters that are used to control the learning process is of great practical importance. Bergstra and Bengio (2012) compare the effect on the learning process of using random search and grid search for parameter values. Random search is a method that selects a value for each hyperparameter independently by means of a specified probability distribution. In grid search one searches exhaustively through a manually specified subset of the hyperparameter space of the targeted algorithm. Both approaches evaluate the efficiency and/or result of the algorithm for each of the generated hyperparameter sets (Chan et al., 2013). Bergstra and Bengio (2012) find that - despite its simplicity - random search yields better results for the machine learning algorithms and claim that this result can mainly be ascribed to the fact that not all hyperparameters are equally important to tune: the effect of changing (some of) these parameters on the learning process of the machine learning algorithm might be negligible.

Sequential Parameter Optimization (SPO) is a search-method that adapts the parameter settings and fits a new model at every iteration. SPO starts with an initial population of parameter values that are tested to determine their utility or fitness. Then a model (this could be a regression model, but can be generalized to any other model) is fitted to represent the relation between the parameter settings and the results. Afterwards, new parameter values are generated (manually, randomly or by any other search method) and tested using this model. The most promising parameter values are added to the population and the algorithm starts again (Smit and Eiben, 2009). The authors show that any algorithm that is used to tune parameters of evolutionary algorithms outperforms both parameter setting conventions as well as individual intuition.

Alibrahim and Ludwig (2021) compare the performance of GA on the accuracy of a machine learning algorithm against grid search and a Bayesian algorithm. The Bayesian algorithm attempts to predict how untested combinations of parameter settings will perform and consists of two key components: the probabilistic surrogate model and the acquisition function. At each iteration the surrogate model is fitted to all current observations of the target function and then the acquisition function searches for possibly improving parameter settings (Alibrahim and Ludwig, 2021). GA starts with an initial population of parameter settings and selects the best individuals by assessing their fitness values. Crossover and mutation operators define the creation of new individuals, whose fitness values are also assessed. The authors find GA to be the fastest and most accurate algorithm among the three discussed algorithms.

A RL approach, where an agent is trained to decide on parameter values is proposed by Jomaa et al. (2019) for hyperparameters in machine learning algorithms and Semendiak (2020) for evolutionary algorithm parameters. Both approaches yield

promising results, outperforming each of the assessed conventional strategies.
Note that the effectiveness of the previously discussed algorithms is highly dependent on the problem at hand: if it is costly to calculate fitness values and/or objective values at every iteration fewer parameter settings can be tested within reasonable time. Furthermore, when the number of parameters and/or their possible values are large the algorithms' effectiveness might decrease. In those cases it is necessary to reduce the search space in a (more) efficient way. For this purpose Wang et al. (2014) propose the use of the Taguchi Method for parameter design. The Taguchi Method can be used to find the minimum number of experiments to be performed within the allowed limit of factors and levels and makes use of orthogonal array techniques to guide experiments with the goal of maximizing test convergence while minimizing the number of cases that have to be tested (Wang et al., 2014).

Chapter 4

(Semi-)Markov Decision Process

This chapter contains an introduction to Markov Decision Processes (MDPs) in Section 4.1, Semi-Markov Decision Processes (SMDPs) in Section 4.2 and Partially Observable Markov Decision Processes (POMDPs) in Section 4.3. The former is the main ingredient for many reinforcement learning algorithms, while SMDPs and POMDPs are generalizations of MDPs to which reinforcement learning algorithms have recently been applied as well. The main elements and assumptions are discussed in order to be applied in Chapter 5 where we introduce several reinforcement learning algorithms that lead to RLA: the reinforcement learning algorithm that we use to derive scheduling decisions for the ShCs on the simulated container terminal.

4.1 Markov Decision Process

Markov chains are an important class of discrete-time stochastic processes and are characterized by the Markov property that the future behaviour, given past and present behaviour, only depends on the present and not on the past. The Markov property is formally described in Definition 4.1.1 (van Handel, 2016).

Definition 4.1.1 (Markov Chain)
Let $\{X_n\}_{n\geq 0}$, be a random process where each X_n takes values in a finite set $\mathcal{D}$. $\{X_n\}_{n\geq 0}$ is called a Markov chain if the following holds:

$$\begin{aligned}&P\{X_{n+1} = x_{n+1} | X_n = x_n, ..., X_0 = x_0\}\\ =&P\{X_{n+1} = x_{n+1} | X_n = x_n\}\\ &\textit{for all } n \geq 0 \textit{ and } x_1, ..., x_{n+1} \in \mathcal{D}\end{aligned}$$

The continuous-time equivalent of a Markov chain is called a Markov process. The formal definition is given in Definition 4.1.2 (van Leeuwaarden, 2020).

Definition 4.1.2 (Markov Process)
A continuous-time stochastic process $\{X(t), t \geq 0\}$ with state space $\mathcal{S}$ is a Markov process if

$$\begin{aligned}&P\{X(t+s) = j | X(s) = i, X(u) = x(u), 0 \leq u < s\}\\ =&P\{X(t+s) = j | X(s) = i\}\\ &\textit{for all } s, t \geq 0 \textit{ and all } i, j, x(u) \in \mathcal{S}, 0 \leq u < s.\end{aligned}$$

The notion of a Markov Decision Process (MDP) was first introduced by Bellman (1957) in the form of a theory - the Markovian Decision Process - that could solve sequential decision problems numerically. He considers observations of a system over both finite and infinite time horizons that are split up into different stages. At each stage, the current state of the system is fully observed and a decision is to be made, which influences the state that is observed at the next decision stage. Note that it is not necessary that a specific decision in a specific state always leads to the same subsequent state. Transition probabilities $p(s'|s,a)$ define the probability of reaching state s' when taking decision a in state s for all s and s' in the state space and all possible decisions a. These transition probabilities also need to obey the Markov property, hence can only depend on the current state and decision and not on the previous states and/or decisions. Depending on the state and the decision that was made, a reward is gained. The formal definition - as proposed by Kaelbling et al. (1998) - of a MDP is given in Definition 4.1.3.

Definition 4.1.3 (Markov Decision Process)
The tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R})$ describes a Markov Decision Process with the following properties:

1. *State space $\mathcal{S}$: a finite, observable set of states of the world.*
2. *Action space $\mathcal{A}$: a finite set of decisions/actions. If not all actions are valid in every state of the world, we refer to the action space as $\mathcal{A}(s)$*
3. *Transition probability distribution $\mathcal{T}$: defines the probability of transitioning to a certain state, when the current state and action are known.*
4. *Reward function $\mathcal{R}$: the expected immediate reward, defined for every combination of action and state.*

Both the transition probability distribution and the reward function obey the Markov property as defined in Definition 4.1.1.

The total (cumulative) expected reward from the current stage until the end of the time horizon can be expressed by a value function. The functional equation expresses the relation between the value function at the present stage and the value function at the following stage. By backwards recursively maximizing the functional equation optimal decisions that depend on the stage and state can be determined. This method of optimizing a policy is based on the principle of optimality as defined by Bellman (1957) and stated in Definition 4.1.4. More detailed formulations are provided in Chapter 5.

Definition 4.1.4 (Principle of Optimality)
An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision.

The sequential approach that maximizes some form of reward over multiple stages of a decision process is very suitable for a wide variety of optimization problems, from robot motion control (Ding et al., 2012) to image processing (Petrov and Kharina,

2018) and portfolio allocation (Jilani et al., 2019). In the specific context of container terminal scheduling we can define a state space based on various attributes (the number of assignable vehicles, crane placement, the number and/or types of containers that are to be moved), depending on what is deemed important in the decision making. Scheduling decisions can consist of different (heuristic) methods that decide which container is to be handled next by which equipment type and rewards (QC productivity, number of performed jobs, idle time of equipment) ultimately determine which scheduling decision performs best in each state of the terminal. One could also address the different machines as separate entities and consider multiple MDPs simultaneously. In that case the state space would be based on machine-specific attributes (location, previous job, idle time), the scheduling decision defines the next container to be handled and rewards are machine-specific, as proposed and implemented by (Brauer and Weiss, 1998) for relatively small instances of a multi-machine scheduling problem.

4.2 Semi-Markov Decision Process

Depending on the definition of the state space a Markov Decision Process can be generalized to a Semi-Markov Decision Process (SMDP). Where MDPs only allow decision-making at predetermined discrete points in time, in SMDPs these decisions can be made at any time the system state changes (Giannoccaro and Pontrandolfo, 2002). Furthermore, in both MDPs and SMDPs state changes occur according to the Markov Property, but for MDPs the sojourn time in a state is a discrete random variable independent of the next state (van Leeuwaarden, 2020). For SMDPs the sojourn time in a state is a continuous random variable with a distribution that depends on both the current state as well as the next state. A formal definition, as proposed by Baykal-Gürsoy and Gürsoy (2007), is given in Definition 4.2.1.

Definition 4.2.1 (Semi-Markov Decision Process)
Consider the continuous-time stochastic process $\{X(t), t \geq 0\}$ *with values in the finite state space* $\mathcal{S}$. *Also consider* m *decision moments with* $m \in \mathbb{N}$.
We can then define the state process by $\{X_m, m \in \mathbb{N}\}$. *At each epoch* m *a decision* A_m *is chosen from the action space* $\mathcal{A}$.
We refer to the sojourn time between epochs (m-1) and m as the random variable Γ_m *and assume the initial state is fixed and given. The underlying sample-space* $\Omega = \{\mathcal{S} \times \mathcal{A} \times (0, \infty)\}^{\infty}$ *consists of all possible realizations of states, actions and transition times and will be equipped with the* σ*-field that is generated by the random variables* $\{X_m, A_m, \Gamma_{m+1} : m \geq 0\}$.
Let $\mathcal{P}_{xay}, x \in \mathcal{S}, a \in \mathcal{A}, y \in \mathcal{S}$ *define the probability measure for all policies* $\boldsymbol{u}$ *and all epochs* m *in the following way:*

$$\mathcal{P}_{\boldsymbol{u}}\{X_{m+1} = y | X_0, A_0, \Gamma_1, ..., X_m = x, A_m = a\} = \mathcal{P}_{xay}$$

Conditional on the event that the next state is y, Γ_{m+1} *has distribution function* $F_{xay}(.)$*:*

$$\mathcal{P}_{\boldsymbol{u}}\{\Gamma_{m+1} \leq t | X_0, A_0, \Gamma_1, ..., X_m = x, A_m = a, X_{m+1} = y\} = F_{xay}(t)$$

We assume that $F_{xay}(0) < 1$ and can then refer to the process $\{S_t, B_t : t \geq 0\}$ where S_t is the state at time t and B_t the decision that was made at time t as a Semi-Markov Decision Process.
Let $T_n = \sum_{m=1}^{n} \Gamma_m$ denote the time of the n-th transition. Then for $t \in [T_m, T_{m+1})$ we have:
$S_t = X_m$ and $B_t = A_m$.

4.3 Partially Observable Markov Decision Process

While MDPs mandate full observability of the environment in which the decisions are made, Partially Observable Markov Decision Processes (POMDPs) allow for partial observability (Kaelbling et al., 1998). In this setting, the decision maker makes an observation that does not fully capture the underlying environment. This underlying environment is a MDP, but the partial observations are not necessarily so. A formal definition, as proposed by Kaelbling et al. (1998) is given in definition 4.3.1.

Definition 4.3.1 (Partially Observable Markov Decision Process)
The tuple $(\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}, \Omega, \mathcal{O})$ describes a Partially Observable Markov Decision Process with the following properties:

1. *$\mathcal{S}, \mathcal{A}, \mathcal{T}, \mathcal{R}$: The underlying, not completely observed MDP.*
2. *Observations Ω: a finite set of observations that a decision maker can experience from the underlying environment. with $\mathcal{A}(s)$*
3. *Observation function $\mathcal{O}$: the probability distribution over possible observations for each action. We define $\mathcal{O}(s', a, o)$ as the probability of experiencing observation o when action a was taken and state s' was reached.*

Reinforcement learning (RL) is a machine learning method that evolves around training one or multiple agents to take actions that maximize some form of cumulative reward in specific states of the world and is therefore particularly suitable for solving MDPs and even SMDPs, as was shown by Bradtke and Duff (1994). Recent research involves the applicability of RL to POMDPs (Sutton and Barto (2018), Meng et al. (2021)). A major advantage of RL is its ability to handle complex (S)MDPs with large state and action spaces. These complex problems require huge computational effort when applying traditional approaches such as value iteration, policy iteration and linear programming (Giannoccaro and Pontrandolfo, 2002).

Chapter 5

Reinforcement learning

This chapter describes the general framework of reinforcement learning and Q-learning algorithms, that rely heavily on the concept of MDPs that was discussed in the previous chapter. Note that the goal of this thesis is to improve the scheduling algorithm of TBA. Providing a complete overview of all RL algorithms and the underlying theory - even though this is an interesting and continuously expanding research field - would be beyond its scope. This chapter focusses on the theoretical foundation of most RL algorithms and sequentially builds towards three explicit algorithms that were proposed by renowned researchers: Deep Q-Network with Experience Replay, Reward Backpropagation Prioritized Experience Replay and Double Deep Q-Network with Experience Replay. Section 5.1 provides a general overview of the main elements of reinforcement learning algorithms, Section 5.2 introduces the state-value and action-value functions and Section 5.3 discusses the general policy evaluation and iteration algorithm. Section 5.4 introduces the Q-value algorithm and Section 5.5 briefly discusses the main component of machine learning algorithms: the Artificial Neural Network (ANN). In Section 5.6 the methodology that is used to update the weights of this ANN is discussed and the last three sections of the chapter describe the algorithms that are combined into RLA: Section 5.7 introduces the Deep Q-Network with Experience Replay and Section 5.8 extends the notion of *experience replay* to Reward Backpropagation Prioritized Experience Replay. The chapter concludes with Section 5.9 by introducing the Double Deep Q-Network with Experience Replay.

5.1 General overview of reinforcement learning

The goal of a reinforcement learning algorithm is to train one or multiple agents in an environment that consists of a finite number or infinitely many states. We focus on the single-agent version of the algorithm, in which one agent makes decisions or performs specific actions and receives rewards. These rewards can be given after each decision moment, when the last state has been visited and/or at any moment in between.
Since RL is based on (S)MDPs it consists of the five main elements that were discussed in the previous chapter:

1. Decision epochs; moment in time when the agent makes a decision. These are

predetermined discrete time points for MDPs and continuous random variables for SMDPs.

2. State space $\mathcal{S}$; all possible states of the system. Can be infinitely large or of finite size, which highly affects computational time of the algorithm. Each state can be described by a vector with multiple entries, or be one-dimensional.

3. Decision/action space $\mathcal{A}$; all possible actions that can be taken by the agent in any state of the world.

4. Transition probabilities; the probability of transitioning from state s to state s', defined for all $s, s' \in \mathcal{S}$.

5. Rewards; positive or negative real numbers that define the quality of the decisions that were made. Rewards can be given after each decision epoch, at the end of the time horizon or at any time point in between. Rewards that are not given regularly are called sparse rewards.

5.2 State-value functions and action-value functions

The reinforcement agent is trained to determine a policy π: a mapping from states to probabilities of selecting each possible action in each of the states belonging to the state space $\mathcal{S}$. This policy generates a sequence of rewards $\{R_k\}$ and the goal of the agent is to learn the optimal policy π^* that maximizes a value function V of this sequence of rewards: the optimality criterion. The most commonly used optimality criterion is the maximization of the cumulative expected future discounted reward, that directly follows Bellman's recursive equation (Bellman, 1957). Inspired by Sutton and Barto (2018) we define the cumulative expected future discounted reward as in Definition 5.2.1.

Definition 5.2.1 (Cumulative expected future discounted reward)
Let $\{R_k\}$ be the sequence of rewards and $0 \leq \gamma \leq 1$ a discount rate. Then the cumulative expected future discounted reward at time t for all (infinitely many) time steps ahead is given by:

$$\begin{aligned} G_t &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... \\ &= \sum_{k=0}^{\infty} \gamma R_{t+k+1} && (5.1) \\ &= R_{t+1} + \gamma G_{t+1} && (5.2) \end{aligned}$$

By Definition 5.2.1 a reward that is received k time steps in the future is worth γ^{k-1} times its value when received at the current moment in time. As γ approaches 0 the agent is mostly concerned with maximizing immediate rewards, while for γ close to 1 future rewards are more valuable. Even though the cumulative expected future discounted reward G_t in Equation 5.1 sums infinitely many terms, for a bounded and nonzero reward sequence $\{R_k\}$ and $\gamma < 1$ it attains a finite value (Sutton and Barto, 2018). In order to find the optimal policy π^* the reinforcement agent needs to learn to estimate the value of being in state s and the value of taking action a in state s under policy π (Hare, 2019). In line with the formulation of Sutton and Barto (2018) we refer to the former as the state-value function ν_π (as given in Definition 5.2.2) and to the latter as the action-value function q_π (as defined in Definition 5.2.3). The value functions hence specify which actions and states are desirable in the long run.

Definition 5.2.2 (MDP State-value function for policy π)
Let s be the state at time step t, G_t be the expected discounted future reward and $0 \leq \gamma \leq 1$ be the discount rate. Furthermore, let $\{R_k\}$ be the sequence of future rewards and let $\mathbb{E}_\pi[\cdot]$ denote the expected value of the term in between the brackets given that the agent follows policy π. We can then define - for MDPs - the state-value function for policy π by:

$$\begin{aligned}\nu_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\ &= \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \Big| S_t = s\Big], \text{ for all } s \in \mathcal{S}\end{aligned}$$

Definition 5.2.3 (MDP Action-value function for policy π)
Let s, G_t, γ and $\{R_k\}$ be as defined in Definition 5.2.2. For MDPs, the action-value function for policy π is given by the expected value of the future discounted reward of taking action a in state s and subsequently following policy π and can be defined as:

$$\begin{aligned}q_\pi(s,a) &= \mathbb{E}_\pi[G_t | S_t = s, A_t = a] \\ &= \mathbb{E}_\pi\Big[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \Big| S_t = s, A_t = a\Big], \text{ for all } s \in \mathcal{S} \text{ and } a \in \mathcal{A}\end{aligned}$$

The state- and action-value functions can be estimated from the agents experience by exploiting the recursive relationship that is defined by Bellman's recursive equation Bellman (1957). We first define $p(s', r|s, a)$ as the probability of reaching state $s' \in \mathcal{S}$ and receiving reward $r \in \mathcal{R}$, given particular values of the previous state s and action a. Formally:

$$p(s', r|s, a) \equiv \mathcal{P}[S_{t+1} = s', R_t = r | S_t = s, A_t = a] \tag{5.3}$$
$$\forall s', s \in \mathcal{S}, r \in \mathcal{R}, a \in \mathcal{A}(s)$$
$$\sum_{s' \in \mathcal{S}} \sum_{r \in \mathcal{R}} p(s', r|s, a) = 1 \tag{5.4}$$
$$\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$$

Note that, even though the reward $r \in \mathcal{R}$ is received when state $s' \in \mathcal{S}$ is reached, it is commonly notated by $R_t = r$ instead of $R_{t+1} = r$. Furthermore, when some actions cannot be performed in specific states, the notation $\mathcal{A}(s)$ is used for the action space in state s. Since the reinforcement agent of our experiments can take any action from the action space in any state of the world, we will simply refer to the action space as $\mathcal{A}$ for all $s \in \mathcal{S}$. It is the tuple (s, a, s', r) that defines the transition from state $s \in \mathcal{S}$ to state $s' \in \mathcal{S}$ by taking action $a \in \mathcal{A}$ and receiving reward $r \in \mathcal{R}$.

Then the state-value function satisfies the following recursive relationship that is used in both reinforcement learning and dynamic programming:

$$\begin{aligned}
\nu_\pi(s) &= \mathbb{E}_\pi[G_t|S_t = s] && (5.5)\\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s] && (5.6)\\
&= \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} p(s', r|s, a)\Big[r + \gamma\mathbb{E}_\pi[G_{t+1}|S_{t+1} = s']\Big] && (5.7)\\
&= \sum_{a\in\mathcal{A}} \pi(a|s) \sum_{s'\in\mathcal{S}, r\in\mathcal{R}} p(s', r|s, a)\Big[r + \gamma\nu_\pi(s')\Big], \forall s \in \mathcal{S} && (5.8)
\end{aligned}$$

Equation 5.6 follows from Equation 5.5 by applying Equation 5.2 and Equation 5.7 follows from Equation 5.6 by conditioning on $S_{t+1} = s'$ and applying the Law of Iterated Expectations. Note that Equation 5.8 expresses the relationship between the value of being in state s and in the successor states s' and is called the Bellman self consistency equation for the value function, as proposed by Bellman (1957). Similarly, the action-value function $q_\pi(s, a)$ satisfies the self-consistency Equation 5.12.

$$\begin{aligned}
q_\pi(s, a) &= \mathbb{E}_\pi[G_t|S_t = s, A_t = a] && (5.9)\\
&= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1}|S_t = s, A_t = a] && (5.10)\\
&= \sum_{s'\in\mathcal{S}} \sum_{r\in\mathcal{R}} p(s', r|s, a)\Big[r + \gamma\mathbb{E}_\pi[G_{t+1}|S_{t+1} = s', A_{t+1} = a']\Big] && (5.11)\\
&= \sum_{s'\in\mathcal{S}, r\in\mathcal{R}} p(s', r|s, a)\Big[r + \gamma \sum_{a'\in\mathcal{A}} \pi(a'|s') q_\pi(s', a')\Big], \forall s \in \mathcal{S} && (5.12)
\end{aligned}$$

We can now define the policy that achieves the highest cumulative expected future discounted reward over the long run: the optimal policy π^*. Definition 5.2.4 states explicitly when one policy is considered better than another policy.

Definition 5.2.4 (Better policy π^*)
A policy π^* is better than or equal to a policy π if the cumulative expected future discounted reward is greater than or equal to that of π for all states $s \in \mathcal{S}$:

$$\pi^* \geq \pi \quad \Leftrightarrow \quad \nu_{\pi^*}(s) \geq \nu_\pi(s) \quad \forall s \in \mathcal{S}$$

The optimal policy π^* optimizes both the state-value function $v_\pi(s)$ as well as the action-value function $q_\pi(s, a)$, hence the following equations hold:

$$\begin{aligned}
v_{\pi^*}(s) &= \max_\pi[\nu_\pi(s)] && \forall s \in \mathcal{S} && (5.13)\\
q_{\pi^*}(s, a) &= \max_\pi[q_\pi(s, a)] && \forall s \in \mathcal{S}, a \in \mathcal{A} && (5.14)
\end{aligned}$$

Since the state-value function $\nu_{\pi^*}(s)$ maximizes the cumulative expected future discounted reward of being in state s under the optimal policy π^* and the action-value function $q_{\pi^*}(s, a)$ maximizes the cumulative expected future discounted reward of taking action a in state s under policy π^* the two value functions have to attain the same maximum value:

$$\begin{aligned}\nu_{\pi^*}(s) &= \max_{a\in\mathcal{A}} q_{\pi^*}(s,a) \\ &= \max_{a\in\mathcal{A}} \mathbb{E}_{\pi^*}[R_{t+1} + \gamma G_{t+1} | S_t = s, A_t = a] \end{aligned} \tag{5.15}$$

By using the relation between the state-value and action-value function we can derive the famous two forms of the Bellman optimality equation for ν_{π^*} and q_{π^*} as stated in Definition 5.2.5 and proposed by Bellman (1957). Note that - since it concerns the optimal value function - maximizing with respect to a automatically yields the optimal policy π^* and we could therefore omit the reference to a specific policy π^*. Intuitively, these equations state that the value of a state under an optimal policy is equal to the cumulative expected future discounted reward for the best action in that state.

Definition 5.2.5 (Bellman optimality equations for ν_{π^*} and q_{π^*})

$$\nu_{\pi^*}(s) = \max_{a\in\mathcal{A}} \mathbb{E}[R_{t+1} + \gamma \nu_{\pi^*}[(S_{t+1}) | S_t = s, A_t = a] \tag{5.16}$$

$$= \max_{a\in\mathcal{A}} \sum_{s',r} p(s',r|s,a)[r + \gamma \nu_{\pi^*}(s')] \tag{5.17}$$

$$q_{\pi^*}(s,a) = \mathbb{E}[R_{t+1} + \gamma \max_{a'} q_{\pi^*}(S_{t+1},a') | S_t = s, A_t = a] \tag{5.18}$$

$$= \sum_{s',r} p(s',r|s,a)[r + \gamma \max_{a'} q_{\pi^*}(s',a')] \tag{5.19}$$

For finite MDPs - with a finite number of $|\mathcal{S}|$ states in the state space, a finite number of actions and a finite set of rewards the Bellman optimality equation $\nu_{\pi^*}(s)$ consists of a system of $|\mathcal{S}|$ nonlinear equations and $|\mathcal{S}|$ unknowns. Even when the set of rewards is infinitely large, existence and uniqueness of solutions was proven by Rincón-Zapatero and C. (2003).
Once $\nu_{\pi^*}(s)$ is found one can determine the optimal policy π^* by assigning nonzero probabilities to the action(s) for which the maximum of ν_{π^*} is attained, looking one timestep ahead each time. Or one could first determine q_{π^*} to find the optimal actions without looking at possible successor states and values.

5.3 Policy evaluation and iteration

If not the entire state space $\mathcal{S}$ is known, or if there is a large number of states and/or actions, these computations can become time-consuming and costly, which is why approximations for the value functions are needed. The basis for reinforcement learning - and for dynamic programming - are iterative solution methods based on the self-consistency equation as defined in Equation 5.8. We first consider an approximative, iterative solution method to compute the state-value function ν_π for any policy π, also referred to as the *policy evaluation* or *prediction* problem. It starts with an initial approximation ν_0 (note that the terminal state, if it exists, attains value 0) that is chosen arbitrarily and each subsequent value for $\nu_{k+1}(s)$ is computed as in Equation 5.21, where k refers to the iteration number of the algorithm.

$$\nu_{k+1}(s) = \mathbb{E}_\pi[R_{t+1} + \gamma \nu_k(S_{t+1})|S_t = s] \tag{5.20}$$
$$= \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\Big[r + \gamma \nu_k(s')\Big] \quad \forall s \in \mathcal{S} \tag{5.21}$$

Combining the Bellman equation for ν_π (Equation 5.8) and Equation 5.21 shows that $\nu_k = \nu_\pi$ satisfies both equations. Generally, the sequence $\{\nu_k\}$ converges to ν_π as $k \to \infty$ (Sutton and Barto (2018)), as long as ν_π exists. The algorithm for estimating the state-value function ν is generally called *iterative policy evaluation*, since it evaluates policy π in an iterative manner over the state space $\mathcal{S}$, as denoted in the first part of the pseudocode in Algorithm 1.

We can use the approximated state-value function ν_π for policy π to find better policies (policies that yield higher values for ν than policy π). That is where the action-value function $q_\pi(s,a)$ is of use. Equations 5.22 and 5.23 state the action-value function $q_\pi(s,a)$ in terms of the state-value function of the subsequent state.

$$q_\pi(s,a) = \mathbb{E}[R_{t+1} + \gamma \nu_\pi(S_{t+1})|S_t = s, A_t = a] \tag{5.22}$$
$$= \sum_{s',r} p(s',r|s,a)\Big[r + \gamma \nu_\pi(s')\Big] \tag{5.23}$$

Suppose there exists a policy π', that is different from π only in the state s, so $\pi'(s) = a \neq \pi(s)$. We can then show that the new policy π' is better than the old policy π if and only if $q_\pi(s, \pi'(s)) > \nu_\pi(s))$ for that specific state s. The proof - a special case of the more general *policy improvement theorem*, as shown by Sutton and Barto (2018) - is given in Theorem 1.

Theorem 1 (Policy improvement theorem)
Let π and π' be two policies that only differ in the action chosen in state s^*. So for $s^* \in \mathcal{S}$: $\pi'(s^*) = a \neq \pi(s^*)$ and for $s' \in \mathcal{S}, s' \neq s$: $\pi'(s') = a = \pi(s')$.
For all states $s \in \mathcal{S}$ we have:

$$\nu_\pi(s) \leq q_\pi(s, \pi'(s)) \tag{5.24}$$
$$= \mathbb{E}[R_{t+1} + \gamma \nu_\pi(S_{t+1})|S_t = s, A_t = \pi'(s)] \tag{5.25}$$
$$= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \nu_\pi(S_{t+1})|S_t = s] \tag{5.26}$$
$$\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma q_\pi(S_{t+1}, \pi'(S_{t+1}))|S_t = s] \tag{5.27}$$
$$= \mathbb{E}_{\pi'}[R_{t+1} + \gamma \mathbb{E}[R_{t+2} + \gamma \nu_\pi(S_{t+2})|S_{t+1}, A_{t+1} = \pi'(S_{t+1})]|S_t = s] \tag{5.28}$$
$$= \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 \nu_\pi(S_{t+2})|S_t = s] \tag{5.29}$$
$$\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 \nu_\pi(S_{t+3}|S_t = s] \tag{5.30}$$
$$... \tag{5.31}$$
$$\leq \mathbb{E}_{\pi'}[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \gamma^3 R_{t+4} + ...|S_t = s] \tag{5.32}$$
$$= \nu_{\pi'}(s) \tag{5.33}$$

Here Equation 5.25 follows from Equation 5.23.
For all states $s' \in \mathcal{S}, s' \neq s^*$ we have $\nu'_\pi(S) = \nu_\pi(s) = q_\pi(s, \pi(s)) = q_\pi(s, \pi'(s))$ and hence the inequalities hold as equalities. For all states $s^* \in \mathcal{S}$ for which $\pi'(^*s) = a \neq \pi(^*s)$ strict inequalities hold, hence $\nu_\pi(s) < q_\pi(s, \pi'(s))$ and therefore $\nu_\pi(s) < \nu_{\pi'}(s)$ if policy π' is an improvement over policy π.

We can then find this new policy π' that is at least as good as the current policy π by finding the action that maximizes the state-value function ν at one timestep ahead, by applying Equation 5.36.

$$\pi'(s) = \text{argmax}_a q_\pi(s,a) \tag{5.34}$$
$$= \text{argmax}_a \mathbb{E}[R_{t+1} + \gamma \nu_\pi(S_{t+1}) | S_t = s, A_t = a] \tag{5.35}$$
$$= \text{argmax}_a \sum_{s',r} p(s',r|s,a)\Big[r + \gamma \nu_\pi(s')\Big] \tag{5.36}$$

Similarly as we did for the state-value function we can iteratively approximate the policy $\pi(s)$ by applying Equation 5.36 until no better policy is found. This process is called *policy iteration*. Combining the processes of *policy evaluation* and *policy iteration* as in Algorithm 1, following Sutton and Barto (2018), yields approximations for the optimal policy and state value functions of a MDP.

Algorithm 1: Policy iteration with iterative policy evaluation

```
Initialize $V(s) \in \mathbb{R}, \pi(s) \in \mathcal{A}$ arbitrarily for all $s \in \mathcal{S}$
Initialize discount factor $\gamma$
$V(\text{terminal}) \leftarrow 0$
Initialize small threshold $\theta > 0$ that determines accuracy of estimation
Policy Evaluation
$\Delta \leftarrow 0$
for $s \in \mathcal{S}$ do
    $\nu \leftarrow V(s)$
    $V(s) \leftarrow \sum_{s',r} p(s',r|s,\pi(s))[r + \gamma V(s')]$
    $\Delta \leftarrow \max(\Delta, |\nu - V(s)|)$
    until $\Delta < \theta$
Policy iteration
policy stable $\leftarrow$ true
for $s \in \mathcal{S}$ do
    old action $\leftarrow \pi(s)$
    $\pi(s) \leftarrow \text{argmax}_a \sum_{s',r} p(s',r|s,a)[r + \gamma V(s')]$
    if old action $\neq \pi(s)$ then
        policy stable $\leftarrow$ false
if policy stable then
    stop and return $V \approx \nu_\pi^*$and $\pi \approx \pi^*$
else
    go to Policy Evaluation
```

5.4 Q-learning

One of the drawbacks of the implementation of Algorithm 1 is the large number of operations required when the state space and/or action space is large. The need for methods that can learn without using a specific model of the environment gave rise to the implementation of Monte Carlo methods and *temporal-difference* learning to create experience-based learning methods instead of model-based learning methods. Sutton and Barto (2018) refer to the former as model-free methods, since exact initial knowledge of the underlying model and the transition probabilities is not needed but will be learned from experience. The concept of *temporal-difference* learning is based on Equation 5.20 and uses the updating rule in Equation 5.37 to update the state-value function after each transition, without the need to wait for the experiment to end. In order to distinguish between the (approximative) updating rules of RL and the previously discussed Bellman Equations, we will from here on denote the state-value function with a capital V and the action-value function by a capital Q, as opposed to the lowercase ν and q.

$$V(s) \leftarrow V(s) + \alpha\Big[r + \gamma V(s') - V(s)\Big] \tag{5.37}$$

In this equation α is a step size parameter ($\alpha \in (0, 1]$). This concept was used by Watkins (1989) when introducing *Q-learning*, one of the major early breakthroughs in reinforcement learning, and the step size parameter α is later referred to as the learning rate. In *Q-learning* the action-value function Q approximates the optimal action-value function q_{π^*} by means of the updating rule from *temporal-difference* learning applied on the action-value function. The pseudocode is given in Algorithm 2, following the definition as defined by Watkins (1989):

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big(Y^Q - Q(s,a)\Big) \tag{5.38}$$

$$Y^Q \equiv r + \gamma \max_{a'} Q(s', a') \tag{5.39}$$

Note that an episode is a sequence of transitions from the initial state until the terminal state. Generally, a large number of episodes is required to complete the Q-learning algorithm.

It is common practice to refer to the entire table with Q-values as the Q-table and to the values Y^Q as the target values. Note that the above updating rule, as well as the previously discussed action-value and state-value functions can be extended from MDPs to SMDPs, as was done by Bradtke and Duff (1994) in great detail. This increased the scope of reinforcement learning, since the discussed methods and algorithms could then also address SMDPs.

The Q-tables initially were manageable tables, but as data environments and the action and state spaces grew larger managing Q-tables became very inefficient and they are more and more often replaced by artificial neural networks (ANNs) that approximate these Q-values by using Stochastic Gradient Descent (SGD) to update the network weights.

Algorithm 2: Q-learning

```
Initialize learning rate $\alpha \in (0, 1]$
Initialize small $\epsilon > 0$
Initialize $Q(s, a) \forall s \in \mathcal{S}, a \in \mathcal{A}$ arbitrarily
$Q(terminal, a) \leftarrow 0 \quad \forall a \in \mathcal{A}$
for each episode do
    Initialize first state $s_1$
    for each transition do
        Apply $\epsilon$-greedy policy to choose action a using policy derived
          from Q
        Perform action $a$, observe reward $r$ and new state $s'$
        $Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$
        $s \leftarrow s'$
    Until $s'$ is a terminal state
```

5.5 Artificial Neural Networks

An ANN is a network of interconnected units or nodes that are called artificial neurons. Their properties are derived from the (non-artificial) neurons that form the main components of our nervous systems. The first computational model for ANNs was introduced by McCulloch and Pitts (1943), but Minsky and Papert (1969) pointed out that any useful application of neural networks would require higher computational power than available at that moment. This induced a brief stagnation in the research field of ANNs, but the field resurged nearly two decades later, when both computational power as well as internet growth yielded new opportunities (Mead and Ismail (1989), Schmidhuber (1992)).
Currently, ANNs in different forms are widely used for nonlinear function approximations and are particularly useful for machine learning algorithms like RL. Figure 5.1 depicts a generic ANN with four input units, two hidden layers and two output units. The black solid lines indicate links between the layers, each having a real-valued weight and the circles correspond to the artificial neurons. The arrows point from left to right, meaning that data only flows through the neural network in a feedforward manner and the output of one neuron cannot influence its input. Networks with one or multiple loops are called Recurrent Neural Networks (RNNs), but these will not be addressed in this thesis.
The example in Figure 5.1 - when applied in the context of RL - would require a four-dimensional state as an input and outputs the approximated action-value for two actions. In other words, when the input is the observed (four-dimensional) state $s \in \mathcal{S}$, two Q-values are predicted: $Q(s, a_1)$ and $Q(s, a_2)$, with actions a_1 and a_2 $\in \mathcal{A}$.
In order to make these predictions the artificial neurons compute a weighted sum of their input signals and then apply an activation function to produce their output. This activation function is a non-linear and typically S-shaped, sigmoid or a logistic function (Sutton and Barto, 2018).

The ANN is trained by adjusting the weights that correspond to each of the links in the network, with the goal of improving the performance of the network, measured

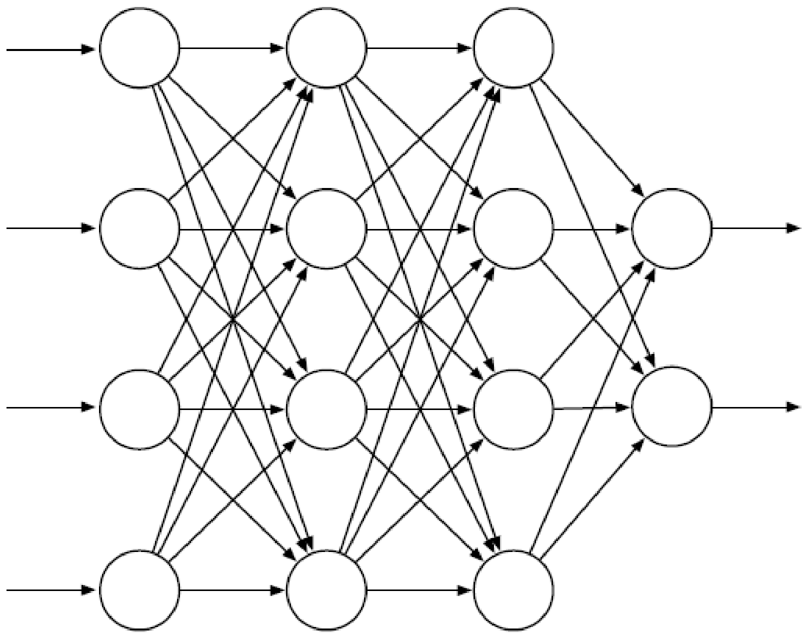


Figure 5.1: Generic feedforward ANN with four input units, two output units, and two hidden layers (Sutton and Barto, 2018).

by the minimization or maximization of a defined objective function. A commonly used objective function is the expected error or loss function, which can be optimized by Stochastic Gradient Descent (SGD). The partial derivatives of the objective function with respect to each of the weights of the network need to be calculated (or estimated). This is often done by alternating forward and backward passes through the ANN. First, a forward pass computes the activation of each unit and then a backward pass computes a partial derivative for each weight. This vector of partial derivatives is then used as estimate for the true gradient and thereby defines the direction of steepest descent of the objective function.

5.6 Deep Q-Network Stochastic Gradient Descent

In the context of RL an ANN with one or multiple hidden layers is referred to as a Deep Q-Network. Let us denote weights of the network by θ such that we can denote the Q-values that correspond to this Deep Q-Network (DQN) by $Q(s, a; \theta)$. Furthermore, in this thesis we consider algorithms with two Q-networks: an evaluation network $Q(s, a; \theta)$ and a target network $Q(s, a; \theta')$. The evaluation network is trained on the data, while the target network regularly updates the weights θ' in accordance with the weights of the evaluation network. These updating rules and algorithmic choices will be discussed in the next sections. This section focusses on the SGD algorithm that is used to train the evaluation network. By using the distinction between the evaluation and target network we can rewrite the updating rule from Equation 5.38 as follows:

$$Q(s,a;\theta) \leftarrow Q(s,a;\theta) + \alpha\Big(Y^{DQN} - Q(s,a;\theta)\Big) \tag{5.40}$$

$$Y^{DQN} \equiv r + \gamma \max_{a'} Q(s',a';\theta') \tag{5.41}$$

Notice that Equation 5.41 uses the next state to predict the maximum Q-value that can be acquired by using the target network, while Equation 5.40 adjusts the weights of the evaluation network by calculating the difference between Y^{DQN} and the Q-value that is predicted by the evaluation network. After training, we want the evaluation network to correctly predict the Q-values and therefore we want update the weights θ such that the term $(Y^{DQN} - Q(s,a;\theta))$ is minimized. A commonly used metric in RL algorithms is the mean squared error, which results in minimization of a loss function at every step i of the learning algorithm. Multiple transitions (s,a,s',r) are sampled from the reinforcement agent's history $\mathcal{D}$ (the specific sampling strategy is discussed in Section 5.8) in order to minimize the loss function in Equation 5.42.

$$\mathcal{L}_i(\theta_i) = \mathbb{E}_{s,a,s',r\sim\mathcal{D}}\Big(Y_t^{DQN} - Q(s,a;\theta_i)\Big)^2 \tag{5.42}$$

As discussed in Section 5.5 the vector of partial derivatives with respect to each weight is computed to minimize the objective: the loss function $\mathcal{L}_i(\theta_i)$ (Mnih et al., 2013). Notice that we can omit the constant (-2) in Equation 5.44.

$$\nabla_{\theta_i}\mathcal{L}_i(\theta_i) = \nabla_{\theta_i}\mathbb{E}_{s,a,s',r\sim\mathcal{D}}\Big(Y_t^{DQN} - Q(s,a;\theta_i)\Big)^2 \tag{5.43}$$

$$= \nabla_{\theta_i}\mathbb{E}_{s,a,s',r\sim\mathcal{D}}\Big(Y_t^{DQN} - Q(s,a;\theta_i)\Big)\nabla_{\theta_i}Q(s,a;\theta_i) \tag{5.44}$$

Most RL algorithms avoid computing the full expectations in the above gradient - as this can be computationally exhaustive - and instead optimise the loss function by SGD (Mnih et al., 2013). Several variations of SGD methods have been proposed, of which most can easily be implemented in most programming languages. A thorough comparison of the proposed methods is beyond the scope of this thesis, but the general SGD updating rule for the vector of weights θ of the evaluation network makes use of the step-size parameter α, as in Equation 5.40, and is given in Equation 5.45 (Sutton and Barto, 2018) for every weight update moment i.

$$Q(s,a;\theta_{i+1}) \leftarrow Q(s,a;\theta_i) + \alpha\Big(Y^{DQN} - Q(s,a;\theta_i)\Big)\nabla_{\theta_i}Q(s,a;\theta_i) \tag{5.45}$$

5.7 Deep Q-Network with Experience Replay

One of the issues that arise when using neural networks to predict the Q-values is correlation between consecutive observations. Since subsequent observations in many different environments are rarely independent and identically distributed, using the above updating rule on subsequent observations can lead to instability or even divergence, as was shown by Mnih et al. (2013) and Mnih et al. (2015). They address these issues by introducing a new reinforcement learning algorithm that is based on a Deep Q-Network (DQN).

The DQN algorithm, as proposed and implemented by Mnih et al. (2013), combines the target and evaluation neural networks with the biologically inspired mechanism termed *experience replay*. The use of *experience replay* in reinforcement learning was first introduced by Lin (1992) and involves remembering past experiences to increase the learning speed of the reinforcement agent and to prevent inefficient use of data. In the Q-learning algorithm experiences are used to update the neural network only once and are then forgotten. The DQN algorithm by Mnih et al. (2013) - as stated in Algorithm 3 - saves past experiences for a specified amount of time (or memory) and uniformly samples a batch of these observations to update the evaluation network $Q(s, a; \theta)$. This reduces the correlation incurred by only learning from subsequent observations and - as the authors state - *averages the behavior distribution over many of its previous states, smoothing out learning and avoiding oscillations or divergence in the parameters.* The target network with weights θ' copies the evaluation network every τ time steps of the algorithm. Only at $\tau = t$ the two Q-networks are equal: $Q(s, a; \theta) = Q(s, a; \theta')$. Periodically updating the Q-values that are used to calculate the target values reduces correlation between the Q-values in Equation 5.38 and the target values in Equation 5.39. Furthermore, updating the Q-values with a batch of observations at each time step allows the agent to learn faster as it processes more data in each learning step. Notice that it is not necessary to update the Q-values at every state transition: learning moments can but do not have to be performed after each transition. The uniform sampling method ascribes equal probabilities to sampling each of the transitions in the replay buffer, but depending on the problem more sophisticated sampling strategies can be implemented. These strategies emphasize transitions that are deemed more important or useful in training, as proposed by Moore and Atkeson (1993) and referred to as *prioritized sweeping*. This method assigns priority weights to each of the transitions according to their importance in the learning process and samples each transition with a probability proportional to this priority weight. Combining the notion of *prioritized sweeping* and *experience replay* results in an algorithm that Schaul et al. (2015) call Prioritized Experience Replay (PER).

5.8 Reward Backpropagation Prioritized Experience Replay

One of the possible frequently used metrics to define the importance of a transition in the learning process is (some function of) the error between the target values and the estimated Q-values. In a sparse reward environment, however, this error might not be a suitable candidate for the priority weights, as Zhong and Wang (2017) point out. Consider a sparse reward environment where approximately one in one hundred transitions yields a non-zero reward. Obviously, most can be learned from the one transition that yields a non-zero reward and the errors corresponding to the zero reward transitions may be small. Their sum, however, can become very large, which would lead to a higher probability of sampling a transition with a zero reward than sampling the one transition with the non-zero reward. Therefore Zhong and Wang (2017) introduce an algorithm that is specifically designed for environments where rewards are sparse and/or delayed. The *Reward Backpropagation Prioritized Experience Replay* algorithm also assigns priority weights to transitions, but gives transitions with non-zero rewards higher priority weights in the very beginning. Once

Algorithm 3: Deep Q-learning with Experience Replay

```
Initialize learning rate $\alpha$, mini-batch size $n$
Initialize replay memory $\mathcal{D}$ with capacity $N$
Initialize evaluation network $Q$ with random weights $\theta$ and target
  network $Q$ with weights $\theta'$
Initialize target network replacement frequency $\tau$
for each episode do
    Initialize state $s_1$
    for each transition do
        Draw $u \sim Unif(0, 1)$
        if $u < \epsilon$ then
            select random action $a$
        else
            select $a = \max_a Q(s, a; \theta_t)$
        Execute action $a$ and observe reward $r$ and state $s'$
        Store transition $(s, a, s', r)$ in $\mathcal{D}$
        Sample random mini-batch of size $n$ with transitions $(s, a, s', r)$
          from $\mathcal{D}$
        for j=1,...,n do
            if $s'_j = terminal$ then
                Set $y_j = r_j$
            else
                set $y_j = r_j + \gamma \max_{a'} Q(s'_j, a'; \theta')$
        Perform a gradient descent step by minimizing the loss
          $\mathcal{L}(\theta) = \frac{1}{n}\sum_{j=1}^{n}(y_j - Q(s_j, a_j; \theta))^2$
        Replace target network parameters $\theta'$ with $\theta$ every $\tau$ timesteps
```

they have been sampled, these priority weights are propagated backwards to the previous transitions. That way, the entire learning process is repeated in a backward manner. Two additional parameters are introduced: a priority weight β that is assigned to a transition with a non-zero reward at the moment it is received and a priority decay rate λ that defines the importance of the transition after every round of backpropagation. Each observation that enters the replay memory is assigned a priority weight p. If the transition is a terminal transition or a transition with non-zero reward $p = \beta$, where $\beta \ggg 1$ and otherwise $p = 1$. At every learning step a batch of transitions is sampled with probabilities proportional to their priority weights and once a transition that has priority weight $p > 1$ is sampled, that weight is backpropagated to the previous transition in the memory. The current transition then gets a normal priority of 1 until the backpropagation procedure reaches the previous non-zero reward. In that case, one round of backpropagation is completed and the first transition with a non-zero reward receives a priority weight of $\lambda^i\beta$, where i is the number of backpropagation rounds that have been completed. A simplified schematic overview is provided in Figure 5.2, for $\beta = 100$, $\lambda = 0.1$, a memory capacity of 7 and a mini-batch size of 3.

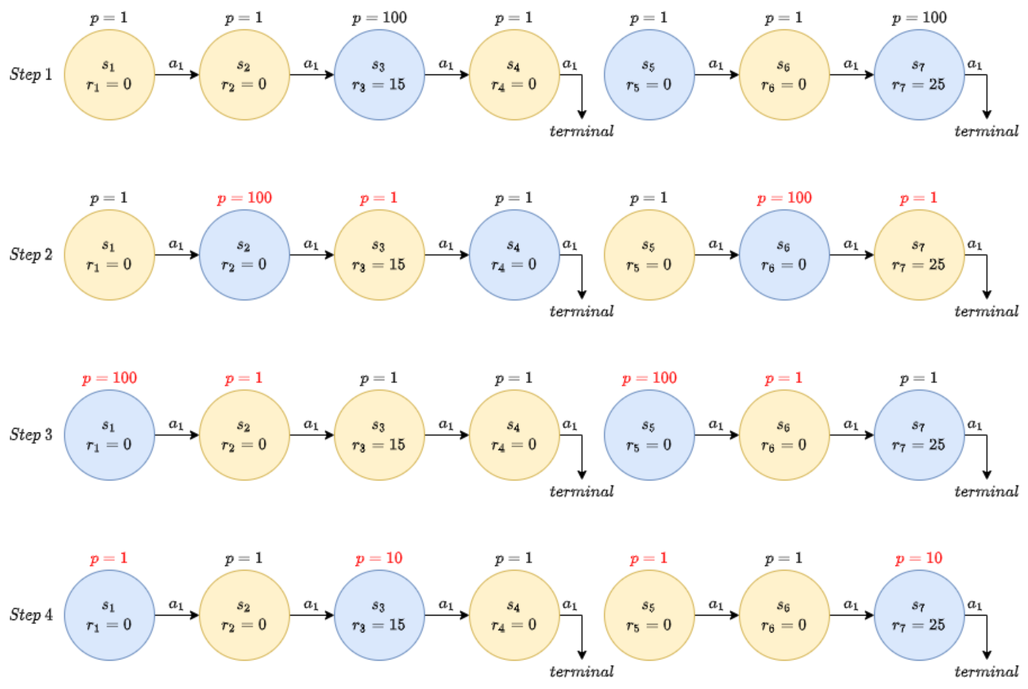


Figure 5.2: Reward Backpropagation Prioritized Experience Replay with $\beta = 100$, $\lambda = 0.1$, a memory capacity of 7 and mini-batches of size 3. In this illustration, no new transitions are added after step 1. Blue transitions are sampled in each step and red denotes a change in priority weight p. Changes occur when a transition with higher priority was sampled in the previous step and at every round of backpropagation the priority weight p is multiplied by λ.

Zhong and Wang (2017) claim that a good performance is reached by setting β equal to the ratio of the non-zero to the zero reward transition frequency (i.e. 100 when 1 in 100 transitions yields a non-zero reward) and that λ should be chosen such that each transition is included in 2 or 3 rounds of backpropagation before its priority weight reaches value 1.0. The pseudocode for the RBPER is given in Algorithm 4.

Algorithm 4: Reward Backpropagation Prioritized Experience Replay

```
Initialize replay memory $\mathcal{D}$ with capacity $N$
Initialize evaluation network $Q$ with random weights $\theta$ and target
  network $Q$ with weights $\theta'$
Initialize target network replacement frequency $\tau$
Initialize non-zero reward priority weight $\beta$ and priority decay rate $\lambda$
for each episode do
    Initialize state $s_1$
    for each transition do
        Draw $u \sim Unif(0,1)$
        if $u < \epsilon$ then
            select random action $a$
        else
            select $a = \max_a Q(s,a;\theta)$
        Execute action $a$ and observe reward $r$ and state $s'$
        if $r \neq 0$ or $s'$ is terminal then
            $p = \beta$
        else
            p = 1
        Store transition $(s,a,s',r,p)$ in $\mathcal{D}$
        Sample mini-batch of size $n$ with transitions $(s,a,s',r,p)$ with
          probabilities proportional to $p$ from $\mathcal{D}$
        for j=1,...,n do
            $(s,a,s',r,p)_{j-1} \leftarrow$ transition prior to transition $j$ in replay
              memory $\mathcal{D}$
            if $s'_j = terminal$ then
                Set $y_j = r_j$
            else
                set $y_j = r_j + \gamma \max_{a'} Q(s',a';\theta')$
            if $r_{j-1} = 0$ and $p_j > 1$ then
                $p_{j-1} \leftarrow p_j$ in replay memory $\mathcal{D}$
            else if $r_{j-1} \neq 0$ and $p_j > 1$ or $s'_j = terminal$ and $p_j > 1$
              then
                $(s,a,s',r,p)_k \leftarrow$ first transition after $(s,a,s',r,p)_j$ in $\mathcal{D}$
                  with $s'_k$ =terminal or $r_k \neq 0$
                $p_k \leftarrow \max(\lambda p_j, 1)$ in replay memory $\mathcal{D}$
            $p_j \leftarrow 1$ in replay memory $\mathcal{D}$
        Perform a gradient descent step by minimizing the loss
          $\mathcal{L}(\theta) = \frac{1}{n}\sum_{j=1}^{n}(y_j - Q(s_j,a_j;\theta))^2$
        Replace target network parameters $\theta'$ with $\theta$ every $\tau$ timesteps
```

5.9 Double DQN

One of the pitfalls of applying Equation 5.39 is caused by the overestimation of action values, due to a positive bias that is also referred to as the *Optimizer's Curse*, as proven by Smith and Winkler (2006). They state that decision makers that consistently choose alternatives that seem best based on their estimated values should expect to be disappointed in the long run. In other words: uncertainty in value estimates for picking the best action is positively biased. A proof - as proposed by Smith and Winkler (2006) is provided in Theorem 2.

Theorem 2 (The Optimizer's Curse)
Let $V_1, ..., V_n$ be estimates of $\mu_1, ..., \mu_n$ such that $E[V_i|\mu_1, ..., \mu_n] = \mu_i, \forall i$. Hence the estimates are conditionally unbiased. Let i^* denote the alternative with the maximal estimated value $V_i^* = \max[V_1, ..., V_n]$. Then

$$E[\mu_{i^*} - V_{i^*}] \leq 0$$

Furthermore, if there is a chance of selecting the "wrong" alternative: μ_{i^*} is not the maximum of $\mu_1, ..., \mu_n$ then

$$E[\mu_{i^*} - V_{i^*}] < 0$$

Proof *First consider a set of true values $\mu = \mu_1, ..., \mu_n$ with uncertain value estimates $V = V_1, ..., V_n$. Let j^* be the alternative with the maximum true value: $\mu_{j^*} = \max[\mu_1, ..., \mu_n]$. Since μ is fixed and V uncertain, j^* is a constant and i^* a random variable. For any V we have:*

$$\mu_{i^*} - V_{i^*} \leq \mu_{j^*} - V_{i^*} \leq \mu_{j^*} - V_{j^*} \tag{5.46}$$

The first inequality follows from the definition of j^* and the second from the definition of i^*. If we then take expectations and condition on μ we get:

$$E[\mu_{i^*} - V_{i^*}|\mu] \leq E[\mu_{j^*} - V_{j^*}|\mu] = 0 \tag{5.47}$$

The last equality follows from the assumption that the value estimates are conditionally unbiased. Since $E[\mu_{i^*} - V_{i^*}|\mu] \leq 0$ for all μ, integrating over the uncertain μ yields $E[\mu_{i^*} - V_{i^*}] \leq 0$. If $i^* = j^*$ with probability one all inequalities become equalities and $E[\mu_{i^*} - V_{i^*}] = 0$, but if there is some chance that $i^* \neq j^*$ the equalities will be strict and $E[\mu_{i^*} - V_{i^*}] < 0$

van Hasselt (2010) applies this proof to the situation of the Q-learning algorithm. The maximization operator in the target value uses the same values to select as well as evaluate an action, based on an approximation that is made by the Q-learning algorithm. This becomes clear when we rewrite Equation 5.41 as follows:

$$Y^{DQN} = r + \gamma Q(s', \text{argmax}_{a'} Q(s', a'; \theta'); \theta')$$

The author states that this approximation leads to overoptimistic value estimates and proposes to decouple the selection and evaluation procedure with an extra Q network:

$$Y^{DoubleQ} \equiv r + \gamma Q(s', \text{argmax}_{a'} Q(s', a'; \theta); \theta')$$

Note that in the initially proposed Double-Q algorithm the distinction between an evaluation and target network as in the DQN algorithm was not used. Only in a later article (van Hasselt et al., 2016) the author and his colleagues combine the two, since they claim that the evaluation network is perfectly suitable to replace the extra Q-network that was proposed in van Hasselt (2010). The combined algorithm is called Double DQN and provides improved results when compared to the DQN algorithm that was proposed by Mnih et al. (2013), without the need for any additional networks or parameters. The improved updating rule is then as follows:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\Big(Y^{DoubleDQN} - Q(s,a;\theta)\Big) \tag{5.48}$$

$$Y^{DoubleDQN} \equiv r + \gamma Q(s', \text{argmax}_{a'} Q(s', a'; \theta); \theta') \tag{5.49}$$

The only difference with the DQN algorithm is that the evaluation network with weights θ is used for the action selection with the *argmax* operator. Still, the target network is used to estimate the target value. The pseudocode for the algorithm is given in Algorithm 5. Note that this algorithm does not make use of the RBPER algorithm that was discussed in Section 5.8, but randomly samples the mini-batches from the replay memory, as described in Section 5.7 and proposed by Mnih et al. (2013). The algorithms are combined in the next chapter, when we introduce the methodology that is used for RLA.

Algorithm 5: Double DQN with Experience Replay

```
Initialize replay memory $\mathcal{D}$ with capacity $N$
Initialize evaluation network Q with random weights $\theta$ and target
  network $Q$ weights $\theta'$
Initialize target network replacement frequency $\tau$
for each episode do
    Initialize state $s_1$
    for each transition do
        Draw $u \sim Unif(0,1)$ if $u < \epsilon$ then
            select random action $a$
        else
            select $a = \max_a Q(s,a;\theta)$
        Execute action $a$ and observe reward $r$ and state $s'$
        Store transition $(s, a, s', r)$ in $\mathbb{D}$
        Sample random mini-batch of size $n$ with transitions $(s, a, s', r)$
          from $\mathcal{D}$
        for j=1,...,n do
            Define $a^{max}(s'_j;\theta) = \text{argmax}_{a'} Q(s'_j, a'; \theta)$
            if $s'_j = terminal$ then
                Set $y_j = r_j$
            else
                set $y_j = r_j + \gamma Q(s'_j, a^{max}(s'_j;\theta); \theta')$
        Perform a gradient descent step by minimizing the loss
          $\mathcal{L}(\theta) = \frac{1}{n}\sum_{j=1}^{n}(y_j - Q(s_j, a_j; \theta))^2$
        Replace target network parameters $\theta'$ with $\theta$ every $\tau$ timesteps
```

Chapter 6

RLA for container terminal scheduling

In this section we present a detailed description of the reinforcement learning algorithm that we have implemented in order to improve TBA's scheduling algorithm for ShCs on a simulated container terminal. The algorithm is based on a combination of the previously discussed algorithms, but has been adapted to fit the container terminal environment and characteristics. Modelling and algorithmic choices are discussed in this chapter and the results are presented in Chapter 8. Section 6.1 and Section 6.2 introduce the state and action space, respectively. Section 6.3 discusses the reward function and Section 6.4 places the created algorithm in the (PO)MDP framework. Section 6.5 provides a detailed description of RLA.

6.1 State characteristics

Consultation of several simulation experts at TBA led to the establishment of a number of terminal situations that might require a different parameter setting of the current scheduling algorithm. We are specifically interested in the cumulative expected future discounted reward of taking scheduling actions in these situations and have therefore created a state space with 4 different attributes, consisting of 144 states. The following parameters are used to define the state space:

- *PLoad*: Percentage of the total number of QCs that is currently performing (twinlift) loading operations.
- *PBusy*: Percentage of containers that have to be moved to or from a busy module or QC. A module or QC is defined as busy when 4 or more jobs have that location as their origin or destination. We keep track of all m containers that correspond to a job that can currently be assigned to a ShC. Then, for each module and buffer, we sum the total number of containers that have this location as their origin or destination. Let L_i be the number of containers that have location i as their origin or destination for $i = 1, ..., n$, where n is the total number of modules and QCs on the container terminal. Then $\sum_{i=1}^{n} L_i \mathbb{1}_{L_i \geq 4}$ is the total number of containers that are to be moved to or from a busy module or buffer and $PBusy = \frac{\sum_{i=1}^{n} L_i \mathbb{1}_{L_i \geq 4}}{m}$.

- *AssJobs*: Number of currently assignable jobs. These jobs - as described in Section 1.2 - consist of a specific container (with a unique number), its current location (origin) and the location it has to be delivered to (destination). Notice that unloading jobs (jobs that have to be picked up at a QC and moved to the yard) only receive their destination when they are picked up at a QC, since they always pick the first container that is ready. This way, a vehicle at a discharging QC never has to wait for another vehicle that was assigned first, but had to drive longer.

- *NumV*: Number of currently assignable (idle) vehicles. A vehicle is idle as soon as it has dropped off the container of its previous job at the specified destination. When no new job is assigned immediately, the vehicle will go to a parking place somewhere on the container terminal until a new job is assigned. On its way to this parking and when parked the vehicle remains assignable.

- *Cl*: Number of QC clusters. Two QCs are in the same cluster when the distance between them is smaller than twice the cranewidth (36 meters).

- *NumQC*: Number of QCs that is currently active on the container terminal.

	Attribute 1	**Attribute 2**	**Attribute 3**	**Attribute 4**
1	$0 \leq PLoad \leq 25$	$0 \leq PBusy \leq 25$	$\frac{AssJobs}{NumV} \leq NumV$	$Cl > 0.8NumQC$
2	$25 < PLoad \leq 50$	$25 < PBusy \leq 50$	$NumV < \frac{AssJobs}{NumV} \leq 2.5NumV$	$0.5NumQC \leq Cl < 0.8NumQC$
3	$50 < PLoad \leq 75$	$50 < PBusy \leq 75$	$\frac{AssJobs}{NumV} > 2.5NumV$	$Cl < 0.5NumQC$
4	$75 < PLoad \leq 100$	$75 < PBusy \leq 100$		

Table 6.1: RLA state attributes and values.

Table 6.1 defines the values of each of the states of the terminal. Due to their definitions the states never overlap and there is zero probability that a non-defined state is visited during the container terminal simulation and learning process. We abbreviate Attribute by A when we refer to a state description. For example, state $[A_1 = 4, A_2 = 3, A_3 = 1, A_4 = 2]$ is the state in which:

- $75 - 100\%$ of the total number of QCs is currently (twinlift) loading. That means that $0 - 25\%$ of the total number of QCs is currently (twinlift) unloading.

- $50 - 75\%$ of the containers that currently have to be moved share an origin or destination with at least 3 other containers that have to be moved.

- The ratio of the number of assignable jobs over the number of assignable vehicles is smaller than the total number of assignable vehicles. In other words: there are more vehicles than needed at this moment.

- If there are 8 active QCs there at least 4 and at most 6 clusters.

6.2 Action set and selection

Every time a vehicle becomes idle a new action in the form of a scheduling decision is required. The assignable jobs and the current state of the terminal are sent to the reinforcement agent. When the state did not change since the last scheduling decision was made, the same action is used to assign the next job to the idle vehicle. In case the state did change, a (possibly) different action is picked by applying an ϵ-greedy policy. A random number u between 0 and 1 is drawn from the uniform distribution function. If $u < \epsilon$ a random action is chosen from the action space and if $u > \epsilon$ the action that yields the highest cumulative expected future discounted reward $a = \text{argmax}_a Q(s, a; \theta)$ is selected. The value for ϵ decreases as the algorithm proceeds and thereby allows for more exploration in the beginning and exploitation in the end.
The action space consists of 14 actions, each based on a specific ordering of the currently assignable jobs, matched with the currently assignable vehicles. Depending on the number of idle vehicles one or more jobs are then dispatched. We briefly discuss the parameters that are used to define the scheduling heuristics:

- Duetime is the time at which the container is planned at its destination. There is no real penalty when this duetime is not exactly met, but once multiple containers arrive late congestion or delay of vessels might occur.

- Driving distance is the distance in meters that the vehicle has to drive from its current location to the origin of the job. The job itself also requires the vehicle to drive a certain distance, but this is not included in the parameter as it is exactly equal for each of the vehicles.

- *MaxToQC* and *MaxFromQC* are parameters that define how many vehicles are allowed to drive to and from QCs, respectively. By respecting *MaxToQC* we mean that no more than the specified amount of vehicles can drive to or from QCs at the same time. No *MaxToQC* means that more than the specified amount of vehicles can drive to or from QCs at the same time. This is included in the action space in order to determine whether or not including this parameter is desirable in each of the states of the state space.

- The busiest buffer is the (module or QC) buffer that is the origin or destination for the largest number of containers that currently need to be moved, out of all modules or buffers at the terminal, as defined in Section 6.1 by L_i for $i = 1, ..., n$. The busiest buffer hence maximizes L_i.

- The quietest buffer is the (module or QC) buffer that is the origin or destination for the smallest number of containers that currently need to be moved and therefore minimizes L_i.

- Load moves are containers that need to be loaded onto a vessel, and therefore have their origin in the storage yard and their destination at a QC;

- Sequence numbers define the order in which containers will be placed onto a vessel. Loading jobs (where containers have to be picked up at a buffer on the yard and brought to a QC TP) always have a sequence number. Since a number of containers can be placed near a QC at the same time, it is not vital nor necessary that they are brought to the QC in exactly that sequence. It can, however, cause problems when multiple containers with higher sequence numbers are delivered before the one with a lower sequence number. Therefore, for all the actions in the action space, after ordering according to the specified method, the jobs are ordered according to sequence number.

The actions are defined below.

1. *ESR*: Earliest duetime, shortest driving distance, respect *MaxToQC*.
2. *ESN*. Earliest duetime, shortest driving distance, no *MaxToQC*.
3. *SER*. Shortest driving distance, earliest duetime, respect *MaxToQC*.
4. *SEN*. Shortest driving distance, earliest duetime, no *MaxToQC*.
5. *BBR*. Busiest buffer, shortest driving distance, respect *MaxToQC*.
6. *BBN*. Busiest buffer, shortest driving distance, no *MaxToQC*.
7. *QBR*. Quietest buffer, shortest driving distance, respect *MaxToQC*.
8. *QBN*. Quietest buffer, shortest driving distance, no *MaxToQC*.
9. *LSS*. Load moves, sequence number, shortest driving distance, respect *MaxToQC*.
10. *LSR*. Loading* QC first, shortest driving distance, respect *MaxToQC*.
11. *LSN*. Loading* QC first, shortest driving distance, no *MaxToQC*.
12. *USR*. Unloading* QC first, shortest driving distance, respect *MaxToQC*.
13. *USN*. Unloading* QC first, shortest driving distance, no *MaxToQC*.
14. *Random*: Pick a random job from the list.

* For these actions we first order the loading and unloading QCs in increasing order of the currently assigned vehicles. Therefore, the jobs to or from QCs with the smallest number of currently assigned vehicles are assigned first.

6.3 Reward function

As discussed in Chapter 1, the main goal of adapting the scheduling algorithm of TBA is to increase QC productivity. This therefore has to be included in the reward function, but after performing a specific job the effect on this productivity is not immediately visible. Containers first have to pile up before a discharging QC becomes idle and the piles have to be eliminated before it can become productive again (and vice versa for loading QCs). The rewards are therefore only given once per simulation hour. Since - generally - multiple state changes have taken place within this hour we are dealing with sparse rewards. State transitions in between this hourly reward receive a reward of 0. At every simulation hour - except for the first one, since then most processes are still starting up - the QC productivity of that preceding hour is calculated. For each of the functions (loading, discharging,

twinlift unloading and twinlift discharging) the number of boxes that was handled and the time spent performing that functionality is saved. As it proved effective to also incur negative rewards we then compare this QC productivity with a base case: a scenario with exactly the same terminal settings and the scheduling algorithm of TBA. We subtract 80% of the QC productivity for each of the functions separately, taking into account the time a QC has spent performing that functionality. We train the reinforcement on scenarios that differ in the placement of the QCs and in the number of ShCs that are working on the terminal and therefore also distinguish between these scenarios in the definition of the reward function. More ShCs on a terminal generally lead to higher QC productivities, hence without making this distinction the reinforcement agent would think his moves are worse or better than they actually are. In addition, we do not want to specify the exact integer number of ShCs in the state space in order to provide results that are as generally applicable as possible, with a reasonable amount of states for the final algorithm. By comparing the received reward with the base case of every scenario, we can avoid the need to do so.
Let B_{ij} be the average number of moves of functionality i that is performed in one hour by one QC in the base case with scenario j, let RL_{ik} be the total number of moves that is performed in the last hour by QC k with functionality i and let T_{ik} be the fraction of the last hour QC k has spent performing functionality i during the last hour. We can then define the hourly reward for scenario j as

$$Reward_j = \sum_{i=1}^{4} \sum_{k=1}^{NumQC} RL_{ik} - 0.8 * \sum_{i=1}^{4} \sum_{k=1}^{NumQC} T_{ik} B_{ij}$$

6.4 MDP framework

The previously discussed definitions of the state and action space and reward function require additional explanation in order to be placed in the MDP - and therefore - RL framework. It should be noted that the true underlying environment - the container terminal simulation program TIMESQUARE - contains a large number of complex processes that are not all of interest for the goal of this thesis. These processes all follow a certain distribution function which results in - among many other uncertainties - different truck arrival times at the landside of the terminal for each experiment. Including each of these processes and all uncertainties in the state space would result in an infinitely large state space and would prevent us from extracting useful scheduling decisions from the RLA output. These processes do, however, influence the state transitions of our model. We could therefore address the problem as a POMDP, where our agent only observes the states and rewards, by means of communication with the simulation program. The complete container terminal environment could be seen as a MDP (as was done in different forms by Fotuhi et al. (2013) and Hirashima (2018)). Even though this area is still being extensively researched and the authors also propose a novel automata algorithm, Gaon and Brafman (2019) claim that the use of a replay memory combined with DQN might still yield desirable results for the POMDP, since the external memory *essentially learns an extended state representation* and hereby learns information about the unobserved parts of the environment.
Additionally, following the formal definition as in Definition 4.1.3, the reward function needs to obey the Markov property as well. In this setting - as in many real-

world settings - this is not the case. Each state-action combination in the simulation hour preceding the reward is responsible for the QC productivity in that hour and we could therefore refer to the reward function as non-Markovian. This does not necessarily mean that RL is not a proper solution-method for the problem. Many RL models with sparse or delayed rewards actually include non-Markovian reward functions. In fact, a delayed reward is by definition non-Markovian, since it is not directly attached to the transition that is mainly responsible for the reward. Walsh et al. (2007) address this issue in various ways and state that *treating the delayed problem as a regular MDP and using the memoryless policy can produce reasonable policies, especially if the delay is relatively small compared to the magnitude of the state transitions.* We furthermore claim that repeatedly sampling the full trajectories will help our agent to remember which actions ultimately resulted in higher rewards and therefore RLA could lead to desired results in the form of a Q-table that predicts the action that yields the highest cumulative expected future discounted reward in each of the defined states of the terminal. We therefore implement the algorithm in order to use the results to dynamically tune the scheduling parameters of TBAA.

6.5 Reward Backpropagation Prioritized Experience Replay Double DQN

The Double DQN algorithm - as discussed in Section 5.9 - was previously tested by van Hasselt et al. (2016) on a testbed with a number of Atari 2600 games. Most of these games only incur a reward of +1 for winning the game or -1 for losing and therefore the reinforcement agent has to handle sparse rewards. The Double DQN algorithm yields new state-of-the art results on the games when compared to other reinforcement learning methods, which is why we adopt the algorithm and implement it in order to train a reinforcement agent on the container terminal simulation. The simulation program runs significantly slower than the Atari 2600 games, which is one of the reasons why we adapt some of the parameters of the algorithm. Furthermore, we combine the Double DQN algorithm with the RBPER algorithm of Zhong and Wang (2017). To the best of our knowledge, this combination has not been studied before (Schaul et al. (2015) combine PER with Double DQN and Zhong and Wang (2017) combine RBPER with a DQN, both on a number of Atari 2600 games), but due to the sparsity of the rewards, their delay and the limited amount of data (limited by the runtime of the simulation program) our reinforcement agent has to train on the combination seems appropriate. The experience replay buffer allows the agent to combine learning trajectories of multiple simulation models simultaneously, while the reward backpropagation method aims to deal with the delayed and sparse rewards. The use of Double DQN - as explained in Section 5.9 - prevents against autocorrelation. The input of both the neural networks (the evaluation and the target network) is a 4-dimensional array (one dimension for each of the four state attributes). The two hidden layers are fully-connected and consist of 128 rectifier units. The output layer is again a fully connected linear layer with one output for each action. Each of the layers applies the rectified linear activation function (ReLU). Furthermore, we use Adam optimizer (introduced by Kingma and Ba (2015)) with learning rate $\alpha = 0.001$ and its default settings for β_1, β_2 and ϵ ($\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$). A brief discussion of these default parameters is provided in 8.1.1 and the pseudocode of the Adam optimizer is presented in Appendix A.1.

We set the discount rate $\gamma = 0.999$ and the number of learning steps between target network updates $\tau = 1000$. A total of 10^5 transitions is saved in the memory and learning takes place in batch sizes of 64. Preliminary experiments that led to these parameter settings are discussed in Section 8.1. The exploration parameter ϵ starts with a value of 1.0 and decreases by a factor of 0.996 at every 100 learning steps, to allow for sufficient exploration of the algorithm. The minimum value for ϵ is 0.01, when this value is reached the parameter remains constant and the algorithm performs the action with the highest cumulative expected future discounted reward with probability 0.99. The pseudocode of the Reward Backpropagation Prioritized Experience Replay Double DQN algorithm for the container terminal scheduling problem is stated in Algorithm 6 and a schematic overview is provided in Figure 6.1. Note that Zhong and Wang (2017) assign higher priority weights to transitions with non-zero rewards and transitions that lead to a terminal state. Since the reinforcement agent in our setting receives a non-zero reward every simulation hour it always receives a non-zero reward in the terminal state (at the eighth simulation hour) and we therefore do not have to distinguish between these cases.

Algorithm 6: RLA: Reinforcement learning algorithm container terminal scheduling with Reward Backpropagation Prioritized Experience Replay and Double DQN

```
Initialize learning rate $\alpha$, mini-batch size $n$
Initialize replay memory $\mathcal{D}$ with capacity $N$
Initialize evaluation network $Q$ with random weights $\theta$ and target
  network $Q$ with weights $\theta'$
Initialize target network replacement frequency $\tau$
for each episode do
    Initialize state $s_1$
    for each transition do
        Draw $u \sim Unif(0,1)$
        if $u < \epsilon$ then
            select random action $a$
        else
            select $a = \max_a Q(s,a;\theta)$
        Execute action $a$ and observe reward $r$ and state $s'$
        if $r \neq 0$ or $s'$ is terminal then
            $p = \beta$
        else
            p = 1
        Store transition $(s,a,s',r,p)$ in $\mathcal{D}$
        Sample mini-batch of size $n$ of transitions $(s,a,s',r,p)$ with
          probabilities proportional to $p$ from $\mathcal{D}$
        for j=1,...,n do
            $(s,a,s',r,p)_{j-1} \leftarrow$ transition prior to transition $j$ in replay
              memory $\mathcal{D}$
            Define $a^{max}(s'_j;\theta) = argmax_{a'}Q(s'_j,a';\theta)$
            if $s'_j = terminal$ then
                Set $y_j = r_j$
            else
                set $y_j = r_j + \gamma Q(s'_j, a^{max}(s'_j,\theta);\theta')$
            if $r_{j-1} = 0$ and $p_j > 1$ then
                $p_{j-1} \leftarrow p_j$ in replay memory $\mathcal{D}$
            else if $r_{j-1} \neq 0$ and $p_j$¿1 then
                $(s,a,s',r,p)_k \leftarrow$ first transition after $(s,a,s',r,p)_j$ in $\mathcal{D}$
                  with $s'_k$ =terminal or $r_k \neq 0$
                $p_k \leftarrow \max(\lambda p_j, 1)$ in replay memory $\mathcal{D}$
            $p_j \leftarrow 1$ in replay memory $\mathcal{D}$
        Perform a gradient descent step by minimizing the loss
          $\mathcal{L}(\theta) = \frac{1}{n}\sum_{j=1}^{n}(y_j - Q(s_j,a_j;\theta))^2$
        Replace target network parameters $\theta'$ with $\theta$ every $\tau$ timesteps
```

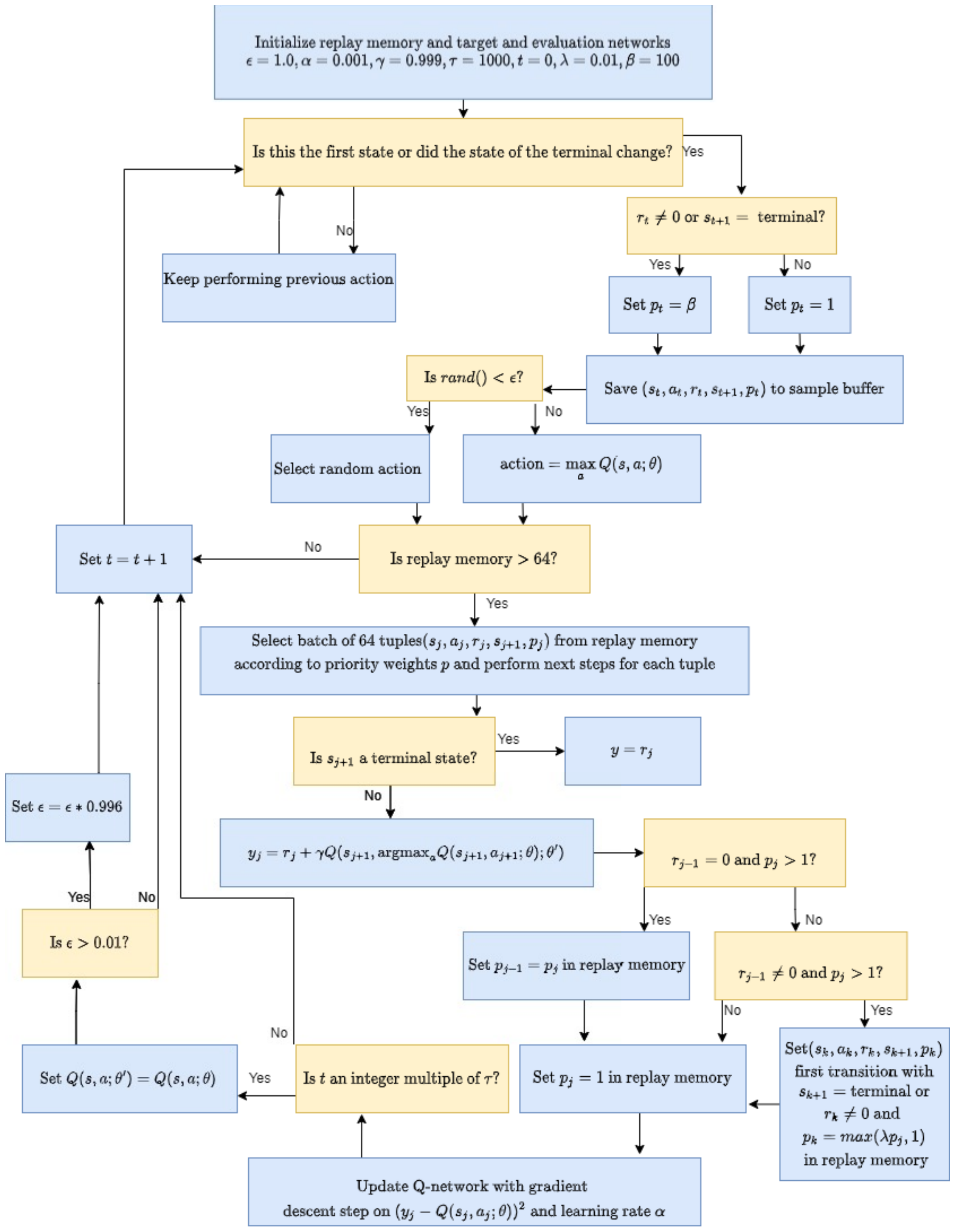

Initialize replay memory and target and evaluation networks
$\epsilon = 1.0, \alpha = 0.001, \gamma = 0.999, \tau = 1000, t = 0, \lambda = 0.01, \beta = 100$
Is this the first state or did the state of the terminal change?
Yes
No
Keep performing previous action
$r_t \neq 0$ or $s_{t+1} =$ terminal?
Yes
No
Set $p_t = \beta$
Set $p_t = 1$
Save $(s_t, a_t, r_t, s_{t+1}, p_t)$ to sample buffer
Is $rand() < \epsilon$?
Yes
No
Select random action
action $= \max_a Q(s, a; \theta)$
Set $t = t + 1$
No
Is replay memory > 64?
Yes
Select batch of 64 tuples$(s_j, a_j, r_j, s_{j+1}, p_j)$ from replay memory according to priority weights p and perform next steps for each tuple
Is s_{j+1} a terminal state?
Yes
$y = r_j$
No
$y_j = r_j + \gamma Q(s_{j+1}, \text{argmax}_a Q(s_{j+1}, a_{j+1}; \theta); \theta')$
$r_{j-1} = 0$ and $p_j > 1$?
Yes
No
Set $p_{j-1} = p_j$ in replay memory
$r_{j-1} \neq 0$ and $p_j > 1$?
No
Yes
Set $\epsilon = \epsilon * 0.996$
Yes
No
Is $\epsilon > 0.01$?
Set $Q(s, a; \theta') = Q(s, a; \theta)$
Yes
No
Is t an integer multiple of τ?
Set $p_j = 1$ in replay memory
Set$(s_k, a_k, r_k, s_{k+1}, p_k)$ first transition with $s_{k+1} =$ terminal or $r_k \neq 0$ and $p_k = max(\lambda p_j, 1)$ in replay memory
Update Q-network with gradient descent step on $(y_j - Q(s_j, a_j; \theta))^2$ and learning rate α

Figure 6.1: RLA: Reinforcement Learning Algorithm container terminal scheduling with Reward Backpropagation Prioritized Experience Replay and Double DQN.

Chapter 7

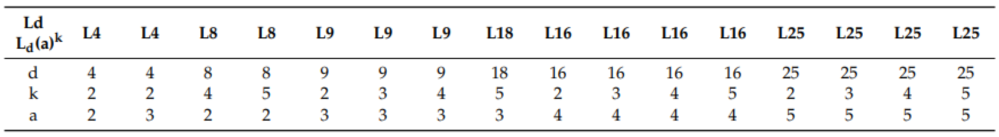
Taguchi Method

This chapter introduces the Taguchi Method: a experimental design method that can be used to test different levels of parameter settings within a minimal number of experiments. In order to efficiently combine the results of RLA and TBAA into DSA, we apply the Taguchi Method to an adapted version of TBAA. The theoretical background, algorithmic choices and results of the method are discussed in this chapter and the results of the produced algorithm DSA are provided in Section 8.2. Section 7.1 introduces the Taguchi methodology and Section 7.2 discusses its application to the setting in this thesis.

7.1 Taguchi Experimental Design Method

Dr. Genichi Taguchi (1924-2012) was a Japanese statistician and engineer who devoted his life to continually evolving his philosophy of quality engineering. He developed a method - consisting of five steps - that was shown to improve the quality of manufactured goods, while reducing costs (Taguchi et al., 2004). The method - first referred to as the Experimental Design Method, and later as the Taguchi Method - has been applied to a wide variety of processes and experiments, ranging from mechanical component design in automotive applications (Shim and Kim, 2009) and meta-heuristic parameter optimization (Candan and Yazgan (2014), Wang et al. (2014)) to the optimization of process parameters for the production of liquid fuel from waste (Agboola et al., 2017).
As in most applications full factorial design - testing of each possible combination of parameter settings - is computationally too exhaustive, the Taguchi Method investigates only a subset of all possible parameter settings. Therefore it is a fractional factorial design method, but with additional general guidelines that aim to reduce the variance of the experiments. The Taguchi Method consists of five main steps (Woolf et al., 2020).
The first step is to define the objective: the desired outcome of the process or set of experiments. This can be a minimum or maximum objective value, or an exact target value, completely depending on the problem at hand. In line with the notation of Taguchi et al. (2004) we refer to this objective or target value as τ and to the observed objective value as y.
In the second step the design parameters that affect the process are determined and the number of levels that the parameters can attain are specified. In-depth understanding of the process, parameters and possible interactions between parameters is

crucial for this step and for a successful implementation of the Taguchi Method. The third step selects the appropriate orthogonal array with the number of experiments and the parameter settings for each experiment. An orthogonal array is suitable for this purpose since it obeys two important properties: any two columns contain each level combination for an equal number of times and the arrays are balanced. Balanced arrays contain each combination of parameter levels with equal frequency when compared to any other combination of parameter values. Therefore, orthogonal arrays determine the minimum number of experiments that need to be conducted for an equal comparison of the effects of different parameters and levels. The general representation of these orthogonal arrays is $L_d(a)^k$, where L is the type of orthogonal array, d the total number of trials, k the number of parameters and a the (maximum) number of levels per parameter (Uray et al., 2022). Figure 7.1 shows (a subset of) the possible choices of orthogonal arrays, which can be read in the following way: assume we want to conduct experiments with four parameters, that can each attain three different levels ($k = 4$ and $a = 3$). We then look up orthogonal array $L9$ in - for example - Hedayat et al. (1999) and conduct 9 experiments.



Ld $L_d(a)^k$	L4	L4	L8	L8	L9	L9	L9	L18	L16	L16	L16	L16	L25	L25	L25	L25
d	4	4	8	8	9	9	9	18	16	16	16	16	25	25	25	25
k	2	2	4	5	2	3	4	5	2	3	4	5	2	3	4	5
a	2	3	2	2	3	3	3	3	4	4	4	4	5	5	5	5

Figure 7.1: Orthogonal arrays $L_d(a)^k$, where L is the type of orthogonal array, d is the total number of trials that is needed, k is the number of parameters and a the (maximum) number of levels per parameter (Uray et al., 2022).

Table 7.1 denotes orthogonal array $L9$. The rows represent the experiment numbers and the columns the four different parameters A, B, C and D. The first experiment hence sets all four parameters to their first level. Notice that it is advisable to conduct multiple replications of each experiment in random order, to increase the accuracy of the results and diminish the effects of possible correlation between experiments (Taguchi et al., 2004).

Experiment	A	B	C	D
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

Table 7.1: Orthogonal array $L9$, with 9 experiments (rows), four parameters (columns) and three different parameter levels that are specified for each of the experiments in the table. Copied from Hedayat et al. (1999).

The fourth step consists of conducting the experiments that are indicated in the orthogonal array and collecting the data on the process objective. This results in objective values y_{ij} for each experiment i and replication j.
In the last step of the method the researcher conducts a complete data analysis to determine the effect of the different parameters on the process objective. Taguchi et al. (2004) advise to first calculate the signal-to-noise ratio (SN): a performance measure that evaluates the effect of each parameter setting on the output. The signal-to-noise ratio - in this context - is the ratio between the effect of a parameter setting (the desired information, or the signal) and other processes affecting the experiment (the undesired signal, or background noise). A larger SN ratio indicates a larger effect of the parameter setting when compared to the noise and is therefore more desirable than a lower SN ratio. Taguchi et al. (2004) also provide the possibility to add controlled noise factors by employing an additional orthogonal array. In that case, the previously discussed orthogonal array is the inner orthogonal array, and the controlled noise factors (e.g. environmental changes or adjustments in the setup of the experiment that are not caused by a change in the parameters under consideration) are specified in an outer array. Since in our experiments the noise factors are induced by the numerous processes with probability distributions that are defined in the simulation program TIMESQUARE, we will not vary those manually and therefore do not consider any outer arrays in this thesis. Krishnaiah and Shahabudeen (2012) point out that most traditional techniques in experimental design focus solely on identifying parameters that affect the mean of the objective values, while the SN ratio considers both the mean as well as the standard deviation of the objective values corresponding to the conducted experiments. This results in the selection of a parameter setting that is more robust to the effects other (uncontrolled) processes might have on the considered objective values than when only mean objective values are assessed.
Taguchi et al. (2004) introduce three equations to calculate SN: one for the setting in which the objective value y is maximized (Equation 7.1), one for the setting in which there is a target value τ (Equation 7.2) and one for the setting in which the objective value y is minimized (Equation 7.5). In these equations N_i is the number of replications for experiment i and $\bar{y}_i$ and s_i are the sample mean and standard deviation of the observations belonging to experiment i.

$$SN_i = -10log_{10}\Big[\frac{1}{N_i}\sum_{j=1}^{N_i}\frac{1}{y_j^2}\Big] \tag{7.1}$$

$$SN_i = 10log_{10}\frac{\bar{y}_i^2}{s_i^2} \tag{7.2}$$

$$\bar{y}_i^2 = \frac{1}{N_i}\sum_{j=1}^{N_i} y_{i,j} \tag{7.3}$$

$$s_i^2 = \frac{1}{N_i - 1}\sum_{j=1}^{N_i}(y_{i,j} - \bar{y}_i) \tag{7.4}$$

$$SN_i = -10log_{10}\Big[\frac{1}{N_i}\sum_{j=1}^{N_i} y_j^2\Big] \tag{7.5}$$

Notice that Equation 7.2 explicitly calculates the sample standard deviation of each of the experiments, while Equations 7.1 and 7.5 do not. The SN ratios for these

equations, however, are still higher for experiments with the same mean but smaller standard deviations than for experiments with the same mean and larger standard deviations.
The SN ratio is first determined for each experiment (SN_i) and then the average for each combination of parameter and level is calculated, resulting in $a \times k$ SN ratios. For each parameter, the level that corresponds to the largest of the a SN-values is chosen as the final parameter setting. For a complete analysis Taguchi et al. (2004) propose to conduct an Analysis of Variance (ANOVA): a collection of statistical tools that determine the statistical significance of and differences between the results. Candan and Yazgan (2014) and Krishnaiah and Shahabudeen (2012) propose to include at least - per parameter - the sum of squares, the percent contribution, F-values and p-values. We elaborate on these terms in Section 7.2, when we apply the Taguchi Method in order to optimize the parameter settings of DSA.

7.2 DSA parameter design by Taguchi Method

The scoring algorithm TBAA is relatively simple to understand, but has a large number of parameters (approximately 60, of which 30 are used in the model under consideration) that affect the scoring of each job. The actions that can be chosen by the reinforcement agent, as described in Section 6.2, consist of combinations of less than 10 parameters in total. Some parameters - like duetime or driving distance - are already present in TBAA, but others - like whether a buffer is quiet or busy - need to be implemented. We want to refrain from deleting all original parameters and their settings, since these are based on long expertise of simulation consultants. We therefore aim to merge the results of RLA and the existing TBAA in such a way that the two algorithms complement each other. RLA provides us with the most suitable action in each of the 144 states of the terminal, hence we adapt the scoring algorithm in such a way that the scoring parameters corresponding to this action attain different values. These values should not be much higher than any of the other (existing) scoring parameters in order to be effectively merged with TBAA, but neither should they be lower since the RLA decisions would then have little to no effect. The exact value of the parameters is to be determined by the Taguchi Method, that we perform in the exact order as prescribed in Section 7.1.

As we are maximizing QC productivity a straightforward objective would be to maximize the hourly QC productivity on the terminal. This is easily determined for different parameter settings after implementation in the simulation program TIMESQUARE, which allows for a thorough comparison of the various experiments. RLA results (elaborate results are provided in Section 8.1) indicate that 6 out of the proposed 14 actions provide the highest cumulative expected reward in at least one of the 144 states. In other words: in any of the states one of these 6 actions is the optimal one. We therefore need to establish suitable parameter settings for the parameters corresponding to each of these 6 actions:

- Action 3: *SER*. Shortest driving distance, earliest duetime, respect *MaxToQC*.
- Action 6: *BBN*. Busiest buffer, shortest driving distance, no *MaxToQC*.
- Action 8: *QBN*. Quietest buffer, shortest driving distance, no *MaxToQC*.

- Action 9: *LSS*. Load moves, sequence number, shortest driving distance, respect *MaxToQC*.
- Action 10: *LSR*. Loading QC first, shortest driving distance, respect *MaxToQC*.
- Action 12: *USR*. Unloading QC first, shortest driving distance, respect *MaxToQC*.

Three scoring parameters were added to TBAA: one that increases the score for jobs that correspond to loading QCs, one that increases the score for jobs that correspond to discharging QCs and one that penalizes jobs according to the number of containers that need to be moved to or from its origin and destination (the busiest buffer). The latter can be multiplied by a negative number to promote jobs with busy buffers (action 6) and with a positive number to promote jobs with quiet buffers (action 8). Preliminary experiments led to a base level for all three parameters, in which they have a minor, but non-zero effect on the final scores of each job.
Recall that - after ordering on the specified parameters - the reinforcement agent orders the jobs in increasing order of sequence number to prevent containers from being picked up or delivered in an unfavourable order. Furthermore, actions 10 and 12 were designed to first assign the jobs to or from QCs with the smallest number of currently assigned vehicles. *MaxToQC* does not require a specific value: we either respect the number of vehicles that is allowed to drive to or from a QC or we do not. The exact number of vehicles that corresponds to this parameter was tuned beforehand and is therefore not considered in these experiments. Each action hence consists of three parameters, except for actions 10 and 12, that consist of four parameters.
For the Taguchi Method to be effective, the different parameters should be as uncorrelated as possible. Correlated parameters have different effects on the objective values when their values are altered separately and would therefore require additional experiments. We therefore choose to combine all three (and four, for actions 10 and 12) parameters into one tuple that defines the factor by which all corresponding scoring parameters of TBAA should be multiplied. For example: one parameter level that corresponds to action 3 is $[2, 1.5, 1]$. Then the scoring parameter corresponding to the driving distance is doubled, the scoring parameter that corresponds to the due time is multiplied by 1.5 and the scoring parameter corresponding to the sequence number remains at its original level. By doing so, we do not have to take into account the correlation between the different parameters *within* one action when constructing the Taguchi experiments. Correlation *between* actions does still affect the reliability of the method, but as these actions are never chosen at the same time and each correspond to a separate state of the terminal we assume their correlation is negligible. Preliminary experiments were conducted to specify the range in which the parameter settings would influence TBAA and five levels per parameter were determined. The parameters and their levels (the multiplication with the base parameter levels) are provided in Appendix A.3.
The appropriate orthogonal array for a model with six parameters that can each attain five levels is $L_{25}(5)^6$ or L_{25}, according to Hedayat et al. (1999). Comparing these 25 experiments to the total number of required experiments for a full factorial design ($5^6 = 15625$) indicates the significant reduction in computational time of applying the Taguchi Method. The orthogonal table that specifies for each of the 25 experiments which parameter settings must be used is provided in Appendix

A.2. Notice that the first experiment (with all parameters at their first levels) corresponds to the parameter settings of TBAA, with the exception of the three added parameters at their base levels. We conduct the experiments that are indicated in the orthogonal array and collect the data on the process objective. For each of the 25 experiments we conduct 8 replications for 3 different QC placements on the terminal with 24 ShCs. The differences between the QC placements are specified in Chapter 8. This results in 600 objective values in the form of hourly QC productivity per replication. The results are provided in Appendix A.4.

In order to conduct a complete data analysis to determine the effect of the different parameters on the process objective we first calculate the SN ratio for each experiment by applying Equation 7.1. The results are provided in the last column of the table in Appendix A.4. We then calculate the average SN ratio for each combination of parameter and level. For example, the SN ratio corresponding to parameter A and level 1 is the average SN ratio over all experiments where parameter A attains level 1: $\frac{30.50+30.47+30.38+30.46+30.39}{5} = 30.44$. Table 7.2 denotes the results (with four decimals for a more thorough comparison, as the differences are relatively small). The parameter-level combinations with the highest SN ratio should be considered best. As explained in Section 7.1 these settings provide results that are robust to other, (in these experiments) uncontrolled factors that influence the QC productivity on the simulated container terminal, while maximizing the mean QC productivity. In this case that is parameter A with level 4, parameter B with level 4, parameter C with level 1, parameter D with level 2 and parameters E and F with level 1. The range denotes the difference between the maximum and the minimum SN ratio of that parameter and the parameter with the largest range is the parameter for which a change in level has the largest effect on the objective values. The rank orders the parameters in decreasing order of this range and therefore in decreasing order of the effect that a change in level has on the objective value.

Parameter	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**	**Range**	**Rank**
A	30.4399	30.4244	30.3974	30.4458	30.409	0.0484	3
B	30.4332	30.4297	30.4311	30.4547	30.3693	0.0854	1
C	30.4366	30.4266	30.3956	30.4320	30.4257	0.0410	4
D	30.4264	30.4318	30.4300	30.4305	30.3978	0.0340	5
E	30.4487	30.4366	30.3970	30.3982	30.4359	0.0517	2
F	30.4358	30.4248	30.4029	30.4199	30.4331	0.0329	6

Table 7.2: Average SN ratios Taguchi experiments. The range is difference between the minimum and maximum SN ratio and the rank orders the parameters in decreasing order of range.

Notice that - in this set of experiments - selecting the parameter setting with the largest mean objective value would result in the exact same parameter setting as the current selection that is based on the maximum SN ratio.

For the sake of completeness we include an ANOVA analysis on the SN ratios in Table 7.3 and adopt the methodology that is proposed by Krishnaiah and Shahabudeen (2012). Calculations were done in RStudio version 1.2.5042. The Sum of Squares (SS) measures the variability in the data that is caused by each of the parameters and is calculated for parameter i, $i = 1, ..., 6$ as follows:

$$SS_i = \frac{1}{J}\sum_{j=1}^{J}(Y_{ij} - \bar{Y}_i)^2$$

In this equation Y_{ij} is the SN ratio of parameter i at level j and $\bar{Y}_i$ is the average SN ratio of parameter i over all 5 levels. J is the total number of SN ratios that are used to calculate the average SN ratio at level j (5 for each level). Then we can calculate the percentage effect each parameter has on the objective value: the factor effect. This is calculated by dividing the Sum of Squares by the total Sum of Squares. It becomes clear from both Table 7.2 as well as Table 7.3 that parameters D and F have the smallest effect on the objective value, when compared to the other parameters. We therefore exclude this parameter when we perform a linear regression and test the significance of the other parameters. Leaving all parameters in the model refrains us from calculating the F-test statistics, since we would then fit a model using all the model terms, leaving zero degrees of freedom for the residual error. Krishnaiah and Shahabudeen (2012) propose to pool the parameters with the smallest factor effect into the error term, up to a maximum of half the total degrees of freedom of the experiment. The degrees of freedom for each parameter are the number of levels it can attain minus one. Therefore, the degrees of freedom of the pooled error term is 8: 4 from parameter D and 4 from parameter F. The Mean Squares (MS) is calculated as $\frac{SS}{DF}$ and can be used to conduct an F-test that tests the null hypothesis that the parameter has no significant effect on the outcome of the experiment. Rejecting this null hypothesis would mean that the parameter does have a significant effect on the objective values and selecting the optimal level by means of the SN ratio is then advised (Krishnaiah and Shahabudeen, 2012). The F-test statistic is calculated by dividing the Mean Squares of a parameter by the Mean Squares of the (pooled) error and the null hypothesis is rejected when the F-test statistic is larger than the critical F-value F_{α,DF_1,DF_2}, where α is the desired confidence level, DF_1 the degrees of freedom of the parameter and DF_2 the degrees of freedom of the error term. Table 7.3 shows that not all parameters have a significant effect on the outcome of the experiment. Depending on the significance level one could state that parameters B (significance level $\alpha = 0.05$, critical F-value 3.84) and E (significance level $\alpha = 0.10$, critical F-value 2.81) have a significant effect on QC productivity. Notice that we could have pooled parameter C into the error term as well. This would have resulted in an increase in the degrees of freedom of the error term and therefore a decrease in the critical F-value. In that case, parameter E would have a significant effect on the outcome of the experiment at a confidence level of $\alpha = 0.05$. The other parameters at these 5 specified levels do not seem to have a significant effect on the outcome of the experiment, but still need to attain a certain level. We therefore adopt the parameter levels with the largest SN ratio per parameter in order to adapt TBAA into DSA.

Source of variation	**SS**	**FE (%)**	**DF**	**MS**	**F-test statistic**
A	0.000332	15.68	4	0.000083	2.20
B	0.000814	38.49	4	0.000204	5.40
C	0.000208	9.83	4	0.000052	1.38
D*	0.000165	7.81			
E	0.000460	21.75	4	0.000115	3.05
F*	0.000136	6.44			
Pooled error			8	0.000038	

Table 7.3: ANOVA results on SN ratios of the 25 Taguchi experiments. The Sum of Squares (SS), factor effect (FE), degrees of freedom (DF), Mean Squares (MS) and the F-test statistic are reported, except for the parameters D and F since these are pooled into the error (denoted by an asterisk). The critical F-values are $F_{0.05,4,8} = 3.84$ and $F_{0.10,4,8} = 2.81$.

Chapter 8

Comparative analysis

In this chapter we provide a thorough analysis of the results after implementation of the algorithms. Section 8.1 concerns the results of the reinforcement learning algorithm RLA and compares those with the simulation results of the existing algorithm TBAA. This section also provides results on the preliminary experiments that led to the parameter settings of RLA. Section 8.2 provides the results of the final algorithm DSA, with dynamic scheduling parameter settings that were determined by means of the Taguchi Method as discussed in Chapter 7. Section 8.3 shows the robustness of the results by discussing the performance of DSA on off-peak simulations.

8.1 Results RLA

All experiments in this thesis were conducted on a Lenovo Thinkpad laptop with an Intel i7 processor of 2.7 GHz and 16 GB RAM. The simulation program TIMESQUARE makes use of the software eM-Plant and runs on external TBA servers. As eM-Plant - with programming language SimTalk - does not provide any additional packages that involve artificial neural networks the reinforcement learning model was implemented in Python 3.10.5 with TensorFlow 2.8.0 and one Nvidia Quadro T1000 GPU. Communication between the Python code and eM-Plant made use of a SQL database, by continuously updating information about the terminal and the required scheduling decisions. The simulated terminal is a smaller version of the Hadarom Container Terminal (HCT) in Israel, with 20 yard modules and 8 QCs. Three different QC placements and models with 20, 24 and 28 ShCs are used to train the reinforcement agent in order to visit as many states as possible and to ensure robustness of the results. Varying the number of ShCs per quay crane (2.5, 3 and 3.5) is a technique that is commonly used by TBA simulation experts to identify bottlenecks on the terminal and to decide on the most desirable setting for every simulation model. Notice that we refer to 2.5, 3 and 3.5 ShCs per quay crane, but we do not assign ShCs to specific QCs: each ShC can service any QC. The referral to the number of ShCs per QC is employed for clarity and in order to compare the results to experiments with a different number of QCs.
Since the reinforcement agent is required to make real-time decisions on the simulated terminal the amount of data that can be progressed is bounded by the computational time of the simulation model. In the original model, one simulation day of 8 hours takes approximately 1.5 hours to complete. This computational time increases to 2-2.5 hours after implementing RLA. In order to process more data we

adjust the Python code such that it is able to train on up to 5 models simultaneously: 5 models in eM-Plant run at the same time and one reinforcement agent in Python makes all of the scheduling decisions, while training the neural network. Adding more models is possible, but increases computational time since the Python code can only make one scheduling decision at a time and the simulation models then have to wait longer for their turn. The computational time that is needed to update the neural network is negligible when compared to the computational time of the simulation model. The data we can use to train the neural network is limited, when compared to the implementation of the DQN algorithm of Mnih et al. (2013). For comparison: Mnih et al. (2013) process 10 million state transitions in 50 hours, while we process approximately 50000 state transitions in the same amount of time. As there is no time nor enough resources to run the algorithm for several months we slightly adjust the parameter settings that were proposed by Mnih et al. (2013) - and adopted by Mnih et al. (2015), Zhong and Wang (2017) and van Hasselt et al. (2016) for the discussed algorithms.

8.1.1 Parameters RLA

Preliminary experiments are conducted to determine parameter settings that are most suitable for the problem at hand. Ideally, each of the following parameters would be thoroughly tested and their performance analysed:

- Mini-batch size n: the number of transitions that are sampled from the experience memory at each learning moment in order to train the neural network. Too small, as well as too large mini-batch sizes decrease accuracy of the results, as shown by Masters and Luschi (2018). The authors furthermore state that the optimal mini-batch size depends on the learning rate and vice versa, hence these are best tuned simultaneously.

- Experience replay memory size N: the number of transitions that is saved in the experience memory. Once this number is reached, the oldest transitions are removed. For larger memory sizes it is less likely to sample correlated transitions and training will be more stable. Smaller memory sizes require less memory, which could speed up the learning process.

- Discount rate γ: a reward that is received k time steps from now is worth γ^{k-1} times its value at the current moment in time. A value for γ closer to 1 makes future rewards more valuable, while for a value closer to 0 the agent is mostly concerned with maximizing immediate rewards.

- Learning rate α: at each learning moment the weights of the neural network are updated by a fraction α of the prediction error. Larger learning rates incur bigger changes to the the network weights at each learning moment, which could lead to quicker learning processes. They can, however, also prevent convergence due to the large variability in Q-values between different learning moments. With smaller learning rates, smaller changes are made to the neural network at each learning moment and one might need more data to train on.

- Exploration rate ϵ: the exploration rate defines the probability of picking a random action at each decision moment. Generally ϵ starts at 1, so the reinforcement agent always picks a random action in the beginning of the algorithm to allow for exploration. During the learning phase ϵ is gradually decreased by a specified rate until it reaches a minimum level. The minimum level should be reached before the end of the learning process, preferably leaving sufficient decision and learning moments at its minimal level to allow for exploitation near the end of the algorithm.

- Target network update frequency τ: the frequency at which the target neural network is updated with the weights of the evaluation network. Larger values of τ can decrease the accuracy of the results, since changes in the network weights between the update moments might not be accounted for. Smaller values of τ decrease the benefit of using a target network, which is reducing correlation between the Q-values (as discussed in Chapter 5). In the specific context of Double DQN it also reduces the benefit of using the target network to estimate the target value and the evaluation network to select the optimal action. As both networks will be very similar for a small value of τ the overestimation of action values will not be reduced.

- Design neural network: the number of hidden layers, their connection (fully-connected or not) and the number of rectifier units. In general, more complicated models require a larger number of hidden layers and rectifier units and less complicated models (with fewer variables) require smaller neural networks. Using too many hidden layers and/or rectifier units on a less complicated model induces the risk of overfitting, while too few hidden layers and/or rectifier units might underfit a more complicated model.

- Priority weight β: the priority weight that is assigned to a transition with non-zero reward when it enters the experience memory. Transitions with larger priority weights have a higher probability of being sampled at each learning moment.

- Priority decay rate λ: the rate at which the priority weight decreases when one round of backpropagation is completed. For higher values of λ, the priority weights approach 1 more quickly and fewer rounds of backpropagation are ensured. Smaller values of λ ensure higher priority weights from the second round of backpropagation and therefore more rounds will be completed.

- TensorFlow settings: in Python, TensorFlow provides a wide variety of options. One needs to choose an activation function: a function that defines how the output of each artificial neuron is calculated from its input signal. Furthermore, an optimizer defines the exact implementation of the gradient descent step at each learning moment. For example: Stochastic Gradient descent, Adagrad, RMSProp and Adam. Each of the optimizers have their own hyper-parameters that can be tuned in order to optimize their performance.

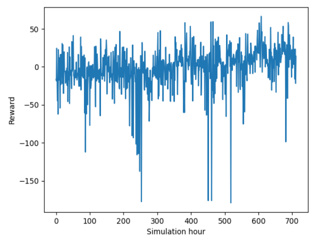
The computational time of RLA does not allow for many and lengthy preliminary experiments to test each of the parameters extensively. We therefore conduct five experiments of 48 hours each and collect the results. The first experiment is a benchmark experiment and at each of the other four experiments we test the effect of changing one parameter to a different value. We are aware of the fact that changing one parameter at a time ignores possible correlation between parameter settings (as indicated by Masters and Luschi (2018) for the learning rate α and mini-batch size n). Furthermore, 48 hours is not enough for our algorithm to provide the most desirable results as more transitions are needed to thoroughly train the reinforcement agent. We therefore rely on the decisions that were made by Mnih et al. (2013), van Hasselt et al. (2016) and Zhong and Wang (2017) for the benchmark experiment, with four adjustments: instead of a neural network with 2 hidden layers and 256 rectifier units we implement a neural network with 2 hidden layers and 64 rectifier units, because our state space is significantly smaller than the state spaces in their experiments (144 states with dimension 4 instead of $84 \times 84 \times 4$ grey-scale images) and we want to reduce the risk of overfitting. Furthermore, as we have limited data to learn from we increase the learning rate α from 0.00025 to 0.001 and decrease the target network update frequency τ from 10000 to 1000 in order to have sufficient updates for experiments with fewer transitions. We can also decrease the experience replay memory size N from 10^6 to 10^5, since 10^6 state transitions require more computational time than we can afford (approximately 40 days). The parameters for the benchmark experiments are denoted in Table 8.1. We set the initial value for ϵ to 1.0 and decrease it every 100 transitions by a factor of 0.988 in order to reach the minimum value of 0.1 after approximately 75% of the designated time of 48 hours. To perform the gradient descent step that updates the neural network we use the Adam (adaptive moment estimation) optimizer that was developed by Kingma and Ba (2015), since it combines the advantages of two other optimizers: AdaGrad (adaptive gradient algorithm) and RMSProp (root mean square propagation), both extensions of existing SGD algorithms. As stated in Section 6.5, the pseudocode for the Adam optimizer - provided by Kingma and Ba (2015) - is presented in Appendix A.1. Kingma and Ba (2015) claim that Adam is a relatively stable optimizer that requires little memory and can handle noisy datasets. Furthermore, they state that the default hyperparameter values perform well on most problems. We therefore adopt these default values: $\beta_1 = 0.9$ (the exponential decay rate for the first moment estimates), $\beta_2 = 0.999$ (the exponential decay rate for the second moment estimates) and $\hat{\epsilon} = 10^{-8}$ (a small constant for numerical stability). We use the rectified linear activation function ReLu that decides whether an artificial neuron should be activated or not. For any input $x \in \mathbb{R}$, ReLu outputs $max(0, x)$. This is a very straightforward mathematical operation that requires little computational effort and it became the most commonly used activation function after AlexNet won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012 by using ReLu on a Convolutional Neural Network (Krizhevsky et al., 2012).

We decide to vary four parameters that presumably have the largest effect on the performance of the algorithm: n, α, γ, and the number of rectifier units of the neural network. We conduct four additional experiments of 48 hours: one with a mini-batch size of 64, one with a learning rate α of 0.01, one with a discount rate γ of 0.999 and one with a neural network consisting of 2 hidden layers and 128 rectifier units. We present the rewards (one per simulation hour, except for the first hour of each simulation day) in Figure 8.1, the loss (of every learning moment) in Figure 8.2 and the average and maximum Q-values (calculated every 100 state transitions) in Figure 8.3 and Figure 8.4, respectively.

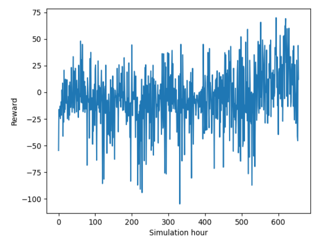
Rewards in the form of QC productivity per hour are compared to the QC productivity of TBAA, as described in 6.3. Therefore, a reward that exceeds zero corresponds to one simulation hour of RLA with a higher QC productivity than 80% of the average QC productivity of TBAA. Similarly, rewards below zero correspond to simulation hours of RLA with a lower QC productivity than 80% of the average QC productivity of TBAA. Due to many different processes at the terminal, each

Parameter	Value
Mini-batch size n	32
Experience replay memory size N	10^5
Discount rate γ	0.99
Learning rate α	0.001
Exploration rate ϵ	$1.0 \rightarrow 0.1$ with a rate of 0.988 every 100 transitions
Target network update frequency τ	1000
Number of hidden layers neural network	2
Number of rectifier units neural network	64
Priority weight β	100
Priority decay rate λ	0.1
Activation function	ReLu
Optimizer	Adam, with default parameters $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$

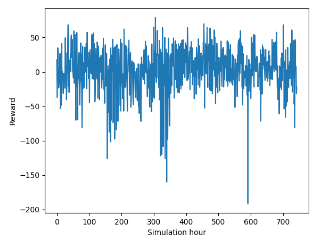
Table 8.1: Parameter settings for the benchmark experiment. Derived from Mnih et al. (2015) and Zhong and Wang (2017), with slight adjustments.



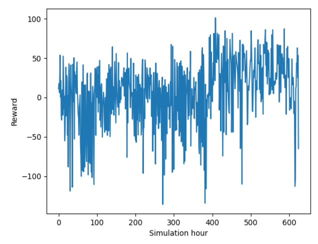
Benchmark experiment.



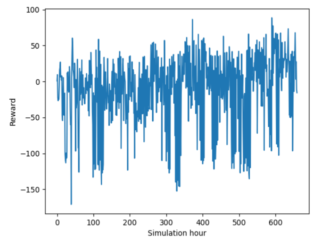
Mini-batch size increased to n = 64.



Learning rate increased to α = 0.01.



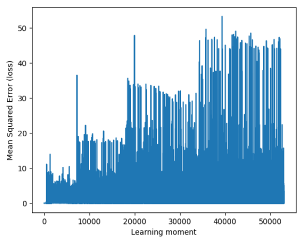
Discount rate increased to γ = 0.999.



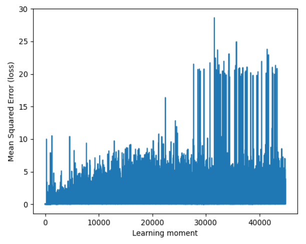
Number of rectifier units increased to 128.

Figure 8.1: RLA rewards per simulation hour for preliminary experiments.

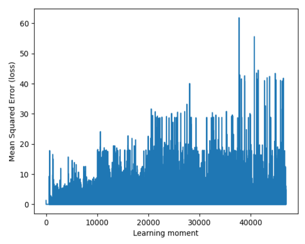
with specified distribution functions, QC productivity fluctuates. The reinforcement agent therefore needs to process a significant amount of data in order to determine whether his actions result in a negative reward due to this degree of randomness, or because the chosen actions are not suitable for the state. Rewards below -100 only occur when one or more QCs did not perform any move in the corresponding simulation hour, either because the reinforcement agent has sent too many vehicles to the same place and congestion occurred or because a bug in the simulation program resulted in an unresponsive crane or vehicle. We have spent a lot of time removing as many bugs from the model as possible, but approximately one in every 20-30 simulation days yields a low reward that is not necessarily caused by the decisions of the reinforcement agent. We therefore do not compare the rewards of the five preliminary experiments in Figure 8.1 based on their most negative rewards. We prefer to pick a parameter setting that allows our reinforcement agent to learn to take actions that yield higher rewards in the majority of the experiments. In each of the five graphs in Figure 8.1 we see a slight increase in rewards from simulation hour 350-400 onwards, except for the experiment with $\alpha = 0.01$. There, maximum rewards are lower and no improvement is visible within 700 simulation hours. The highest rewards are found in the experiments with $\gamma = 0.999$ and with a larger neural network.



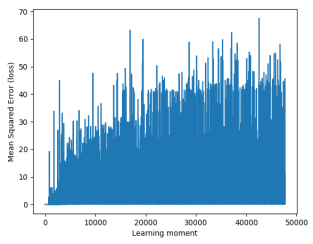
Benchmark experiment.



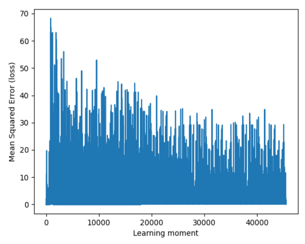
Mini-batch size increased to $n = 64$.



Learning rate increased to $\alpha = 0.01$.



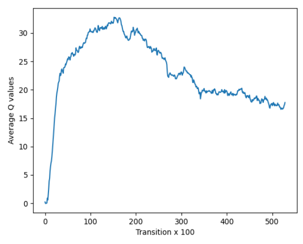
Discount rate increased to $\gamma = 0.999$.



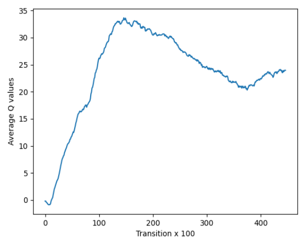
Number of rectifier units increased to 128.

Figure 8.2: RLA Mean Squared Error (loss per learning moment) for preliminary experiments.

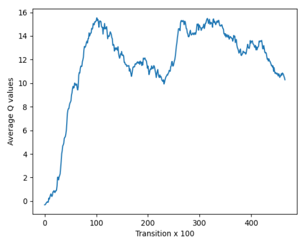
As proposed by Zhong and Wang (2017) the weights of the neural network are updated after each transition that is added to the experience memory. Therefore, the number of learning moments in Figure 8.2 coincide with the number of state transitions. It is clear that the algorithm did not converge (yet), since the mean squared error mostly seems to increase and varies significantly. A decrease is only visible in the graph that depicts the experiment with a larger neural network. Possibly, since the environment is only partly observable to the reinforcement agent, a more complicated neural network is able to capture more information about the underlying MDP. Furthermore, increasing the mini-batch size clearly reduces the mean squared errors. At every learning moment more transitions are sampled from the experience memory and the average loss over more transitions is calculated. Therefore, big outliers have a smaller impact on the average loss and the mean squared errors are lower. They do, however, increase over the course of the preliminary experiment.
Since convergence of the algorithm is not guaranteed, Mnih et al. (2013) and Zhong and Wang (2017) do not report the loss of every learning moment, but focus on the



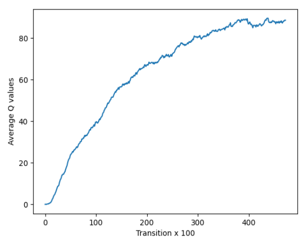
Benchmark experiment



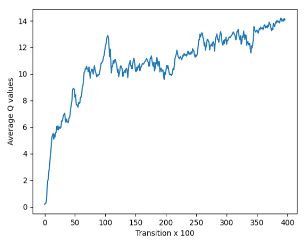
Mini-batch size increased to $n = 64$.



Learning rate increased to $\alpha = 0.01$.



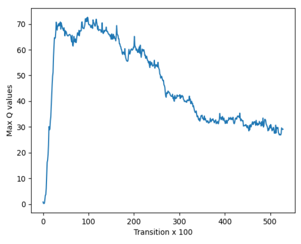
Discount rate increased to $\gamma = 0.999$.



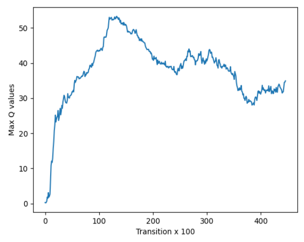
Number of rectifier units increased to 128.

Figure 8.3: RLA average predicted Q-values (calculated every 100 transitions) for preliminary experiments

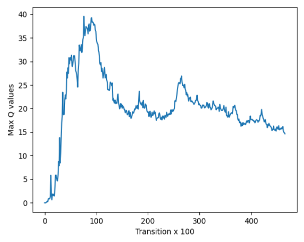
predicted average and maximum Q-values. We report the average and maximum Q-values of the evaluation network that is calculated every 100 transitions. Notice that we did not pick the target network, since that network only updates every 1000 transitions. The evaluation network leads to more data points, but its weights are expected to vary more. Figure 8.3 and Figure 8.4 depict the differences between the five experiments in average Q-values and maximum Q-values, respectively. The benchmark experiment shows a large increase in average and maximum Q-values in the first 5000 transitions, but a decrease after 18000 transitions. This experiment also has fewer large negative rewards in the first 100 transitions than the other experiments, which could explain the steeper ascent. The only experiments that show persistently increasing average Q-values are the experiments with $\gamma = 0.999$ and with the larger neural network. The former graph is the most smooth and has significantly higher values than the latter. A larger discount rate adds more value to future rewards, which can lead to larger Q-values. The graph corresponding to the experiment with a network with 128 rectifier units indicates that larger neural networks can have the tendency to overfit the data, since this graph is less smooth than the other ones. Each network update seems to have a larger effect on the network weights. Lastly, a decrease in average and maximum Q-values is not necessarily a bad sign. Once the values seem to stabilize, the predicted Q-values can be used to derive conclusions. As long as the Q-values are still being updated heavily it is wise to continue the reinforcement learning algorithm, since there is no guarantee that the current Q-values provide the desired results. We therefore aim to run the final experiment of RLA significantly longer than the preliminary experiments.
On the basis of these preliminary experiments we choose to increase the mini-batch size to $n = 64$, since the maximum rewards are higher than the maximum rewards of the benchmark experiment and they seem to increase more towards the end of the algorithm. The average and maximum Q-values also seem slightly more stable. We



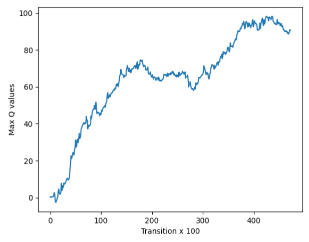
Benchmark experiment



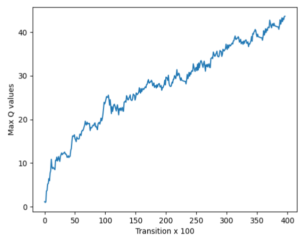
Mini-batch size increased to $n = 64$.



Learning rate increased to $\alpha = 0.01$.



Discount rate increased to $\gamma = 0.999$.



Number of rectifier units increased to 128.

Figure 8.4: RLA maximum predicted Q-values (calculated every 100 transitions) for preliminary experiments

do not increase the learning rate α, since there is no visible increase in rewards, which could be a sign that the reinforcement agent is unable to learn from his experience. We do increase the discount factor γ to 0.999, since the steady and stable increase in Q-values of this experiment could indicate stability in the learning process. Lastly, even though it is hard to tell whether the higher maximum rewards are due to randomness or to the design of the neural network, we increase the number of rectifier units to 128. The graphs with the average and maximum Q-values are slightly less smooth than the other experiments, but increase more steadily. Furthermore, it is the only experiment that is able to decrease the loss during the course of the preliminary experiment. The final parameter settings that are used for RLA are provided in Table 8.2. Notice that we increase the decay rate of ϵ to 0.996 for the final run of RLA, since we run a significantly longer experiment and we do want to allow for a similar balance in exploration and exploitation.

8.1.2 Results

The final run of RLA with the parameter settings as in Table 8.2 processed 253683 state transitions in approximately 14 days (not all five models were running simultaneously for the entire time period). A total of 800 simulation days with 7 non-zero rewards each, provided the agent with 5600 non-zero rewards. The agent trained on models of the same container terminal with three different QC placements and 20, 24 and 28 vehicles. Figures of the three different QC placements on the simulated terminal are provided in Appendix B.1. We refer to the first scenario (where the QCs are spread over the quay and only two of them are close together) as $CL1$, the second scenario (where QCs are placed close together on the right side of the terminal and further apart on the left) as $CL2$ and the last scenario (where there are two groups of three QCs and one of two QCs) as $CL3$. For all experiments the

Parameter	**Value**
Mini-batch size n	64
Experience replay memory size N	10^5
Discount rate γ	0.999
Learning rate α	0.001
Exploration rate ϵ	$1.0 \rightarrow 0.1$ with a rate of 0.996 every 100 transitions
Target network update frequency τ	1000
Number of hidden layers neural network	2
Number of rectifier units neural network	128
Priority weight β	100
Priority decay rate λ	0.1
Activation function	ReLu
Optimizer	Adam, with default parameters $\beta_1 = 0.9, \beta_2 = 0.999, \epsilon = 10^{-8}$

Table 8.2: Parameter settings for the final version of RLA. Derived from Mnih et al. (2015) and Zhong and Wang (2017), with slight adjustments after conducting preliminary experiments.

storage yard consists of 20 modules and there are 8 active QCs.
The graph in the top left of Figure 8.5 depicts the non-zero rewards that were received during the training of the reinforcement agent. Similarly as in the preliminary experiments, these rewards fluctuate significantly. We do, however, see an increase in average and maximum rewards towards the end of the experiment. During that phase of the training the agent rarely explores actions that are not expected to give the highest cumulative future discounted reward, but exploits the current best actions. During the last phase of the training we see fewer large negative rewards than during the first phase. Presumably, the large negative rewards that are received from simulation hour 4000 onwards are caused by the previously discussed bugs in the simulation model. The larger amount of negative rewards in the beginning of the algorithm could be caused by the decisions of the reinforcement agent.
The upper right graph in Figure 8.5 depicts the loss per learning moment. These values seem to increase towards transition 150000 and decrease slightly afterwards, but remain considerably large. The average Q-values (bottom left of Figure 8.5) and the maximum Q-values (bottom right of Figure 8.5) have increased significantly, when compared to the preliminary experiments. From - approximately - transition 22500 the average seems to stabilize, but there is no way of knowing what would happen if RLA would have the opportunity to continue running. The predicted Q-values are provided in Appendix B.3, combined with the 'best action' per state: the action that is expected to yield the highest cumulative future discounted reward among all possible actions in that state. Combined with the number of state visits - provided in Appendix B.2 - we can derive a number of notable results:

- The predicted Q-values are relatively large (an average of 192 and a maximum of 370). One might expect that with the possibility of negative rewards, at least some of the state-action combinations could yield a negative Q-value.
- Each of the states has been visited, but states with a larger number of visits

have larger differences in Q-values. For states with fewer visits, the agent might not have had the opportunity to experiment with enough different actions and is therefore still approximately indifferent between some of them.

- Action 14, the action that picks a random job, is - in almost all states - the action that provides the lowest cumulative future expected reward.

These results indicate that the reinforcement agent could benefit from an (even) longer training time, in order to provide more convincing distinctions between the different actions in each of the states. On the other hand, states that have not been visited often in these 800 simulation days will also have a small probability of being visited in another simulation experiment and therefore presumably have a minor effect on our final algorithm. We therefore choose to adopt the six best actions without focussing extensively on the exact differences between the - surprisingly high - Q-values.
By assessing the actions that yield the highest Q-values in each of the state we can derive a number of conclusions that concern decision-making on the terminal:

- Actions 3 and 12 are chosen in the large majority of states. Action 3 schedules the job that corresponds to the shortest driving distance and is deemed best in 42 out of the 144 states. Action 12, that focusses on unloading QCs with the fewest number of currently assigned vehicles, seems the best action in 58 states. Together, action 3 and 12 cover 69% of the states.

- Actions 9 and 10 are chosen in the minority of states. Action 9 focusses on load moves and has the highest Q-value in 4 states, while action 10 focusses on loading QCs with the fewest number of currently assigned vehicles and is chosen in 9 states.

- Action 8, that schedules the job that shares an origin or destination with the fewest other jobs (the quietest buffer), is mainly chosen when the percentage of containers that have to be moved to or from a busy place on the terminal is high. In other words: when most containers currently are on - or have to be moved to - busy locations, it seems wise to first schedule containers that do not have to be moved to or from these locations. This preference is independent of varying QC placements.

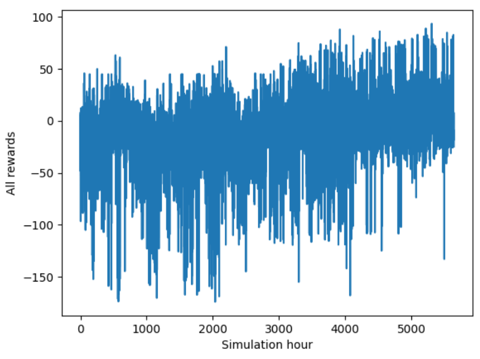
- When less than 25% of the containers that have to be moved within the time horizon has a busy origin or destination actions 3 and 12 are chosen the most. Action 12 (focus on discharging QCs) is more important when a larger percentage of QCs is currently discharging (state attributes $A_1 = 1$ and $A_1 = 2$), while for smaller percentages of unloading QCs actions 12 and 3 (shortest driving distance) are chosen in approximately the same number of states.

- QC clusters diminish the number of states where action 12 is the best action. For the states with attribute $A_4 = 1$, where almost all QCs are further than two cranewidths apart from another QC, action 12 is chosen for 30 out of 48 states. In states with attribute $A_4 = 2$ and $A_4 = 3$ this action is chosen 18 and 9 times, respectively. In these states actions 3 and 6 are more prominent. This indicates that it is less desirable to focus on specific QCs when most QCs are placed in close proximity of each other. Actions 3 and 6 focus on more general terminal characteristics and driving distance, which might decrease the probability of congestion at QCs that are placed in a cluster.

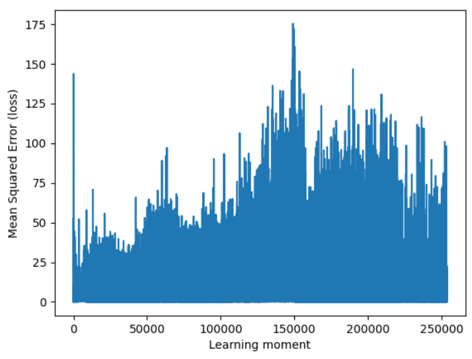
- The parameter *MaxToQC* can be violated in actions 6 and 8 only. This happens in 31 states, mainly when the percentage of containers that need to be moved to or from a busy module or QC exceeds 25. Action 6 dominates when most QCs are loading, while action 8 dominates when most QCs are discharging. It makes sense to assign jobs to the quietest buffer (action 8) when most QCs are discharging, since an unloading job concerns the transport of a container from a discharging QC to a buffer in the yard. After the container is delivered to the buffer, a RMG needs to transport it from the buffer to a place in the storage yard. Sending too many ShCs to a buffer where many unloading jobs are to be delivered to increases the probability that a ShC needs to wait for an RMG to remove a container from the buffer to the storage yard. For loading jobs - that occur more often when more QCs are loading - the RMG has already placed the container in the buffer before it can be assigned to a ShC. Therefore, even though the buffer is defined as busy, the RMGs will not disrupt the pick-up of containers by ShCs and focussing on a busy buffer (action 6) is the best action.

After training the reinforcement agent we run 9 validation experiments for QC placements $CL1$, $CL2$, $CL3$ with 20, 24 and 28 vehicles and 20 replications each. The simulation results on net QC productivity per QC placement are provided in Appendix B.4. The QC productivities are compared with TBAA in order to derive intermediate conclusions about the decision-making of the reinforcement agent. The Figures in Appendix B.4.1 provide information about the net QC productivity per hour: the main objective of this thesis. The net QC productivity is the number of containers that is handled by an active QC per hour, averaged over all conducted experiments (7 productive simulation hours). The only experiments in which RLA outperforms TBAA are the experiments with 24 and 28 vehicles for QC placement $CL2$ and the experiment with 28 vehicles and QC placement $CL3$. The other RLA experiments yield a lower net QC productivity than TBAA. This does not mean that RLA is not successful: the derived scheduling decisions can still prove valuable when implemented in TBAA. RLA makes use of fewer parameters and might thereby miss out on opportunities that TBAA does provide. Combining the best of the two algorithms into DSA can provide desirable results, without the need for implementation of a much more complicated action or state space in RLA.
The Figures in Appendix B.4.2 provide a more detailed description of the net QC productivity per function. A clear difference between RLA and TBAA becomes visible: the twin load QC moves of RLA outperform the twin load moves of TBAA, while the twin discharge QC moves of TBAA outperform the twin discharge QC moves of RLA in each of the experiments. The single load and discharge moves do not differ as much. It is true that RLA picks actions 9 (4 times) and 12 (58 times) that focus on load moves considerably more often than action 10 (9 times) that focusses on unload moves. Generally, loading QCs are bottlenecks as they need to receive the containers in approximately the correct sequence in order to be productive. Discharging QCs do not have to wait for specific containers, but only for a free spot at the TP to place the container they are discharging. It is therefore imaginable to focus more on loading QCs than on discharging QCs. The question remains whether the reinforcement agent (or our defined action space) focusses too

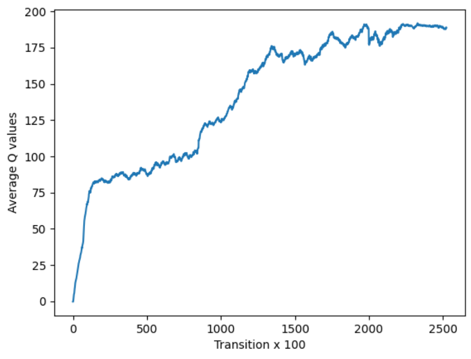
little on discharging QCs or whether the optimal parameter settings as derived by the Taguchi Method will provide improved results. These results are discussed in the next section.



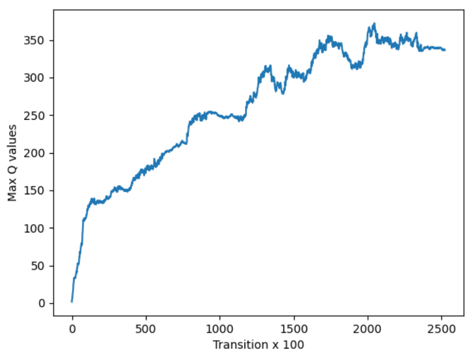
Reward per simulation hour.



Mean Squared Error (loss) per learning moment.



Average predicted Q-values per 100 transitions.



Maximum predicted Q-values per 100 transitions

Figure 8.5: RLA results after 253683 state transitions.

8.2 Results DSA

DSA is the final algorithm that is constructed by implementing the scheduling decisions that were found by RLA in the original scheduling algorithm TBAA. The optimal scheduling parameter settings that were derived by the Taguchi Method, as described in Chapter 7, are implemented and therefore DSA is able to set the scheduling parameters dynamically: for each of the 144 states a specific parameter setting adjusts the scoring algorithm and every state change can hereby result in different job assignments. The additional computational time of DSA when compared to TBAA is negligible, since there are only two minor adjustments: the state needs to be determined and 3 to 4 scoring parameters need to be adjusted. The determination of the state is done by using already existing tables and methods in TIMESQUARE and therefore only requires calculations over relatively small table entries. Multiplying 3 or 4 real numbers is computationally undemanding. The running time of both DSA and TBAA is approximately 1.5 hours for one simulation day consisting of 8 simulation hours.
Similarly as for the validation of RLA we run 9 validation experiments for QC placements $CL1$, $CL2$ and $CL3$ with 20, 24 and 28 ShCs and 20 replications each. We

compare these results to experiments with the same settings for TBAA in Appendix B.5. The graphs in Appendix B.5.1 depict the net QC productivity per hour for each of the three QC placements. It becomes clear that DSA outperforms TBAA in terms of net QC productivity in each of the experiments. Minor increases are seen at QC placement $CL1$ with 20 ShCs (0.2 containers per hour), 24 ShCs (0.4 containers per hour) and at $CL3$ with 20 ShCs (0.3 containers per hour). Larger increases are found at QC placements $CL1$ with 28 ShCs (1.4 containers per hour), $CL2$ with 28 ShCs (1.7 containers per hour) and $CL3$ with 24 ShCs (1.8 containers per hour) and 28 ShCs (2.1 containers per hour). Even though it is understandable that a terminal operator would prefer an algorithm that - without any additional cost, except for the implementation - increases net QC productivity by 2.1 or maybe even by 0.4 containers per hour on average, we test the statistical significance of these results. We conduct a two-sided Welch's t-Test that tests the null hypothesis that the difference between the means of the two compared experiments is zero. The Welch's t-Test, as opposed to the Student's t-Test, does not require the assumption that the variances of the two experiments are approximately equal. As some experiments may include much more twinlift operations than other experiments the average net QC productivities per hour can differ per experiment. The probability that this happens is equal for both algorithms, but it is hard to distinguish between this effect and the effect DSA itself might have on the variance of the objective values with a sample as small as 20. An F-test that tests the null hypothesis that the variances of the two (independent) experiments are equal also would not be able to distinguish between these causes of different variability and even if it would - as Rasch et al. (2011) point out - this is not recommended. Pre-testing by using the same set of observations possibly lead to type I and type II errors (falsely rejecting a true null hypothesis and failing to reject an incorrect null hypothesis, respectively) and it is preferable to use the Welch's t-Test for all experiments, with or without equal variances. The same holds for pre-testing on the normality assumption. We did, out of curiosity, conduct a Shapiro-Wilk test on a number of experiments and always failed to reject the null hypothesis that the data is normally distributed with a confidence level of 0.05. Without using any further pre-tests we calculate the Welch's t-Test statistics and corresponding p-values in R and provide the results in Table 8.3.

QC Placement	**# ShCs**	**t-Test statistic**	**p-value**
CL1	20	1.2024	0.2378
CL1	24	1.5289	0.1356
CL1	28	5.0036	0.0001
CL2	20	4.4667	0.0001
CL2	24	2.2684	0.0318
CL2	28	4.9976	0.0000
CL3	20	1.6325	0.1102
CL3	24	3.1973	0.0044
CL3	28	2.9552	0.0157

Table 8.3: Welch's t-Test for equality of means, without assuming equal variances among the experiments. The test statistics and p-values are reported, and the null hypothesis that the difference between the mean net QC productivities of DSA and TBAA is zero is rejected in 6 out of 9 experiments (with a confidence level of $\alpha = 0.05$).

With a confidence level of $\alpha = 0.05$ we can reject the null hypothesis that the differences between the mean net QC productivities of TBAA and DSA are zero for 6 out of 9 experiments. The only experiments that do not result in a statistically significant increase in QC productivity are the experiments with QC placement $CL1$ and 20 and 24 ShCs and the experiment with QC placement $CL3$ and 20 ShCs. This indicates that DSA outperforms TBAA when the number of ShCs is larger, which was a clear result of RLA (Appendix B.4) as well. It also indicates that the implementation of the Taguchi Method in order to combine TBAA and RLA in an efficient way had a positive effect on the final results: DSA does not only outperform TBAA, but also RLA and therefore seems capable of combining the desirable properties of both algorithms. This becomes even more clear when inspecting the results in Appendix B.5.2, where we depict the net QC productivity per function for each of the experiments. As discussed in Section 8.1, RLA seems to outperform TBAA in twin load moves, while it underperforms in twin discharge moves. DSA is capable of diminishing this effect: while still outperforming TBAA in twin load moves, it rarely - and if so, only slightly - underperforms in twin discharge moves. The differences between DSA and TBAA in productivity of twin load moves is larger for experiments with 3 and 3.5 ShCs per QC than for experiments with 2.5 ShCs per QC. Appendix B.5.3 depicts the average ShC status as a percentage of time for each of the experiments. It becomes clear that ShCs in DSA spend a slightly smaller fraction of the time driving empty in QC placements $CL1$ and $CL3$. Interestingly, they spend a slightly larger fraction of the time driving empty in QC placement $CL2$. The graphs in Appendix B.5.4 depict the average QC status as a percentage of time. The graphs in B.5.1 already indicate an increase in net QC productivity when comparing DSA to TBAA. A higher net QC productivity is, however, not always a clear indicator of QC productivity. If, for example, all QCs in one experiment only perform twinlift operations and in another experiment they only perform single lift operations the first experiment would yield a twice as large net QC productivity, even when all QCs in both experiments were productive for 85% of the time. We safeguard against these differences by conducting 20 experiments and by specifying the probability distribution of each functionality for all QCs in TIMESQUARE. Each of the QCs performs twin load operations with a probability of 0.315, single load operations with a probability of 0.185 and twin discharge and single discharge operations with probabilities 0.315 and 0.185, respectively. After running 20 experiments we inspect the results and remove and/or add experiments when the desired distribution was not yet reached. Therefore, net QC productivity is a reliable indicator of an improved algorithm, which is underlined by the results in Appendix B.5.4: QC productivity as a percentage of time increases for each of the experiments when comparing DSA to TBAA. Furthermore, QCs wait a smaller fraction of time during loading, when comparing DSA to TBAA. This underlines the results in Appendix B.5.2, where we see an increase in net QC productivity, especially for twin load moves and slightly for load moves. Furthermore, TBA considers QCs with a productivity of $80 - 85\%$ as highly productive: this is their target level when conducting experiments. We reach this target level in most experiments with 3 and 3.5 ShCs per QC, but not in experiments with 2.5 ShCs per QC. This could indicate that 20 ShCs for 8 QCs is not enough to reach a higher QC productivity as the ShCs are already working on their maximum productive level and seem to be the bottleneck in the corresponding experiments, impossible to overcome by both DSA as well as TBAA.

8.3 Sensitivity analysis

In order to test whether the scheduling decisions that were derived by RLA and implemented in DSA still yield improved QC productivities for situations that the reinforcement agent has not trained on we conduct additional experiments with an off-peak simulation in which the terminal volume is heavily decreased. In this off-peak simulation 4 QCs are active, the yard density decreases from 65% to 50% and the number of trucks that arrive at the landside of the terminal decreases by 50%. We conduct 20 experiments with 10, 12 and 14 ShCs for both TBAA as well as DSA. This corresponds to 2.5, 3 and 3.5 ShCs per quay crane, similarly as in the previously discussed experiments. The runtime of these simulations is 40 minutes per simulation day of 8 simulation hours. A figure of the simulated container terminal is provided in Appendix B.1 and the results in Appendix B.6.
In Appendix B.6 the results of the off-peak simulation indicate that DSA outperforms TBAA in terms of QC productivity for the experiments with 12 and 14 ShCs with an increase in net QC productivity of 0.9 containers per hour. The experiment with 10 ShCs yields the same net QC productivity for TBAA and DSA when rounded to one decimal. The results on QC productivity per function show similar results as in Section 8.2: the increased QC productivities are mainly the result of an increased twin load QC productivity of DSA. In these off-peak experiments, however, twin discharge moves of DSA slightly outperform twin discharge moves of TBAA as well, but only for the experiments with 3 and 3.5 ShCs per QC. The ShC status for both algorithms are comparable: there are no large differences in laden and empty driving statuses between the two algorithms, except for the experiment with 14 ShCs. In this experiment, ShCs of DSA seem to spend a larger fraction of their time driving (both laden and empty) than ShCs of TBAA. They do spend less time at a QC, but wait slightly more before they can approach a QC. The QC status, however, does show larger differences between the two algorithms. QCs are productive 84.2% (12 ShCs) and 87.3% (14 ShCs) of the time after implementing DSA, compared to 81.0% and 85.9% for TBAA. Note that these QCs productivities are considered very high, both for DSA as well as TBAA. Even though the increases in net QC productivities are smaller than in the scenarios the reinforcement agent has trained on, productivities of this magnitude are highly unlikely to be surpassed by a change in ShC schedules. We do see that the ShCs are rarely idle for any of the off-peak experiments. When we inspect the ShC status of the peak simulations in Appendix B.5.3 we do see that the ShCs are idle for a small fraction of time in experiments with 3.5 ShCs per QC. This could indicate that employing more ShCs might be beneficial in this off-peak scenario, but - as previously stated - an average QC productivity of 87% is unlikely to increase significantly.
We do see a higher QC productivity in the experiment with 10 ShCs for TBAA than for DSA, even though the rounded net QC productivities were exactly the same. It should be noted that TBAA outperformed DSA by 0.04 containers per hour. This - combined with slightly more twin-lift moves in the DSA experiments and more single-lift moves in the TBAA experiments - results in a slightly higher QC productivity. On the basis of these results we can therefore conclude that TBAA slightly - but possibly statistically insignificantly - outperforms DSA in terms of QC productivity for the off-peak experiments with 10 ShCs.
Similarly as in Section 8.2 we assess the statistical significance of the results on net QC productivity and conduct a two-sided Welch's t-Test that tests the null hypoth-

esis that the difference between the mean net QC productivities of TBAA and DSA The resulting test statistics and p-values are provided in Table 8.4 and indicate that the increase in QC productivity for the off-peak experiments with 12 and 14 ShCs is statistically significant at a confidence level of $\alpha = 0.05$.

# ShCs	t-Test statistic	p-value
10	0.2007	0.8422
12	2.0601	0.0490
14	2.1398	0.0388

Table 8.4: Welch's t-Test for equality of means, without assuming equal variances among the experiments. The test statistics and p-values are reported, and the null hypothesis that the difference between the mean net QC productivities of DSA and TBAA for the off-peak experiments is zero is rejected in 2 out of 3 experiments (with a confidence level of $\alpha = 0.05$).

To conclude we can therefore state that DSA is robust to the considered change in daily terminal volume and significantly outperforms TBAA in terms of net QC productivity in experiments with 3 and 3.5 ShCs per QC. The increase in net QC productivity is smaller than on peak simulation days but a productive status of 87.3% is considered extremely high and possibly impossible to increase by employing different ShC scheduling decisions.

Chapter 9

Conclusion

In this final chapter we provide a brief summary of the algorithms and results that were discussed in this thesis in Section 9.1, followed by a description of our main conclusions in Section 9.2. The last section of this thesis, Section 9.3 focusses on recommendations for future research.

9.1 Summary

The goal of this thesis was to implement a dynamic scheduling algorithm in the container terminal simulation software of TBA in order to increase QC - and thereby container terminal - productivity. The current algorithm, TBAA, applies the exact same scoring algorithm throughout an entire simulation experiment and is therefore considered a static algorithm. A dynamic algorithm would involve adjusting this scoring mechanism to different situations that occur on the container terminal. The choice was made to focus on the scheduling of ShCs at the waterside of a simulated version of the Hadarom Container Terminal (HCT) in Israel, but RLA can - with minor adjustments - be implemented in simulation models with other terminal types as well.
After preliminary inspection of TBAA, along with extensive consultation of several simulation and scheduling experts at TBA, a number of situations (states) and possible corresponding parameter adjustments (actions) were defined. These formed the basis for RLA: a reinforcement learning algorithm that learns to take actions that yield the highest cumulative expected future discounted reward in each of the defined states. Since the effect of an action on QC productivity is not immediately visible rewards are given once per simulation hour, resulting in delayed and sparse rewards. Furthermore, due to the relatively long computational time of the simulation program, the amount of data that could be processed was limited. A thorough literature research on these two challenges resulted in the selection of three existing reinforcement learning algorithms: Deep Q-Network with Experience Replay, Reward Backpropagation Prioritized Experience Replay and the Double Deep Q-Network with Experience Replay. The existing algorithms were combined into RLA and the reinforcement agent was trained on the simulated container terminal with three different QC placements and different numbers of ShCs in order to ensure robustness of the results.
By itself, RLA does not outperform TBAA in terms of net QC productivity in most experiments, as it mostly focusses on additional scheduling decisions for the spec-

ified states without taking into account the already acquired knowledge that the simulation experts of TBA implemented in TBAA. Therefore, the Taguchi Method was used to merge the results of RLA into the already existing algorithm TBAA. By conducting 25 experiments with different scoring parameters for each of the experiments, the parameter settings with the highest signal-to-noise ratio were selected and implemented in the final algorithm DSA.
With negligible additional computational time, DSA outperforms TBAA in terms of net QC productivity in each of the conducted experiments. Statistically significant increased QC productivities are mostly found in experiments with 3 and 3.5 ShCs per QC, and especially (twin) load QC moves experience an increased productivity when comparing DSA to TBAA.
Additional experiments were conducted to test the robustness of DSA to a decrease in daily container terminal volume and the results indicate that DSA significantly outperforms TBAA in terms of net QC productivity for experiments with 3 and 3.5 ShCs per QC on off-peak simulation days.

9.2 Conclusions

The three scheduling algorithms that are considered in this thesis are:

- TBAA: the existing scheduling algorithm of TBA, that uses approximately 30 parameters in the simulation model under consideration to score jobs (each job is a combination of a ShC and a container that needs to be transported from the yard to a QC or vice versa). The job with the highest score is assigned.
- RLA: the reinforcement learning algorithm that was implemented especially for this thesis and uses approximately 10 parameters in order to decide which of the 14 specified actions to take in each of the 144 defined states of the terminal.
- DSA: similar to TBAA, but with the implementation of the derived scheduling decisions by RLA. According to the actions that were specified by RLA the scheduling parameters of TBAA (with three additional scoring parameters) are dynamically updated in each of the 144 states.

The simulation results of the validation experiments of RLA indicate a decrease in net QC productivity in most experiments, when compared to TBAA. The only experiments in which RLA outperforms TBAA are 1 out of 3 experiments with 24 ShCs and 2 out of 3 experiments with 28 ShCs (3 out of 9 experiments in total).
A possible reason for this decrease in QC productivity is that RLA does not take all scoring parameters of TBAA into account and could thereby miss out on valuable differences between jobs. This issue might be solved by expanding the state and/or action space of RLA such that the underlying MDP of the container terminal is no longer partially observable - or slightly more observable, but a computationally less exhaustive and therefore possible more efficient solution is to merge the results of RLA and TBAA.
The scoring parameters are adjusted in order to give both RLA and TBAA the opportunity to influence the scheduling decision making and hereby extract the best of the two algorithms. By applying the Taguchi Method on multiplications of the (existing and added) scheduling parameters the final parameter settings for each of the states were found within 25 experiments of 24 replications each.

The results of DSA indicate a statistically significant increase in net QC productivity for 6 out of 9 experiments, particularly for experiments with 3 and 3.5 ShCs per QC. The increases in QC productivity range from 0.2 containers per hour for an experiment with 2.5 ShCs per QC to 2.1 containers per hour for an experiment with 3.5 ShCs per QC. In 5 out of the 9 experiments QC productivities of $80 - 85\%$ were reached: productivities that are considered (near-)optimal by TBA simulation consultants.
The performance of DSA is shown to be robust to a decrease in daily terminal volume and significantly outperforms TBAA in experiments with 3 and 3.5 ShCs per QC on off-peak days. TBAA slightly - but statistically insignificantly - outperforms DSA in terms of net QC productivity for the off-peak experiment with 2.5 ShCs per QC. Generally, the experiments with 2.5 ShCs per QC do not result in improved QC productivities, which could indicate that the number of ShCs is too small to provide the required service and form a bottleneck that cannot be overcome by TBAA nor DSA.
We cannot exclude the possibility that a different parameter setting and/or state and action space for RLA results in different scheduling decisions and considerably higher or lower QC productivities, but as QC productivities of $80 - 85\%$ are considered sufficiently high by TBA simulation consultants we can safely assume that in 8 out of 12 experiments DSA performs satisfactory. We can therefore conclude that the combination of RLA with the Taguchi Method into DSA produces desirable results. The implemented dynamic scheduling decisions show an increased QC - and therefore terminal - productivity, while adding negligible computational time.

9.3 Recommendations for future research

On the basis of the experiments discussed in this thesis we can define a number of recommendations for both TBA as well as other researchers that could be of interest for future research.
First of all, it should be noted that the results of the reinforcement learning algorithm in this thesis are highly dependent on the algorithmic and parametric choices that were made. As this was the first attempt to implement a reinforcement learning algorithm - and any machine learning algorithm - to make scheduling decisions on the simulated container terminal in TIMESQUARE, future research could include a more thorough analysis of the parameters of the algorithm and save time by making use of our written codes.
Furthermore, neural networks are capable of dealing with larger state spaces than the 144 states we addressed, which could influence the final results in a beneficial manner. There is, however, a trade-off between increasing the state space and the simplicity of the final algorithm. As TBA's simulation experts still want to be able to influence the scheduling decisions it might not be desirable to have much more than 144 states to take into account.
The sensitivity analysis in this thesis includes one set of experiments with off-peak simulations, but additional experiments should be conducted in order to thoroughly analyse the final performance of DSA on different versions of the simulation model. In the construction of RLA three different QC placements and various numbers of ShCs were taken into account, but it could be of interest to test larger versions of the simulated container terminal. This would require significant changes to the simulation model in terms of routing, size of the storage yard and an additional

side quay for extra QCs and is therefore omitted for this thesis. We advise TBA to conduct these experiments before implementing DSA in their simulation models.
The Python codes for the reinforcement learning algorithm were written as generically as possible and can therefore relatively easily be implemented in different container terminal models. It would be interesting to train the reinforcement agent on a model with different types of HTVs or a different lay-out in order to derive model-specific scheduling decisions.
Both TBAA as well as DSA are uncoordinated scheduling algorithms: the scheduling decisions of different equipment types are decoupled. By implementing a (possibly multi-agent) reinforcement model that addresses the different scheduling problems simultaneously scheduling decisions might experience an increased efficiency. Furthermore, including more information in the state space of the reinforcement learning algorithm would move the framework from a POMDP in the direction of an MDP, where reinforcement learning is mostly applied and convergence of the results often guaranteed. Note, however, that the implementation of the scheduling decisions that result from such a coordinated scheduling algorithm in the simulation program of TBA would be more complicated than the implementation of uncoordinated scheduling decisions, as was done in this thesis. It could therefore be more desirable to initially apply the methods in this thesis to another scheduling problem on the same simulated container terminal, like for example the scheduling of RMGs in the storage yard. Slight adjustments in the state and action space would be sufficient to run the experiments in this thesis in order to optimize a different part of the container terminal.
The Taguchi Method was applied on experiments with 24 ShCs, in order to limit the number of replications needed for each of the 25 experiments. Furthermore - as becomes clear from the results of both DSA and RLA - varying the number of ShCs in an experiment has a significant effect on net QC productivity and would therefore increase the variance of each of the experiments. Including different numbers of ShCs in one Taguchi experiment might therefore not be desirable, but one could conduct additional experiments with different numbers of ShCs on the terminal. In other words, the Taguchi Method could be applied three separate times in order to find the most desirable parameter settings for each of the scenarios. That might improve the performance of DSA in experiments with 2.5 ShCs per QC - the experiments in which DSA currently shows no significant improvement when compared to TBAA. Furthermore, combining three to four scheduling parameter settings into one of the six parameters that are assessed by the Taguchi Method eliminates the relative effect these scoring parameters might have on each other. The levels were determined by preliminary inspection of the schedules that resulted of various changes to the scheduling parameters, but in fact an additional Taguchi experiment could include an analysis of the relative effects of these scoring parameters.
Lastly, all experiments in this thesis concern a simulated container terminal in TIMESQUARE, but a connection to the real-life setting at the HCT terminal in Israel, where the TBA Scheduler makes the scheduling decisions, already exists. In order to investigate whether the derived dynamic scheduling decisions in this thesis still hold in real-life, DSA could be implemented in the TBA Scheduler.

Bibliography

Agboola, O., Sadiku, R., Mokrani, T., Amer, I., and Imoru, O. (2017). Polyolefins and the environment. In *Polyolefin Fibres*, pages 89–133. Woodhead Publishing, second edition.

Ahmed, E., El-Abbasy, M. S., Zayed, T., Alfalah, G., and Alkass, S. (2021). Synchronized scheduling model for container terminals using simulated double-cycling strategy. *Computers and Industrial Engineering*, 154:107–118.

Alibrahim, H. and Ludwig, S. A. (2021). Hyperparameter optimization: Comparing genetic algorithm against grid search and bayesian optimization. *Proceedings of the 2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 1551–1559.

Bartz-Beielstein, T. and Markon, S. (2004). Tuning search algorithms for real-world applications: a regression tree based approach. *Proceedings of the 2004 IEEE Congress on Evolutionary Computation (CEC)*, pages 1111–1118.

Baykal-Gürsoy, M. and Gürsoy, K. (2007). Semi-markov decision processes. *Probability in the Engineering and Informational Sciences*, 21:635 – 657.

Bellman, R. (1957). *Dynamic Programming.* Princeton University Press, Princeton, NJ, USA, first edition.

Bergstra, J. and Bengio, Y. (2012). Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 13:281–305.

Bierwirth, C. and Meisel, F. (2010). A survey of berth allocation and quay crane scheduling problems in container terminals. *European Journal of Operational Research*, 202(3):615–627.

Box, G., Hunter, W., and Hunter, J. (1978). Statistics for experimenters. *Wiley series in probability and mathematical statistics: Applied probability and statistics.*

Bradtke, S. J. and Duff, M. O. (1994). Reinforcement learning methods for continuous-time markov decision problems. *Proceedings of the 7th International Conference on Neural Information Processing Systems*, page 393–400.

Brauer, W. and Weiss, G. (1998). Multi-machine scheduling-a multi-agent learning approach. *Proceedings of the 1998 International Conference on Multi Agent Systems (Cat. No.98EX160)*, pages 42–48.

Cai, B., Huang, S., Liu, D., Yuan, S., Dissanayake, G., Lau, H., and Pagac, D. (2013). Multiobjective optimization for autonomous straddle carrier scheduling at automated container terminals. *IEEE Transactions on Automation Science and Engineering*, 10(3):711–725.

Candan, G. and Yazgan, H. (2014). Genetic algorithm parameter optimization using taguchi method for a flexible manufacturing system scheduling problem. *International Journal of Production Research*, 53.

Cao, J., Shi, Q., and Lee, D.-H. (2010). Integrated quay crane and yard truck schedule problem in container terminals. *Tsinghua Science and Technology*, 15(4):467–474.

Carpenter, B. C. and Ward, T. (1990). The use of computer simulation for marine terminal planning. *Proceedings of the 1990 Winter Simulation Conference*, pages 802–804.

Chan, S., Treleaven, P., and Capra, L. (2013). Continuous hyperparameter optimization for large-scale recommender systems. *Proceedings of the 2013 IEEE International Conference on Big Data*, pages 350–358.

Chen, L., Bostel, N., Dejax, P., Cai, J., and Xi, L. (2007). A tabu search algorithm for the integrated scheduling problem of container handling systems in a maritime terminal. *European Journal of Operational Research*, 181(1):40–58.

Chen, L., Langevin, A., and Lu, Z. (2013). Integrated scheduling of crane handling and truck transportation in a maritime container terminal. *European Journal of Operational Research*, 225(1):142–152.

Csáji, B. C., Monostori, L., and Kádár, B. (2006). Reinforcement learning in a distributed market-based production control system. *Advanced Engineering Informatics*, 20(3):279–288. Design of Complex Adaptive Systems.

Diabat, A. and Theodorou, E. (2014). An integrated quay crane assignment and scheduling problem. *Computers and Industrial Engineering*, 73:115–123.

Ding, X. C., Wang, J., Lahijanian, M., Paschalidis, I. C., and Belta, C. A. (2012). Temporal logic motion control using actor-critic methods. *Proceedings of the 2012 IEEE International Conference on Robotics and Automation*, pages 4687–4692.

Doerr, O. and Sánchez, R. J. (2006). Indicadores de productividad para la industria portuaria. aplicación en américa latina y el caribe. *Recursos Naturales e Infraestructura*, 112.

Dulebenets, M. A. (2018). Application of evolutionary computation for berth scheduling at marine container terminals: Parameter tuning versus parameter control. *IEEE Transactions on Intelligent Transportation Systems*, 19(1):25–37.

Fotuhi, F., Huynh, N., Vidal, J. M., and Xie, Y. (2013). Modeling yard crane operators as reinforcement learning agents. *Research in Transportation Economics*, 42(1):3–12.

Gambardella, L. M., Rizzoli, A. E., and Zaffalon, M. (1998). Simulation and planning of an intermodal container terminal. *Harbour and Maritime Simulation*, 71.

Gaon, M. and Brafman, R. I. (2019). Reinforcement learning with non-markovian rewards. *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence.*

Geoffrion, A. (1999). Structured modeling: survey and future research directions. *Interactive Transactions of OR/MS*, 1(3).

Giannoccaro, I. and Pontrandolfo, P. (2002). Inventory management in supply chains: a reinforcement learning approach. *International Journal of Production Economics*, 78(2):153–161.

Hare, J. (2019). Dealing with sparse rewards in reinforcement learning. *ArXiv*, abs/1910.09281.

Hedayat, A. S., Sloane, N. J. A., and Stufken, J. (1999). *Tables of Orthogonal Arrays*, pages 317–339. Orthogonal Arrays: Theory and Applications. Springer New York.

Hirashima, Y. (2018). A reinforcement learning with group-based candidate-extraction for container marshalling at marine ports. *Proceedings of the International MultiConference of Engineers and Computer Scientists.*

Hirashima, Y., Takeda, K., and Inoue, A. (2004). A container transfer scheduling using reinforcement learning. *IEEE Transactions on Industry Applications*, 123:1111–1117.

Hu, X., Guo, J., and Zhang, Y. (2019). Optimal strategies for the yard truck scheduling in container terminal with the consideration of container clusters. *Computers and Industrial Engineering*, 137.

Jilani, T., Zaidi, F., Khalid, S., Maqsood, A., and Safdar, S. (2019). Asset allocation using markov decision process. *International Journal of Computer Science and Network Security*, 19(3):185–189.

Jomaa, H. S., Grabocka, J., and Schmidt-Thieme, L. (2019). Hyp-rl : Hyperparameter optimization by reinforcement learning. *ArXiv*, abs/1906.11527.

Jonker, T. (2018). Coordinated optimization of equipment operations on a container terminal. *Delft University Thesis.*

Kaelbling, L. P., Littman, M. L., and Cassandra, A. R. (1998). Planning and acting in partially observable stochastic domains. *Artificial Intelligence*, 101(1):99–134.

Kavoosi, M., Dulebenets, M., Abioye, O., Pasha, J., Theophilus, O., Wang, H., Kampmann, R., and Mikijeljević, M. (2019). Berth scheduling at marine container terminals: A universal island-based metaheuristic approach. *Maritime Business Review*, 5(1):30–66.

Kim, K. H., Lee, K. M., and Hwang, H. (2003). Sequencing delivery and receiving operations for yard cranes in port container terminals. *International Journal of Production Economics*, 84(3):283–292.

Kim, K. H. and Park, Y.-M. (2004). A crane scheduling method for port container terminals. *European Journal of Operational Research*, 156(3):752–768.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. *Proceedings of the 3rd International Conference for Learning Representations.*

Krishnaiah, K. and Shahabudeen, P. (2012). *Applied Design of Experiments and Taguchi Methods*. Asoke K. Ghosh.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Communications of the Association for Computing Machinery*, 60:84–90.

Lee, D.-H., Wang, H. Q., and Miao, L. (2008). Quay crane scheduling with non-interference constraints in port container terminals. *Transportation Research Part E: Logistics and Transportation Review*, 44(1):124–135.

Leriche, D., Oudani, M., Cabani, A., Hoblos, G., Mouzna, J., Boukachour, J., and El Hilali Alaoui, A. (2016). A simulation-optimisation study for comparison of new logistics systems at le havre port. *Proceedings of the 3rd International Conference on Logistics Operations Management (GOL)*, pages 1–6.

Li, X., Wang, J., and Sawhney, R. (2012). Reinforcement learning for joint pricing, lead-time and scheduling decisions in make-to-order systems. *European Journal of Operational Research*, 221(1):99–109.

Lin, L.-J. (1992). *Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching*, pages 69–97. Reinforcement Learning. Springer US.

Luo, J., Wu, Y., Halldorsson, A., and Song, X. (2011). Storage and stacking logistics problems in container terminals. *OR Insight*, 24:256–275.

Masters, D. and Luschi, C. (2018). Revisiting small batch training for deep neural networks. *Graphcore Research*.

McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5:115–133.

Mead, A. C. and Ismail, M. (1989). Analog vlsi implementation of neural systems. *The Kluwer International Series in Engineering and Computer Science*, 80.

Meng, L., Gorbet, R., and Kulic, D. (2021). Memory-based deep reinforcement learning for POMDP. *Proceedings of the 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, page 5619–5626.

Minsky, M. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry*. MIT Press.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., and Riedmiller, M. (2013). Playing atari with deep reinforcement learning. *ArXiv*, abs/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., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540).

Moore, A. W. and Atkeson, C. G. (1993). Prioritized sweeping : Reinforcement learning with less data and less real time. *Machine Learning*, 13:103–130.

Nagurney, A. (2021). Global shortage of shipping containers highlights their importance in getting goods to amazon warehouses, store shelves and your door in time for christmas. Retreived from https://theconversation.com; March 9, 2022.

Ng, W. and Mak, K. (2005). Yard crane scheduling in port container terminals. *Applied Mathematical Modelling*, 29(3):263–276.

Notteboom, T., Pallis, A., and Rodrigue, J.-P. (2020). Disruptions and resilience in global container shipping and ports: Covid-19 pandemic vs. 2008-2009 financial crisis. *Maritime Economics and Logistics*, 23:179–210.

Petrov, E. and Kharina, N. (2018). Markov processes in image processing. *Journal of Physics: Conference Series*, 1015.

Placek, M. (2021). Container shipping statistics and facts. Retreived from https://www.statista.com/topics/1367/container-shipping; March 8, 2022.

Rasch, D., Kubinger, K., and Moder, K. (2011). The two-sample t test: Pre-testing its assumptions does not pay off. *Statistical Papers*, 52:219–231.

Rashidi, H. and Tsang, E. P. (2011). A complete and an incomplete algorithm for automated guided vehicle scheduling in container terminals. *Computers and Mathematics with Applications*, 61(3):630–641.

Rincón-Zapatero, J. and C., R.-P. (2003). Existence and uniqueness of solutions to the bellman equation in the unbounded case. *Econometrica*, 71:1519–1555.

Saanen, Y. (2004). An approach for designing robotized marine container terminals. *PhD Thesis Delft University.*

Safaeian, M., Etebari, F., and Vahdani, B. (2021). An integrated quay crane assignment and scheduling problem with several contractors in container terminals. *Scientia Iranica*, 28(2):1030–1048.

Said, G. A. E.-N. A., Mahmoud, A. M., and El-Horbaty, E.-S. M. (2014). Simulation and optimization of container terminal operations: a case study. *International Journal of Computer Science Engineering and Information Technology Research*, 4:85–94.

Schaul, T., Quan, J., Antonoglou, I., and Silver, D. (2015). Prioritized experience replay. *Proceedings of the 4th International Conference on Learning Representations.*

Schmidhuber, J. (1992). Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4.

Semendiak, Y. (2020). From parameter tuning to dynamic heuristic selection. *Master Thesis Technische Universität Dresden.*

Seyedalizadeh Ganji, S. and Javanshir, H. (2010). Yard crane scheduling in port container terminals using genetic algorithm. *Journal of Industrial Engineering International*, 6.

Shim, H.-J. and Kim, J.-K. (2009). Cause of failure and optimization of a v-belt pulley considering fatigue life uncertainty in automotive applications. *Engineering Failure Analysis*, 16(6):1955–1963.

Skaf, A., Lamrous, S., Hammoudan, Z., and Manier, M.-A. (2021). Integrated quay crane and yard truck scheduling problem at port of tripoli-lebanon. *Computers and Industrial Engineering*, 159.

Smit, S. and Eiben, A. (2009). Comparing parameter tuning methods for evolutionary algorithms. *Proceedings of the 2009 IEEE Congress on Evolutionary Computation*, pages 399–406.

Smith, J. E. and Winkler, R. L. (2006). The optimizer's curse: Skepticism and postdecision surprise in decision analysis. *Management Science*, 52(3):311–322.

Steenken, D., Voß, S., and Stahlbock, R. (2004). Container terminal operation and operations research - a classification and literature review. *OR Spectrum*, 26:3–49.

Stockheim, T., Schwind, M., and Koenig, W. (2003). A reinforcement learning approach for supply chain management. *First European Workshop on MultiAgent Systems*, 40(6):1299–1317.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement Learning: An Introduction.* MIT Press, second edition.

Taguchi, G., Chowdhury, S., and Wu, Y. (2004). *Taguchi's Quality Engineering Handbook.* John Wiley & Sons, Ltd.

Uray, E., Carbas, S., Geem, Z. W., and Kim, S. (2022). Parameters optimization of taguchi method integrated hybrid harmony search algorithm for engineering design problems. *Mathematics*, 10(3).

van Handel, R. (2016). Probability and random processes. *Lecture Notes Princeton University.*

van Hasselt, H. (2010). Double q-learning. *Advances in Neural Information Processing Systems*, 23:2613–2621.

van Hasselt, H., Guez, A., and Silver, D. (2016). Deep reinforcement learning with double q-learning. *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence.*

van Leeuwaarden, J. (2020). Stochastic operations research models. *Lecture Notes Tilburg University.*

van Zijderveld, E. (1995). A structured terminal design method: With a focus on rail container terminals. *Doctoral Thesis Delft University.*

Walsh, T. J., Nouri, A., Li, L., and Littman, M. L. (2007). Learning and planning in environments with delayed feedback. *Autonomous Agent Multi-Agent Systems*, 18(83).

Wang, H., Geng, Q., and Qiao, Z. (2014). Parameter tuning of particle swarm optimization by using taguchi method and its application to motor design. *Proceedings of the 4th IEEE International Conference on Information Science and Technology*, pages 722–726.

Wang, Y.-C. and Usher, J. (2005). Application of reinforcement learning for agent-based production scheduling. *Engineering Applications of Artificial Intelligence*, 18:73–82.

Wang, Y.-C. and Usher, J. (2007). A reinforcement learning approach for developing routing policies in multi-agent production scheduling. *The International Journal of Advanced Manufacturing Technology*, 33:323–333.

Watkins, C. J. C. H. (1989). *Learning from delayed rewards.* Cambridge University.

Wolpert, D. and Macready, W. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82.

Woolf, P., Fraley, S., Zalewski, J., Oom, M., and Terrien, B. (2020). *Chemical Process Dynamics and Control*, chapter 14. Design of Experiments via Taguchi Methods. University of Michigan.

Wuest, T., Weimer, D., Irgens, C., and Thoben, K.-D. (2016). Machine learning in manufacturing: advantages, challenges, and applications. *Production & Manufacturing Research*, 4(1):23–45.

Zeng, Q. and Yang, Z. (2009). Integrating simulation and optimization to schedule loading operations in container terminals. *Computers and Operations Research*, 36(6):1935–1944.

Zhang, Z., Zheng, L., Li, N., Wang, W., Zhong, S., and Hu, K. (2012). Minimizing mean weighted tardiness in unrelated parallel machine scheduling with reinforcement learning. *Computers & Operations Research*, 39(7):1315–1324.

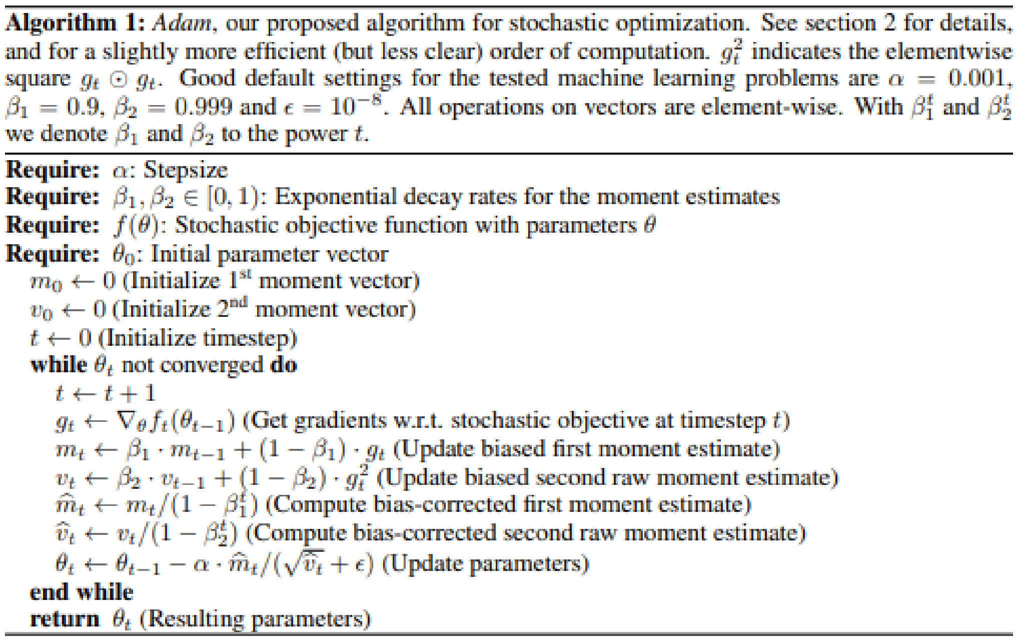
Zhao, Y., Wang, Y., Tan, Y., Zhang, J., and Yu, H. (2021). Dynamic jobshop scheduling algorithm based on deep q network. *IEEE Access*, 9:122995–123011.

Zhong, Y. and Wang, B. (2017). Reward backpropagation prioritized experience replay. *Stanford University Research.*

Appendix A

Auxiliary Figures and Tables

A.1 Pseudocode Adam optimizer



Algorithm 1: *Adam*, our proposed algorithm for stochastic optimization. See section 2 for details, and for a slightly more efficient (but less clear) order of computation. g_t^2 indicates the elementwise square $g_t \odot g_t$. Good default settings for the tested machine learning problems are $\alpha = 0.001$, $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. All operations on vectors are element-wise. With β_1^t and β_2^t we denote β_1 and β_2 to the power t.

```
Require: $\alpha$: Stepsize
Require: $\beta_1, \beta_2 \in [0, 1)$: Exponential decay rates for the moment estimates
Require: $f(\theta)$: Stochastic objective function with parameters $\theta$
Require: $\theta_0$: Initial parameter vector
  $m_0 \leftarrow 0$ (Initialize 1st moment vector)
  $v_0 \leftarrow 0$ (Initialize 2nd moment vector)
  $t \leftarrow 0$ (Initialize timestep)
  while $\theta_t$ not converged do
    $t \leftarrow t + 1$
    $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Get gradients w.r.t. stochastic objective at timestep $t$)
    $m_t \leftarrow \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$ (Update biased first moment estimate)
    $v_t \leftarrow \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$ (Update biased second raw moment estimate)
    $\widehat{m}_t \leftarrow m_t / (1 - \beta_1^t)$ (Compute bias-corrected first moment estimate)
    $\widehat{v}_t \leftarrow v_t / (1 - \beta_2^t)$ (Compute bias-corrected second raw moment estimate)
    $\theta_t \leftarrow \theta_{t-1} - \alpha \cdot \widehat{m}_t / (\sqrt{\widehat{v}_t} + \epsilon)$ (Update parameters)
  end while
  return $\theta_t$ (Resulting parameters)
```

Figure A.1: Pseudocode Adam optimizer, as proposed by Kingma and Ba (2015). Notice that the authors refer to α as the stepsize parameter instead of the learning rate. The only difference is the terminology, the function of α remains as discussed in this thesis.

A.2 Orthogonal array $L25$

Experiment	A	B	C	D	E	F
1	1	1	1	1	1	1
2	1	2	2	2	2	2
3	1	3	3	3	3	3
4	1	4	4	4	4	4
5	1	5	5	5	5	5
6	2	1	2	3	4	5
7	2	2	3	4	5	1
8	2	3	4	5	1	2
9	2	4	5	1	2	3
10	2	5	1	2	3	4
11	3	1	3	5	2	4
12	3	2	4	1	3	5
13	3	3	5	2	4	1
14	3	4	1	3	5	2
15	3	5	2	4	1	3
16	4	1	4	2	5	3
17	4	2	5	3	1	4
18	4	3	1	4	2	5
19	4	4	2	5	3	1
20	4	5	3	1	4	2
21	5	1	5	4	3	2
22	5	2	1	5	4	3
23	5	3	2	1	5	4
24	5	4	3	2	1	5
25	5	5	4	3	2	1

Table A.1: Orthogonal array $L25$, with 25 experiments (rows), six parameters (columns) and five different parameter levels that are specified for each of the experiments in the table. Copied from Hedayat et al. (1999).

A.3 Parameters Taguchi Method

Parameter	**Label**	**Level 1**	**Level 2**	**Level 3**	**Level 4**	**Level 5**
Action 3: SER	A	[1, 1, 1]	[1.25, 1.125, 1.0625]	[1.5, 1.25, 1.125]	[2, 1.5, 1.25]	[100, 50, 25]
Action 6: BBN	B	[-1, 1, 1]	[-1.25, 1.125, 1.0625]	[1.5, 1.25, 1.125]	[2, 1.5, 1.25]	[100, 50, 25]
Action 8: QBN	C	[1, 1, 1]	[1.25, 1.125, 1.0625]	[1.5, 1.25, 1.125]	[2, 1.5, 1.25]	[100, 50, 25]
Action 9: LSS	D	[1, 1, 1]	[1.25, 1.125, 1.0625]	[1.5, 1.25, 1.125]	[2, 1.5, 1.25]	[100, 50, 25]
Action 10: LSR	E	[1, 1, 1, 1]	[1.25, 1.125, 1.0625, 1.03125]	[1.5, 1.25, 1.125, 1.0625]	[2, 1.5, 1.25, 1.125]	[100, 50, 25, 12.5]
Action 12: USR	F	[1, 1, 1, 1]	[1.25, 1.125, 1.0625, 1.03125]	[1.5, 1.25, 1.125, 1.0625]	[2, 1.5, 1.25, 1.125]	[100, 50, 25, 12.5]

Table A.2: Six parameters with five levels that each correspond to a specific multiplication of the base level scoring parameters. The parameters and levels are used to conduct the Taguchi experiments.

A.4 Objective values and SN ratio per Taguchi experiment

Experiment	**R1**	**R2**	**R3**	**R4**	**R5**	**R6**	**R7**	**R8**	**SN**
1	33.643	33.339	33.429	34.929	34.536	32.804	35.107	33.589	30.5026
	33.964	34.089	33.857	33.464	31.893	33.339	31.875	33.518	
	32.625	32.750	33.446	34.607	33.768	34.357	32.625	33.339	
2	33.982	32.679	33.643	34.554	33.125	34.571	32.893	33.429	30.4729
	30.821	33.804	33.196	31.554	32.768	32.786	33.964	34.464	
	33.321	34.018	33.750	33.946	33.554	33.982	33.768	33.679	
3	34.339	34.089	32.946	33.375	32.125	31.446	32.875	34.268	30.3800
	30.125	33.446	33.768	34.429	33.393	32.946	33.429	33.643	
	33.036	32.714	32.589	32.661	32.964	33.714	32.518	33.036	
4	33.804	33.768	32.982	33.036	34.250	34.161	34.628	35.143	30.4587
	33.500	31.857	33.518	32.911	33.161	33.982	33.839	32.661	
	33.304	32.732	33.196	33.375	32.518	32.339	32.821	33.196	
5	33.393	33.161	33.786	34.161	32.554	33.429	33.286	33.643	30.3853
	32.750	34.250	32.768	33.304	32.786	33.518	31.786	33.625	
	31.964	32.036	33.000	32.839	32.911	31.589	33.893	33.482	
6	33.643	32.000	34.804	33.929	33.893	33.196	32.679	32.821	30.4275
	33.143	34.464	31.214	31.143	33.321	33.339	33.964	32.143	
	33.071	34.143	34.304	33.839	32.929	33.911	33.500	32.786	
7	33.018	34.964	34.661	32.929	33.268	33.018	32.839	33.857	30.4353
	32.607	32.393	31.339	34.429	33.214	33.518	33.939	33.054	
	33.643	33.964	32.339	33.429	34.982	33.339	31.768	32.339	
8	33.518	32.696	34.946	31.196	33.786	33.268	34.411	34.321	30.4424
	32.804	33.911	32.357	32.893	34.018	33.268	34.054	33.893	
	32.000	32.250	32.071	33.839	32.679	33.607	34.821	32.939	
9	33.464	33.375	34.661	34.500	34.375	32.946	34.500	33.482	30.4543
	33.625	34.536	32.339	33.071	34.464	32.696	33.625	33.607	
	33.804	32.375	34.571	32.089	33.179	32.857	34.000	29.161	
10	33.714	33.071	33.089	30.946	33.768	32.946	33.018	33.214	30.3625
	33.393	32.964	33.054	32.000	32.661	33.982	33.939	33.232	
	33.125	31.429	32.643	33.000	33.821	33.071	32.750	33.021	
11	34.589	31.804	33.357	33.679	33.429	34.750	33.321	33.339	30.3624
	31.768	34.893	33.714	33.071	34.589	33.804	33.875	26.589	
	33.125	31.518	33.161	33.286	33.589	32.036	33.429	34.071	
12	33.696	35.054	33.946	33.536	32.500	32.286	35.196	33.643	30.3991
	33.411	32.946	33.321	33.375	33.429	33.089	31.089	32.089	
	32.518	34.179	31.500	33.161	33.232	33.911	32.286	32.250	
13	32.939	33.143	33.214	33.089	33.821	31.839	34.411	35.214	30.4035
	32.786	34.196	32.643	33.536	32.393	33.679	33.054	32.304	
	32.386	32.839	34.036	32.804	32.732	32.518	32.607	33.464	

Experiment	R1	R2	R3	R4	R5	R6	R7	R8	SN
14	34.339	33.232	34.000	32.518	33.589	34.250	34.518	34.464	30.4629
	35.304	31.911	32.625	33.518	32.536	32.321	33.500	33.143	
	31.375	34.214	33.732	33.804	32.304	34.054	32.804	33.375	
15	31.946	32.589	33.054	32.714	32.518	33.518	34.125	33.393	30.3589
	34.857	33.107	33.054	31.339	33.554	34.107	33.357	33.714	
	32.446	32.643	30.036	33.554	33.214	32.750	33.018	33.375	
16	34.071	33.607	34.250	33.661	34.268	34.821	34.768	32.786	30.4636
	33.714	33.643	33.411	32.107	33.071	34.339	34.089	33.054	
	33.250	32.214	33.768	32.500	33.214	32.536	34.196	30.304	
17	34.018	34.143	34.107	34.750	33.321	33.143	34.268	33.607	30.4833
	33.143	34.071	33.054	33.679	32.071	32.518	32.589	33.375	
	33.304	33.732	33.179	33.411	33.893	32.750	33.286	33.357	
18	34.750	34.143	34.125	34.589	32.536	32.982	33.607	34.125	30.4972
	32.518	33.696	35.125	33.643	33.446	33.696	32.500	33.607	
	33.446	32.929	33.196	33.875	32.464	33.250	33.036	32.911	
19	34.125	32.214	33.036	32.821	34.750	33.125	33.893	35.643	30.4412
	29.964	32.893	34.054	34.839	32.911	33.911	34.000	32.946	
	32.857	32.429	34.161	31.911	32.911	32.000	34.554	34.089	
20	33.893	31.893	33.750	33.482	33.518	32.804	32.964	33.857	30.3437
	32.350	32.911	32.304	30.286	31.304	33.500	32.768	33.732	
	33.250	33.232	33.018	33.071	33.357	32.661	32.911	33.554	
21	33.357	34.000	33.054	33.179	32.125	31.929	34.143	33.839	30.4022
	32.929	32.500	35.393	33.946	32.768	31.911	32.714	32.893	
	32.589	34.071	33.071	34.232	33.571	31.964	32.804	32.696	
22	31.661	34.500	33.964	33.143	33.661	34.125	34.750	33.232	30.3578
	30.500	33.929	34.107	34.196	32.786	33.571	33.000	32.054	
	33.107	33.464	31.018	30.821	33.750	32.571	31.732	32.732	
23	33.911	34.232	34.500	34.375	33.714	34.446	33.304	32.411	30.4325
	33.089	33.018	32.982	34.696	30.107	34.161	32.696	33.554	
	33.464	32.446	34.071	31.214	33.125	31.732	33.964	33.839	
24	33.714	32.250	33.375	33.429	33.696	31.982	33.482	33.339	30.4563
	33.946	33.750	33.929	33.911	34.571	33.321	33.339	34.179	
	32.089	33.625	32.732	33.696	32.607	32.750	33.607	33.018	
25	34.250	33.196	34.089	33.196	32.786	33.875	32.893	32.982	30.3963
	35.036	33.000	31.464	34.071	32.661	32.054	31.321	32.839	
	32.518	33.661	33.339	32.696	33.446	33.375	34.107	32.321	

Table A.3: Objective values of 8 replications (R) for each of the three QC placement scenarios. As specified by the Taguchi Method, 25 experiments were conducted. The last column denotes the SN ratio for each of the experiments.

Appendix B

Results

B.1 Figures of the simulated terminal

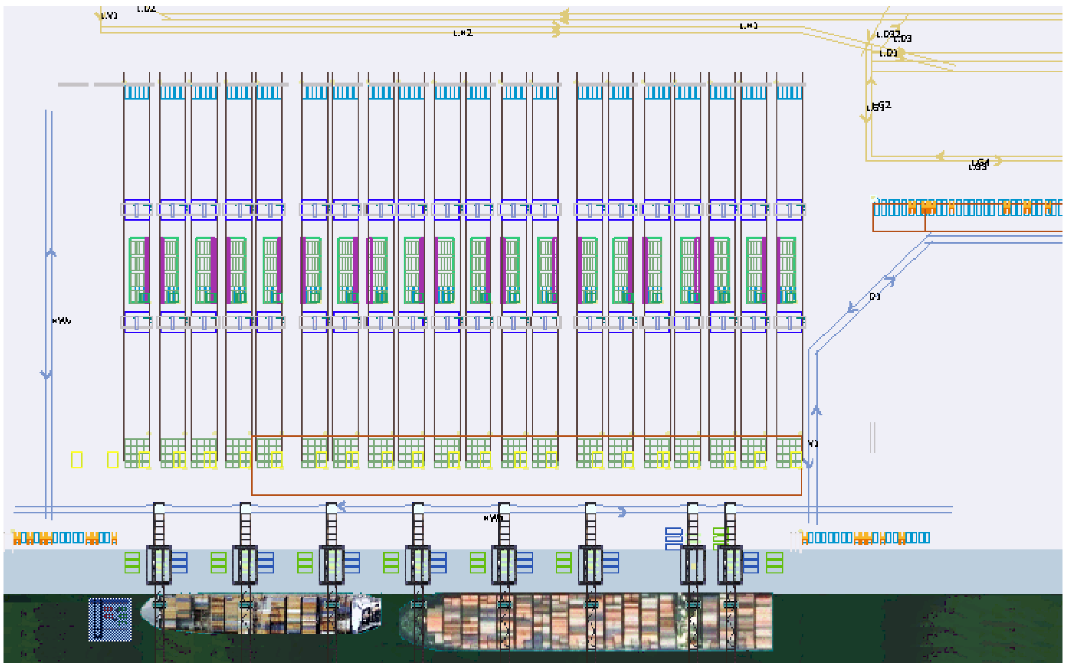


Figure B.1: QC placement $CL1$: the QCs are spread over the quay and only two QCs form a cluster (Screenshot TBA Simulation model TIMESQUARE).

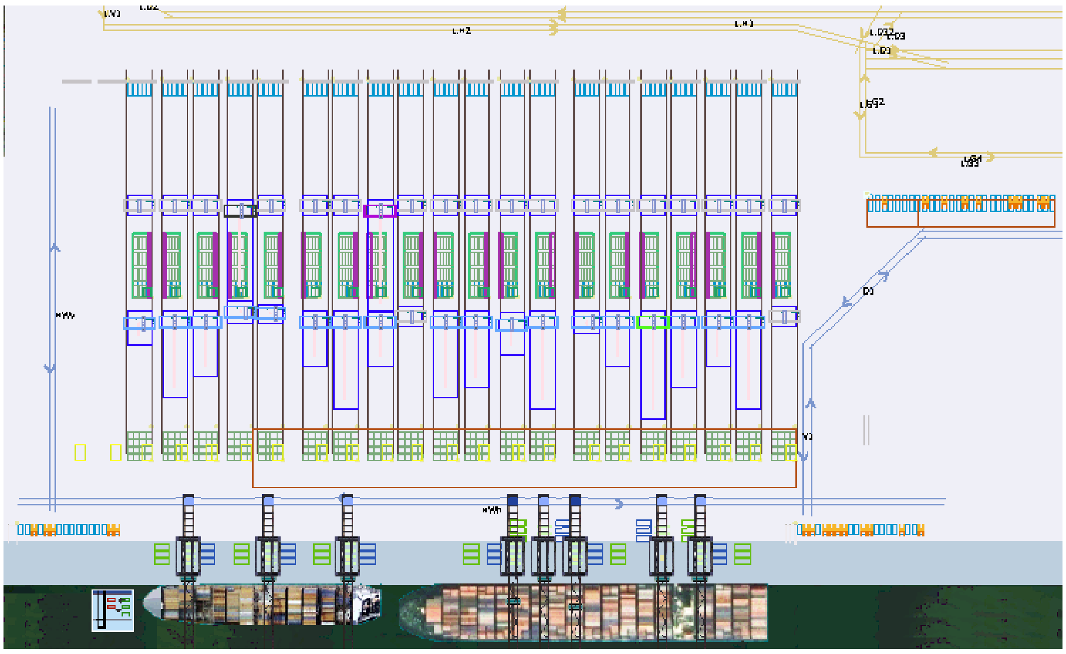


Figure B.2: QC placement $CL2$: on the right side of the quay the QCs are placed closer together (1 cluster of 3 QCs and one of 2 QCs), whereas the left side of the quay consists of three clusters with one QC each (Screenshot TBA Simulation model TIMESQUARE).

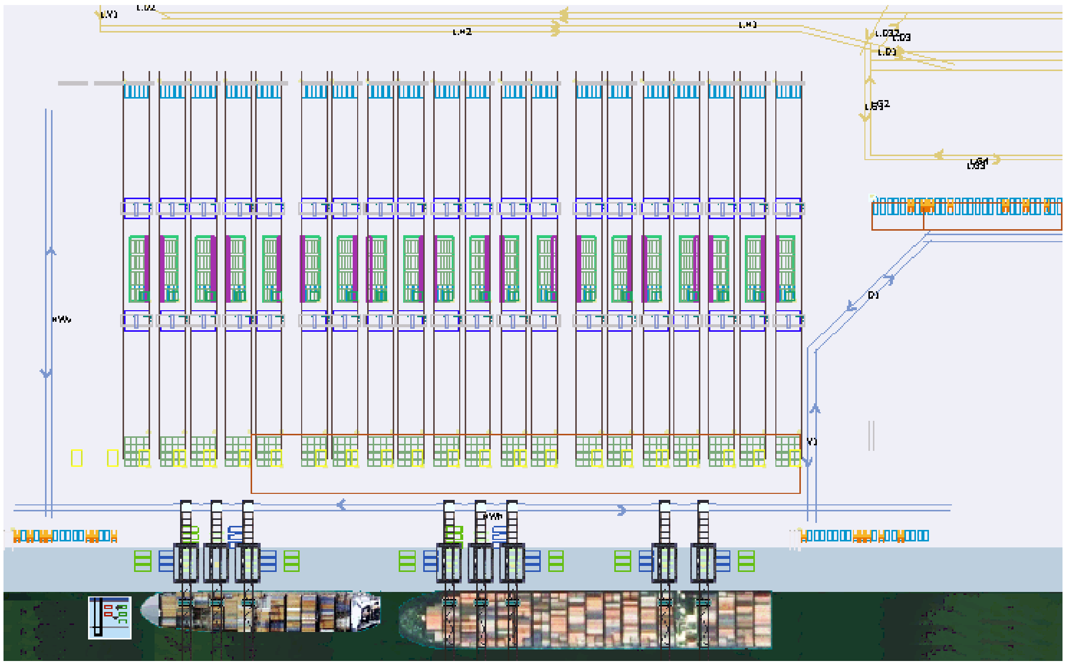


Figure B.3: QC placement $CL3$: each of the QCs is placed within two cranewidths of at least one other QC. There are three clusters: two with 3 QCs and one with 2 QCs (Screenshot TBA Simulation model TIMESQUARE).

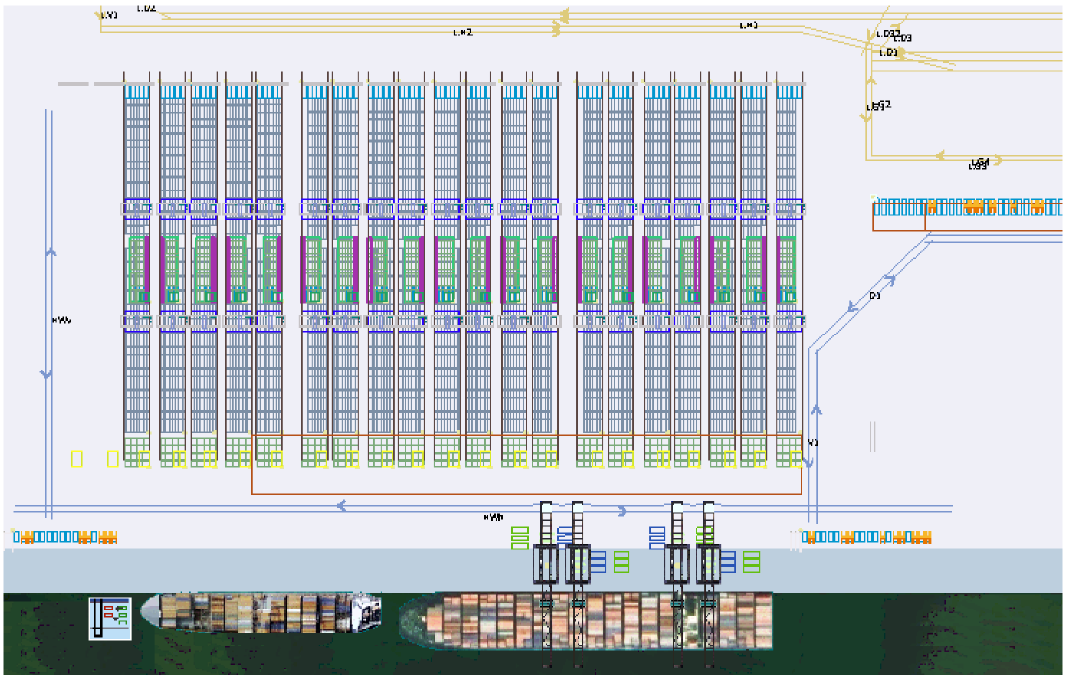


Figure B.4: The off-peak simulated terminal with 4 QCs and a highly decreased throughput (Screenshot TBA Simulation model TIMESQUARE).

B.2 State visits RLA

State	State visits	State	State visits	State	State visits	State	State visits
1111	250	2111	500	3111	1665	4111	435
1112	120	2112	985	3112	2248	4112	868
1113	200	2113	120	3113	100	4113	203
1121	125	2121	195	3121	1121	4121	666
1122	99	2122	510	3122	1750	4122	585
1123	101	2123	220	3123	202	4123	133
1131	2130	2131	5420	3131	6305	4131	2215
1132	3960	2132	13315	3132	12260	4132	2545
1133	120	2133	1415	3133	200	4133	225
1211	99	2211	230	3211	105	4211	205
1212	150	2212	120	3212	109	4212	33
1213	177	2213	143	3213	200	4213	245
1221	142	2221	144	3221	120	4221	1555
1222	131	2222	205	3222	144	4222	145
1223	52	2223	200	3223	143	4223	33
1231	1795	2231	2325	3231	4550	4231	730
1232	2140	2232	11445	3232	10950	4232	875
1233	101	2233	365	3233	220	4233	155
1311	122	2311	305	3311	554	4311	75
1312	133	2312	210	3312	460	4312	210
1313	120	2313	190	3313	103	4313	12
1321	99	2321	199	3321	270	4321	265
1322	89	2322	255	3322	315	4322	970
1323	141	2323	188	3323	180	4323	92
1331	1700	2331	7395	3331	6390	4331	750
1332	3935	2332	18495	3332	15930	4332	5905
1333	645	2333	535	3333	9	4333	85
1411	505	2411	495	3411	1960	4411	250
1412	510	2412	735	3412	870	4412	440
1413	150	2413	202	3413	300	4413	550
1421	1385	2421	655	3421	1675	4421	1080
1422	333	2422	935	3422	1300	4422	2285
1423	101	2423	110	3423	66	4423	45
1431	2080	2431	8710	3431	6975	4431	2540
1432	5005	2432	21255	3432	16755	4432	810
1433	205	2433	500	3433	33	4433	555

Table B.1: Number of visits to each state in the final run of RLA. The state abbreviation 1111 denotes the state with attributes $[A_1 = 1, A_2 = 1, A_3 = 1, A_4 = 1]$.

B.3 Predicted Q-values after RLA

State	Action 1	Action 2	Action 3	Action 4	Action 5	Action 6	Action 7	Action 8	Action 9	Action 10	Action 11	Action 12	Action 13	Action 14	Best Action
1111	175.41	333.51	232.85	244.88	266.41	235.01	194.04	201.40	162.08	235.38	164.58	340.05	235.38	145.25	12
1112	165.72	165.25	164.82	166.01	165.39	166.78	165.28	166.48	162.45	166.09	164.60	169.05	165.31	165.55	12
1113	149.66	149.90	162.76	148.85	149.12	149.75	149.54	147.53	147.42	147.06	149.10	149.88	147.23	145.07	3
1121	205.25	216.27	211.30	202.73	200.65	227.80	211.32	211.31	231.98	213.30	233.36	236.78	223.92	189.41	12
1122	161.67	164.37	161.53	160.69	163.82	165.38	156.44	164.72	173.68	165.03	163.91	176.14	164.89	155.12	12
1123	169.36	169.54	173.36	171.70	167.55	170.28	169.01	169.37	168.85	168.99	170.73	165.33	169.89	160.26	3
1131	303.24	293.28	229.62	231.64	288.76	290.80	228.59	296.29	282.07	260.92	252.23	353.73	312.55	201.61	12
1132	174.04	155.93	154.44	163.66	199.76	204.22	163.91	153.40	144.80	184.16	153.58	205.57	198.81	152.03	12
1133	158.65	148.91	162.77	151.08	156.83	149.40	148.58	155.39	158.86	155.30	158.08	154.72	159.33	149.53	3
1211	155.37	156.28	163.72	162.08	163.05	155.06	163.82	165.14	152.18	164.53	165.58	171.16	167.17	154.42	12
1212	164.73	164.32	164.66	165.87	164.29	166.35	164.65	165.79	163.32	165.34	164.43	165.97	164.83	165.01	8
1213	149.43	149.64	150.07	156.72	148.15	156.14	148.95	149.04	148.39	152.31	149.09	150.00	151.91	140.14	8
1221	153.16	152.85	159.11	153.89	161.29	157.70	158.08	167.95	156.11	162.29	152.32	169.83	162.64	152.52	12
1222	163.87	163.64	164.47	164.64	162.95	165.03	163.94	164.21	164.53	164.33	163.92	167.21	164.57	162.75	12
1223	159.42	159.72	164.31	150.13	157.69	160.23	159.15	159.49	159.23	158.92	149.12	150.10	154.85	150.26	3
1231	231.80	200.30	221.17	220.63	210.16	222.95	259.19	226.95	261.81	261.33	264.39	279.42	251.42	191.25	12
1232	193.76	193.70	204.50	194.12	176.53	194.33	199.81	193.51	215.12	193.83	199.90	185.04	184.75	174.92	9
1233	168.75	169.16	176.73	169.47	161.91	172.34	168.70	168.51	171.06	168.20	170.49	172.52	169.31	159.58	3
1311	164.88	174.26	166.53	171.79	153.20	156.21	144.16	155.45	153.41	144.88	163.62	177.69	163.97	144.27	12
1312	156.80	163.44	164.52	155.74	163.26	169.93	164.04	155.15	164.15	164.58	155.31	161.93	161.38	154.52	6
1313	162.31	170.62	175.02	171.32	168.68	161.25	169.88	160.69	169.85	166.72	163.30	160.63	163.49	161.09	3
1321	183.30	181.66	183.39	181.07	175.82	177.73	182.21	182.40	175.98	171.57	176.44	186.20	172.57	172.85	12
1322	173.47	173.30	167.51	163.99	169.58	175.03	173.75	164.18	166.99	163.92	169.18	165.59	164.46	164.59	8
1323	169.49	169.90	180.27	172.57	177.83	170.18	169.28	169.61	159.62	168.84	159.52	169.87	159.81	160.26	3
1331	192.63	207.16	212.51	231.20	211.04	199.08	241.01	200.62	222.14	252.87	201.78	283.63	231.74	171.99	12
1332	213.47	233.48	214.56	224.59	262.31	224.45	203.70	255.64	273.44	223.51	244.23	204.51	194.69	194.82	9
1333	188.81	189.34	209.68	192.90	187.05	189.30	181.83	185.63	192.44	198.13	188.88	199.29	189.27	179.58	3
1411	205.05	213.23	205.60	211.77	203.40	232.63	215.31	198.06	194.35	205.99	203.68	198.21	193.90	194.56	8
1412	213.19	202.90	236.52	225.85	202.62	195.72	213.69	204.85	214.86	194.00	201.46	203.15	199.18	194.27	3
1413	160.77	171.22	161.39	172.02	169.03	161.65	170.39	172.14	165.42	166.05	161.09	160.85	165.81	161.49	8
1421	234.18	231.34	234.84	231.86	242.57	247.52	223.94	236.17	223.72	215.45	212.91	216.39	222.90	203.61	8
1422	213.19	213.07	245.57	234.46	202.35	215.14	223.65	204.31	215.32	203.59	204.50	213.07	206.40	214.49	3
1423	172.83	179.37	175.50	176.20	178.15	177.44	179.67	179.98	172.16	173.04	172.20	179.95	170.03	170.54	8
1431	244.13	221.13	264.22	232.12	222.26	286.59	273.47	235.30	233.08	254.86	202.63	215.78	202.65	203.29	8
1432	263.14	223.18	264.61	225.04	202.05	244.55	253.59	223.73	281.81	264.19	214.49	263.98	214.59	194.65	9
1433	178.98	189.62	179.74	199.44	167.31	179.38	189.07	200.89	172.97	178.17	169.39	179.19	169.35	169.69	8
2111	202.05	189.99	200.76	191.14	209.61	213.24	200.48	203.22	197.65	192.26	181.11	225.15	202.19	175.79	12
2112	182.37	171.26	202.18	171.83	181.00	183.29	181.93	172.81	171.17	191.15	183.25	204.06	183.02	172.58	12
2113	174.85	165.37	174.66	167.83	171.38	168.76	167.26	171.38	164.60	166.99	164.99	169.63	170.80	162.21	3
2121	215.67	199.97	201.06	220.79	239.33	202.74	210.33	212.64	199.21	241.94	201.87	253.60	212.14	202.96	12

State	Action 1	Action 2	Action 3	Action 4	Action 5	Action 6	Action 7	Action 8	Action 9	Action 10	Action 11	Action 12	Action 13	Action 14	Best Action
2122	192.13	191.07	202.03	191.06	190.54	192.29	181.70	191.99	202.92	192.45	189.14	214.84	203.29	182.63	12
2123	174.81	165.15	175.80	165.09	173.82	169.44	165.08	174.14	171.48	172.93	168.54	175.88	173.44	165.40	12
2131	261.63	250.02	299.06	260.92	269.40	303.41	254.37	276.86	269.29	289.24	261.45	273.29	251.85	212.78	6
2132	202.35	183.33	192.35	221.04	200.65	192.15	202.00	191.78	192.79	212.62	191.32	223.02	203.63	182.92	12
2133	178.94	184.23	177.85	184.45	176.23	174.56	187.42	176.58	175.00	185.28	179.47	191.09	184.60	174.41	12
2211	181.44	183.87	189.54	182.01	180.92	183.69	181.43	179.97	182.05	180.72	184.72	186.87	183.23	179.85	3
2212	176.07	175.95	176.91	177.38	178.96	179.65	174.63	173.28	175.60	172.82	176.20	176.60	172.98	172.40	6
2213	179.69	176.10	182.35	177.77	179.39	180.04	178.79	176.98	179.75	177.34	176.49	180.54	174.28	175.24	3
2221	179.90	175.68	180.78	179.78	178.24	172.57	174.84	175.81	173.89	171.89	171.11	171.24	173.08	170.86	3
2222	151.91	167.94	159.78	166.70	140.61	144.81	141.45	161.38	162.06	152.36	151.00	181.41	179.02	142.39	12
2223	164.39	164.87	165.20	164.77	163.94	164.78	164.70	163.95	165.05	164.49	164.17	170.30	167.95	161.98	12
2231	150.23	166.48	221.95	199.27	198.44	201.13	199.13	200.52	244.84	261.65	255.02	301.38	278.19	121.03	12
2232	122.77	98.76	100.79	121.53	101.36	122.52	141.38	132.21	123.31	204.23	141.78	164.17	133.88	99.32	10
2233	134.06	104.51	124.94	144.73	113.71	104.69	115.61	133.79	155.02	134.54	113.70	125.99	144.53	97.02	9
2311	199.30	129.89	213.20	134.18	145.05	176.66	148.92	128.60	159.52	133.27	125.67	131.17	128.31	121.45	3
2312	141.80	130.74	121.86	131.54	120.89	122.52	131.50	152.09	141.95	173.61	156.34	169.12	142.85	112.56	10
2313	125.20	115.71	135.88	129.80	125.01	115.73	125.38	125.05	116.60	124.31	105.19	106.80	105.52	95.78	3
2321	108.75	111.66	109.95	106.86	108.26	116.29	113.28	107.58	109.55	110.71	100.00	120.59	99.17	100.04	12
2322	131.80	110.89	121.92	141.19	120.62	132.04	111.46	141.64	122.33	173.28	141.35	163.04	144.03	112.42	10
2323	124.51	135.15	135.24	129.13	124.22	126.84	124.87	124.14	125.17	134.59	124.50	125.17	124.92	125.08	3
2331	210.77	244.09	200.43	240.06	228.82	251.95	200.05	200.89	199.77	200.39	200.72	211.52	199.32	191.16	8
2332	192.57	181.63	202.76	191.97	181.14	212.81	202.19	212.52	213.14	222.99	191.99	213.76	183.74	173.25	10
2333	154.31	144.89	175.38	169.28	144.04	154.93	144.88	155.20	157.17	134.76	154.16	135.96	154.59	134.72	3
2411	221.03	250.19	278.00	220.49	219.50	239.26	210.97	266.47	201.54	242.50	241.45	267.74	255.56	199.34	3
2412	181.44	180.43	188.85	171.94	200.59	188.58	161.33	192.15	162.21	223.23	171.58	220.92	172.71	152.03	10
2413	175.33	175.96	177.91	176.31	175.22	174.76	174.59	175.24	176.06	175.30	175.66	175.65	175.55	173.85	3
2421	220.54	279.64	241.09	280.01	209.17	240.35	231.55	220.45	250.90	301.54	270.85	191.07	169.33	200.23	10
2422	271.70	170.77	303.16	271.96	290.57	192.54	271.53	182.16	226.68	212.32	251.76	192.69	203.05	172.43	3
2423	194.82	195.52	201.91	195.87	194.59	200.19	199.21	194.67	195.63	199.87	195.08	196.16	200.04	192.27	3
2431	262.03	126.11	201.66	155.65	200.04	293.87	201.92	202.47	141.40	133.76	122.26	202.49	201.15	102.08	8
2432	92.25	91.29	112.74	102.42	90.84	153.09	172.00	166.78	163.14	172.93	132.10	183.82	123.52	83.02	12
2433	194.73	185.36	236.37	196.03	204.42	205.42	191.25	193.88	195.60	213.08	234.84	185.12	184.90	188.10	3
3111	192.09	200.89	236.48	224.35	190.63	230.58	221.17	182.25	208.14	256.93	212.33	283.36	202.30	184.48	12
3112	143.04	103.79	151.16	123.72	192.15	155.84	162.48	173.21	144.74	205.84	152.88	175.00	123.64	113.42	10
3113	154.99	155.64	156.19	154.50	155.50	155.83	155.85	151.76	155.73	154.61	155.39	155.92	155.50	155.09	3
3121	203.15	202.28	181.07	183.67	200.61	202.57	191.85	182.98	178.43	194.24	153.46	223.95	183.73	163.80	12
3122	194.02	185.05	226.76	193.89	202.90	173.60	164.30	164.36	175.66	170.03	169.34	205.84	200.89	165.17	3
3123	175.32	175.66	176.74	175.87	174.94	176.51	176.35	174.50	176.72	176.48	175.55	176.44	175.57	172.52	3
3131	213.53	181.50	181.09	183.61	221.38	183.86	182.41	202.89	209.35	271.64	215.03	284.90	204.70	164.70	12
3132	193.32	214.65	194.89	252.71	141.98	202.80	153.48	144.02	194.69	192.91	183.81	275.99	205.76	134.88	12

State	Action 1	Action 2	Action 3	Action 4	Action 5	Action 6	Action 7	Action 8	Action 9	Action 10	Action 11	Action 12	Action 13	Action 14	Best Action
3133	135.29	135.27	136.60	136.05	134.88	136.70	136.27	135.57	136.39	136.71	135.12	136.57	135.70	133.43	10
3211	281.91	279.99	282.40	270.97	248.61	279.61	280.63	231.14	237.87	277.04	242.00	282.68	241.62	221.75	3
3212	194.28	176.89	196.87	177.45	185.34	174.02	193.52	185.22	197.10	194.56	174.22	199.80	181.43	164.92	12
3213	135.03	135.18	135.51	135.28	135.23	135.71	135.43	135.33	135.02	135.26	135.04	136.09	135.64	135.33	12
3221	192.28	200.21	190.43	202.91	179.38	180.90	211.20	212.05	199.80	204.44	182.57	222.93	182.59	182.64	12
3222	194.49	195.08	195.49	193.57	193.35	193.66	194.50	194.67	195.56	193.56	194.98	196.68	196.56	195.73	12
3223	104.96	114.99	116.70	115.55	114.96	115.97	112.56	114.69	111.21	115.72	114.82	116.07	115.48	107.19	3
3231	282.42	210.21	270.92	212.65	259.71	282.35	181.44	170.97	179.07	271.96	202.76	283.89	193.19	153.13	12
3232	135.23	134.04	144.49	152.30	131.92	202.98	123.43	143.58	154.77	192.83	173.70	225.84	165.37	104.62	12
3233	135.11	135.02	136.07	135.94	134.99	136.42	135.89	136.14	135.52	136.23	134.83	136.26	135.57	132.27	6
3311	210.53	207.74	221.31	200.09	207.46	228.60	216.38	282.52	269.44	226.17	211.06	230.89	219.96	179.07	6
3312	193.45	173.90	191.97	172.25	189.80	166.37	192.22	164.21	179.17	204.93	193.90	194.80	186.69	153.66	10
3313	135.09	134.42	134.94	135.11	135.38	135.11	135.20	135.39	133.85	134.49	135.24	135.75	135.66	134.35	12
3321	182.96	179.62	180.27	193.63	208.60	179.88	181.11	209.82	211.63	195.61	182.22	222.77	181.92	161.97	12
3322	198.73	180.14	194.46	188.29	182.69	193.24	203.57	183.45	185.75	199.36	174.15	215.64	195.40	164.75	12
3323	134.86	135.01	135.61	135.58	135.14	135.90	135.48	135.70	134.99	135.15	134.80	135.75	135.22	135.12	6
3331	182.29	219.92	179.91	222.62	179.21	181.99	192.51	200.43	179.68	273.88	211.51	282.66	202.62	142.61	12
3332	193.32	163.79	215.34	192.27	176.04	183.59	165.60	143.57	194.89	153.08	193.81	164.83	186.23	134.62	3
3333	145.20	145.24	145.17	144.15	145.38	146.52	139.01	141.33	145.51	142.39	144.02	145.13	144.53	139.40	6
3411	281.91	237.24	260.55	281.95	254.66	220.54	272.57	230.67	211.16	276.98	252.46	282.61	240.60	178.09	12
3412	182.35	142.39	187.05	141.06	133.41	189.77	160.53	163.42	142.05	183.18	143.94	194.52	134.15	141.99	12
3413	165.12	155.84	166.37	142.94	145.12	151.51	143.56	154.30	137.37	139.06	145.53	134.61	146.30	134.96	3
3421	211.61	228.78	270.15	192.44	218.08	223.56	211.09	242.09	250.09	266.42	191.96	282.53	199.34	151.38	12
3422	222.96	193.06	223.44	191.14	211.96	203.05	222.64	192.47	213.79	277.22	193.30	285.40	199.18	143.71	12
3423	144.91	145.14	155.70	156.79	155.50	159.05	145.60	155.91	153.03	149.92	144.95	147.65	140.17	135.19	6
3431	182.31	229.20	180.10	233.02	279.11	202.43	281.85	244.42	200.11	274.92	212.52	284.65	222.29	142.30	12
3432	193.30	163.12	184.04	152.34	166.78	194.43	153.75	193.76	204.79	192.82	174.00	225.73	195.06	154.60	12
3433	145.94	136.08	146.92	136.99	140.40	147.23	136.78	142.19	144.22	137.09	139.94	146.66	141.18	136.21	6
4111	263.62	235.48	226.92	287.45	304.43	302.16	288.52	306.86	297.81	333.52	301.86	355.32	332.61	224.26	12
4112	200.69	242.95	216.92	222.45	215.36	191.45	216.67	201.27	250.59	219.41	220.35	303.54	279.57	201.04	12
4113	152.42	147.72	151.61	162.13	151.63	153.25	160.16	149.59	143.84	163.96	144.55	164.79	150.67	148.91	12
4121	223.89	264.58	301.51	267.29	241.07	301.00	256.11	196.96	204.96	290.21	277.07	358.19	344.38	162.83	12
4122	200.24	222.31	243.19	221.20	199.94	219.10	222.63	201.28	214.99	206.44	200.93	222.42	220.45	191.56	3
4123	152.42	169.77	184.30	163.16	171.38	153.13	163.40	151.52	165.22	153.66	158.77	152.91	157.46	150.40	3
4131	222.80	211.55	300.99	265.33	159.47	258.93	215.97	285.49	234.24	225.56	203.71	284.34	263.39	160.02	3
4132	219.78	241.70	253.38	220.09	230.77	217.30	200.04	191.13	215.24	213.26	221.38	201.96	181.61	192.28	3
4133	174.64	152.31	204.38	198.65	173.61	164.20	163.33	184.18	162.31	173.98	180.80	210.97	184.42	163.40	12
4211	185.27	196.40	195.06	205.75	200.50	211.43	204.81	193.16	216.52	203.30	211.97	212.40	201.82	182.40	6
4212	209.07	210.89	208.27	199.34	204.75	217.37	202.45	222.58	198.37	215.49	200.37	220.40	201.30	188.54	8
4213	141.57	150.04	146.37	157.09	170.38	146.36	141.66	160.55	155.89	142.61	153.59	190.80	180.31	139.33	12

State	Action 1	Action 2	Action 3	Action 4	Action 5	Action 6	Action 7	Action 8	Action 9	Action 10	Action 11	Action 12	Action 13	Action 14	Best Action
4221	265.88	215.60	223.84	208.90	161.01	182.19	258.04	198.82	315.10	342.25	306.04	355.67	345.72	160.57	12
4222	221.29	213.12	223.91	220.56	212.55	219.26	220.03	213.81	212.48	215.23	222.36	203.29	212.41	203.49	3
4223	183.79	184.02	192.77	192.60	184.19	191.85	184.82	194.70	188.52	179.09	180.98	183.95	189.48	179.29	6
4231	363.22	362.05	366.32	365.85	359.36	358.95	356.24	365.18	334.62	337.64	363.95	364.59	363.50	360.13	3
4232	220.25	182.03	223.45	189.48	201.36	197.71	219.85	222.09	203.48	192.62	221.94	220.40	202.37	193.22	3
4233	204.04	204.26	203.34	203.09	204.35	202.34	202.02	205.14	200.81	199.68	191.46	204.18	194.74	194.61	8
4311	363.51	323.19	317.46	363.78	296.76	369.93	335.50	370.62	306.04	342.65	339.39	366.00	328.49	256.23	6
4312	247.10	218.68	196.70	217.82	188.76	216.64	206.11	280.74	188.42	214.43	218.99	238.57	208.85	205.23	6
4313	182.67	182.03	185.70	186.64	188.63	185.04	190.53	182.66	181.13	188.84	179.28	193.81	182.92	181.37	12
4321	265.31	334.83	263.13	316.66	358.91	330.86	357.31	327.36	335.62	343.98	305.52	367.01	324.84	260.52	12
4322	251.69	223.37	263.60	231.09	242.03	228.92	250.45	243.65	253.32	217.03	223.01	233.58	222.49	223.98	3
4323	184.76	175.03	183.97	182.82	175.42	182.20	179.60	185.52	184.92	179.51	179.42	184.54	181.93	175.02	6
4331	363.25	322.62	320.60	165.85	358.74	309.23	355.91	284.69	334.76	339.58	323.68	367.59	353.32	260.80	12
4332	218.56	220.30	225.70	217.31	209.80	196.51	217.99	220.03	211.87	191.76	200.14	190.71	192.33	191.35	3
4333	163.85	154.13	163.26	162.79	164.31	162.09	164.20	166.86	156.71	159.37	160.58	153.81	154.39	154.50	8
4411	261.56	220.76	298.95	331.20	272.32	370.08	325.89	268.76	295.21	342.21	356.21	293.85	305.00	149.74	6
4412	217.52	218.96	186.53	218.58	175.15	228.78	201.48	251.26	179.96	215.13	199.77	199.12	209.81	164.27	6
4413	153.22	153.53	145.00	156.29	156.03	154.63	140.49	153.51	156.01	146.81	148.55	163.76	143.62	152.00	12
4421	264.63	284.24	342.16	305.17	256.66	310.00	256.46	266.08	206.05	345.63	304.79	366.39	253.87	162.38	12
4422	220.93	202.53	233.55	210.44	199.22	197.50	219.68	222.16	193.65	198.37	182.36	222.56	201.30	172.89	3
4423	156.79	156.97	158.10	154.00	157.80	153.45	135.93	156.91	154.02	151.11	149.03	156.13	156.38	152.48	3
4431	263.42	313.28	260.18	296.15	357.40	229.81	185.64	354.47	325.17	221.85	203.65	366.89	323.31	162.62	12
4432	226.00	207.75	288.62	214.32	257.03	233.18	221.26	216.85	210.03	240.73	217.42	200.99	207.31	208.31	3
4433	193.74	184.07	183.19	182.58	184.39	191.87	193.31	194.74	184.85	189.29	180.22	182.47	174.08	174.40	6

Table B.2: Q-values, as predicted by RLA for each of the 144 states and 14 actions. The last column denotes the action corresponding to the highest Q-value: the 'best action.'

B.4 Simulation results RLA

B.4.1 Net QC productivity per hour

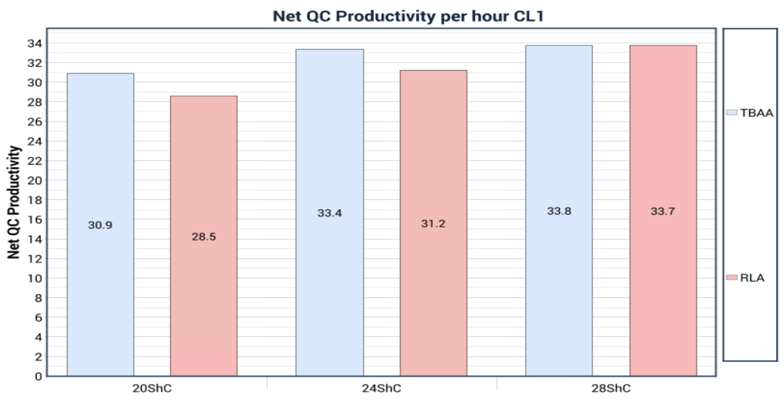


Figure B.5: Net QC productivity in number of containers per hour for QC placement $CL1$ with 20, 24 and 28 ShCs. Comparing simulation results of RLA (pink) and TBAA (blue).

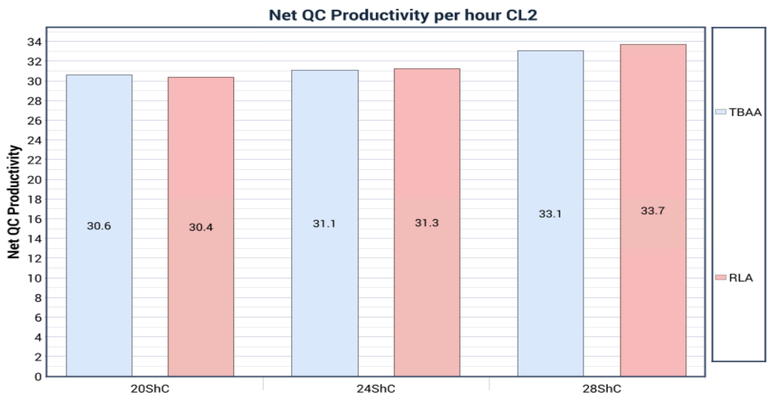


Figure B.6: Net QC productivity in number of containers per hour for QC placement $CL2$ with 20, 24 and 28 ShCs. Comparing simulation results of RLA (pink) and TBAA (blue).

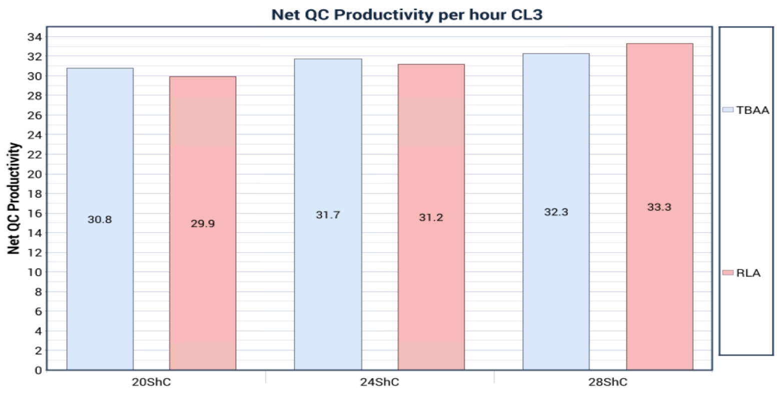


Figure B.7: Net QC productivity in number of containers per hour for QC placement $CL3$ with 20, 24 and 28 ShCs. Comparing simulation results of RLA (pink) and TBAA (blue).

B.4.2 Net QC productivity per function

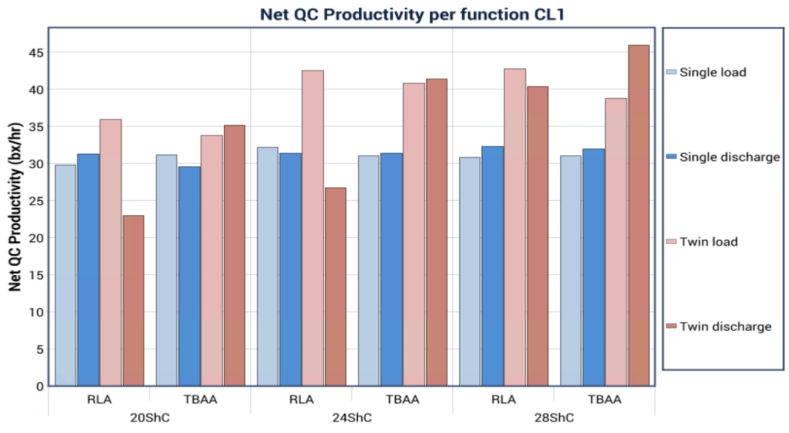


Figure B.8: Net QC productivity per function in number of containers per hour for QC placement $CL1$ with 20, 24 and 28 ShCs. Comparing simulation results of RLA and TBAA.

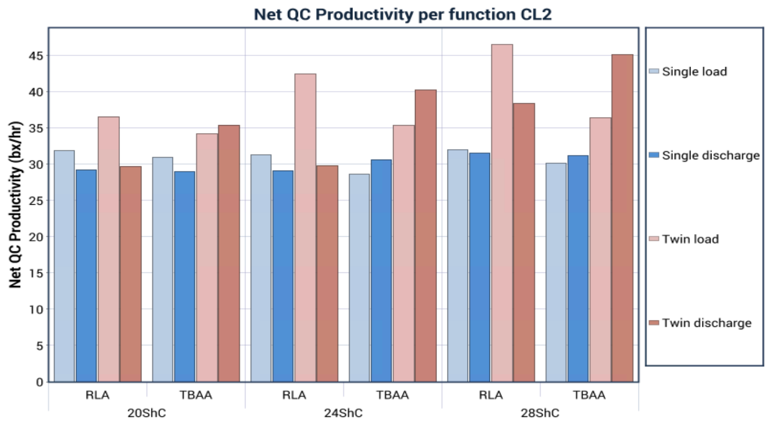


Figure B.9: Net QC productivity per function in number of containers per hour for QC placement $CL2$ with 20, 24 and 28 ShCs. Comparing simulation results of RLA and TBAA.

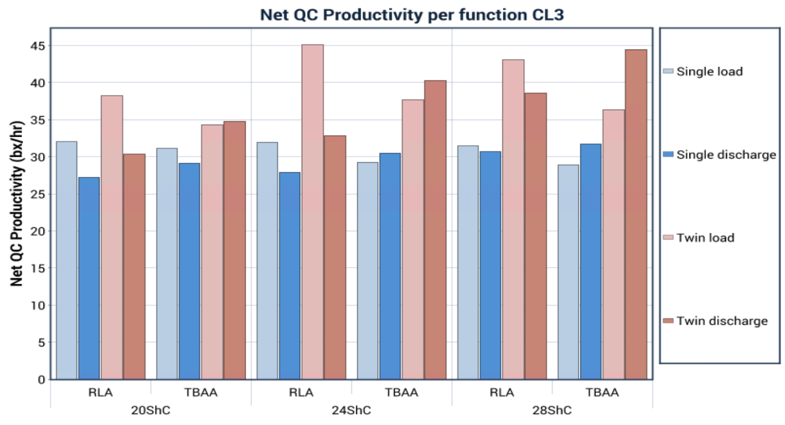


Figure B.10: Net QC productivity per function in number of containers per hour for QC placement $CL3$ with 20, 24 and 28 ShCs. Comparing simulation results of RLA and TBAA.

B.5 Simulation Results DSA

B.5.1 Net QC productivity per hour

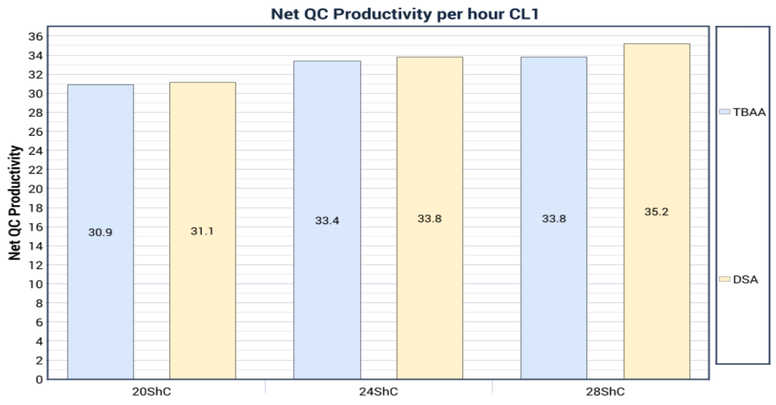


Figure B.11: Net QC productivity in number of containers per hour for QC placement $CL1$ with 20, 24 and 28 ShCs. Comparing simulation results of DSA (yellow) and TBAA (blue).

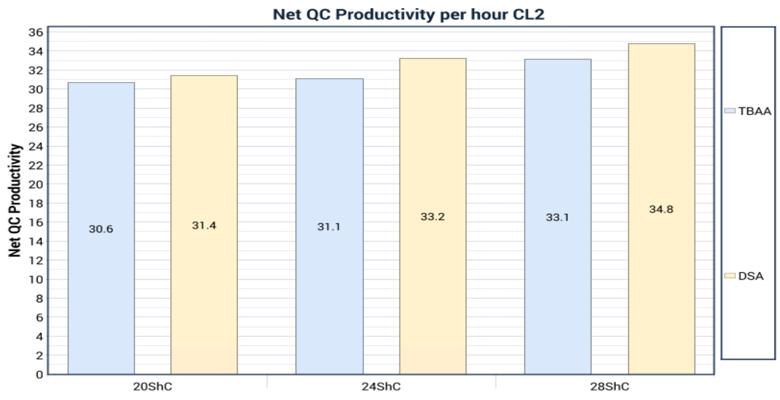


Figure B.12: Net QC productivity in number of containers per hour for QC placement $CL2$ with 20, 24 and 28 ShCs. Comparing simulation results of DSA (yellow) and TBAA (blue).

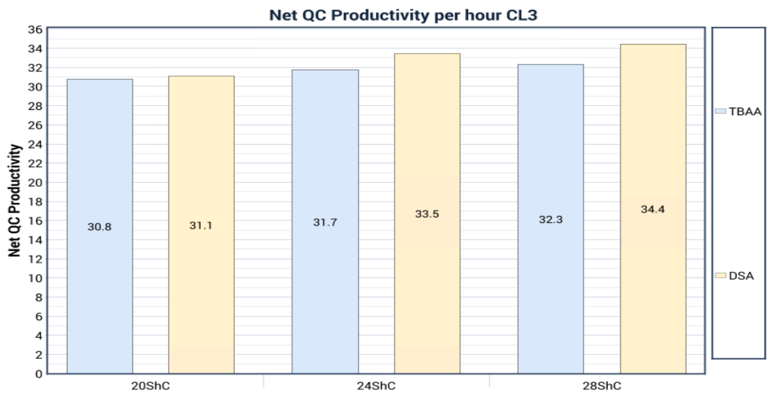


Figure B.13: Net QC productivity in number of containers per hour for QC placement $CL3$ with 20, 24 and 28 ShCs. Comparing simulation results of DSA (yellow) and TBAA (blue).

B.5.2 Net QC productivity per function

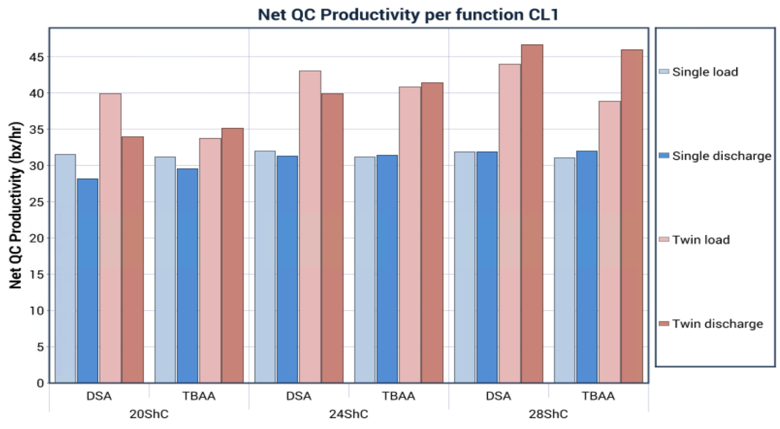


Figure B.14: Net QC productivity per function in number of containers per hour for QC placement $CL1$ with 20, 24 and 28 ShCs. Comparing simulation results of DSA and TBAA.

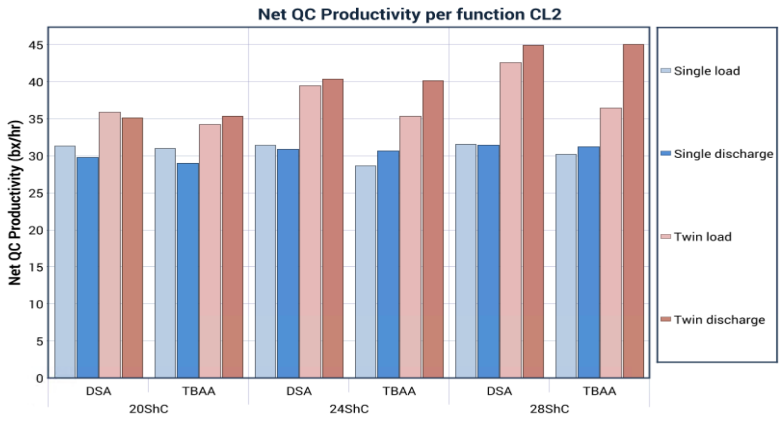


Figure B.15: Net QC productivity per function in number of containers per hour for QC placement $CL2$ with 20, 24 and 28 ShCs. Comparing simulation results of DSA and TBAA.

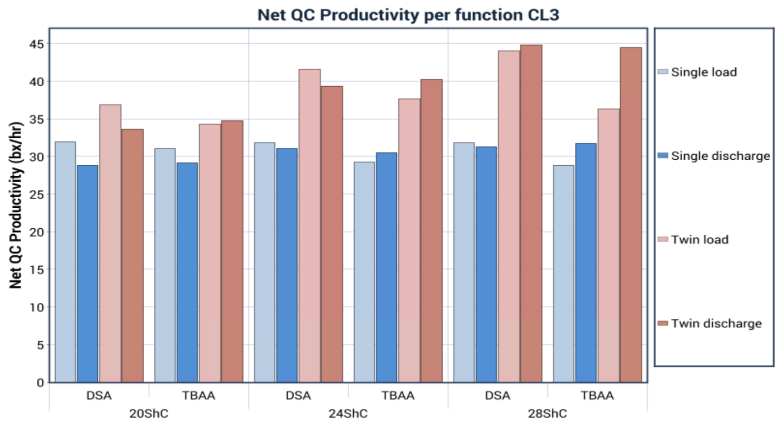


Figure B.16: Net QC productivity per function in number of containers per hour for QC placement $CL3$ with 20, 24 and 28 ShCs. Comparing simulation results of DSA and TBAA.

B.5.3 ShC status

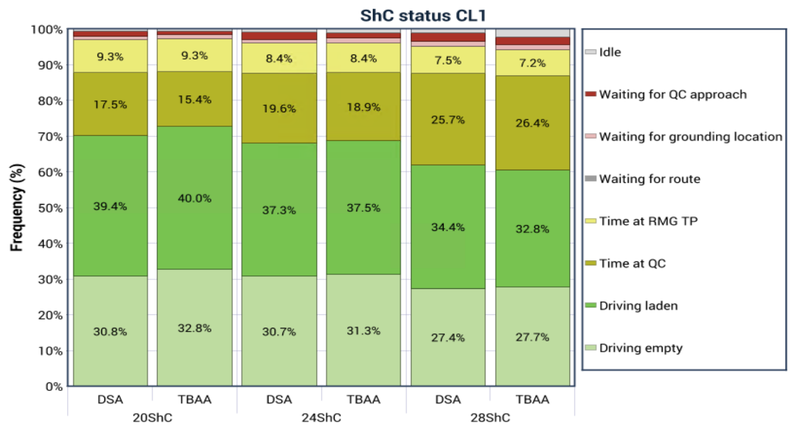


Figure B.17: ShC status as a percentage of time for QC placement $CL1$ with 20, 24 and 28 ShCs. Comparing simulation results of DSA and TBAA.

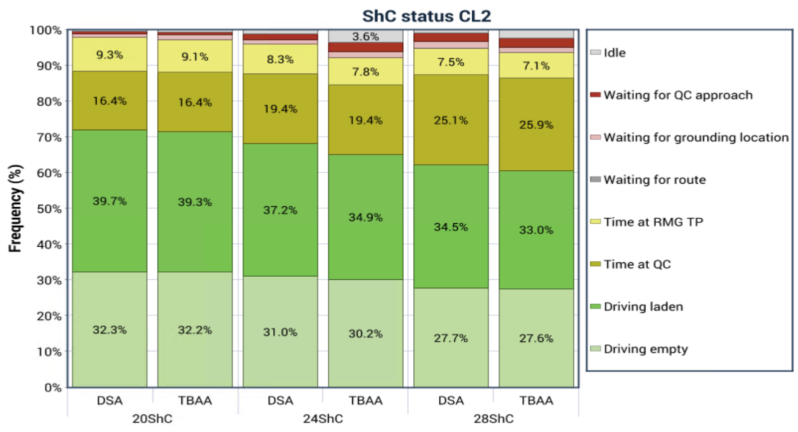


Figure B.18: ShC status as a percentage of time for QC placement $CL2$ with 20, 24 and 28 ShCs. Comparing simulation results of DSA and TBAA.

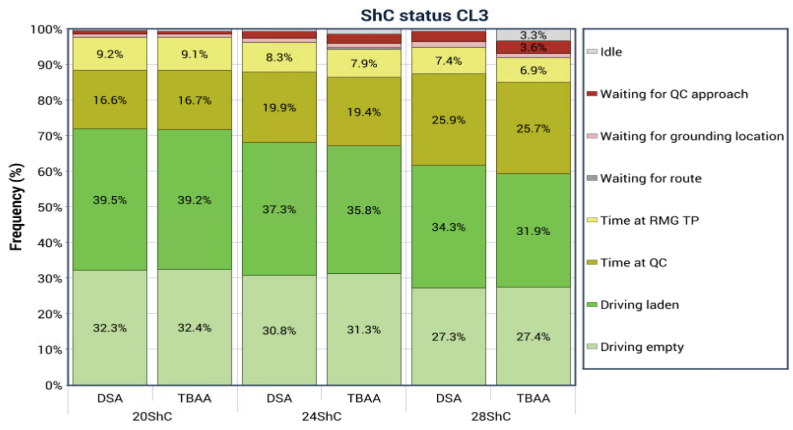


Figure B.19: ShC status as a percentage of time for QC placement $CL3$ with 20, 24 and 28 ShCs. Comparing simulation results of DSA and TBAA.

B.5.4 QC status per container

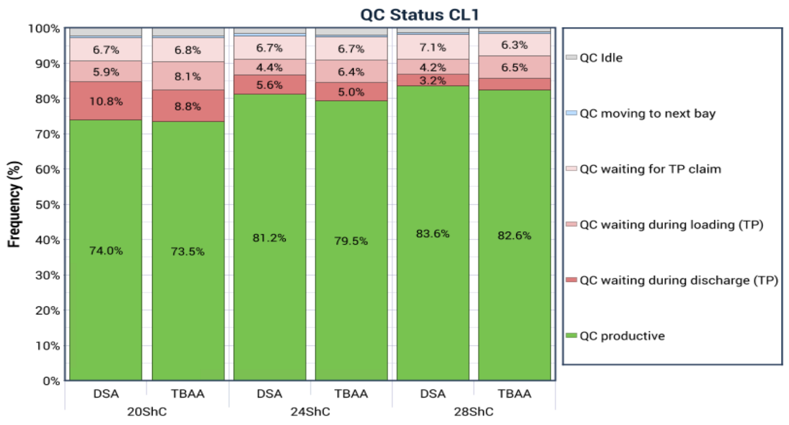


Figure B.20: QC status as a percentage of time for QC placement $CL1$ with 20, 24 and 28 ShCs. Comparing simulation results of DSA and TBAA.

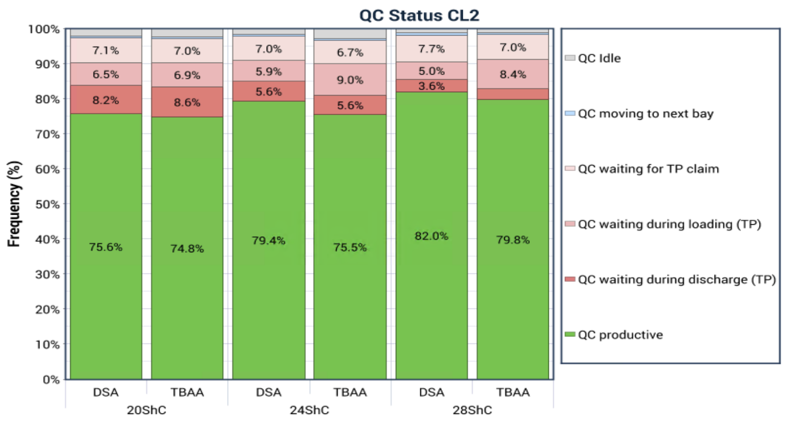


Figure B.21: QC status as a percentage of time for QC placement $CL2$ with 20, 24 and 28 ShCs. Comparing simulation results of DSA and TBAA.

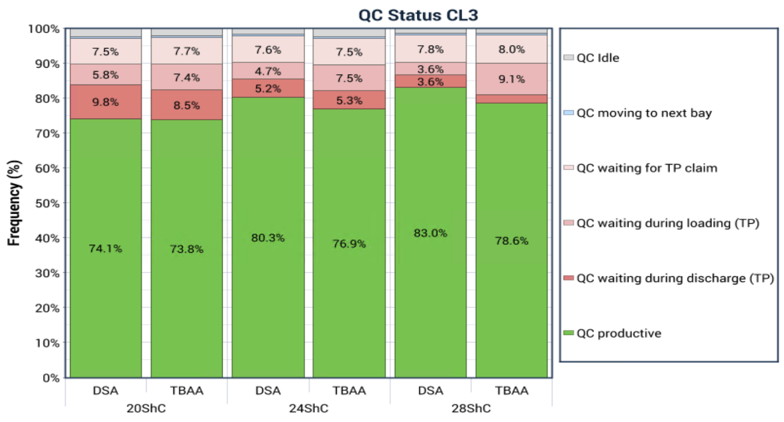


Figure B.22: QC status as a percentage of time for QC placement $CL3$ with 20, 24 and 28 ShCs. Comparing simulation results of DSA and TBAA.

B.6 Off-peak simulation

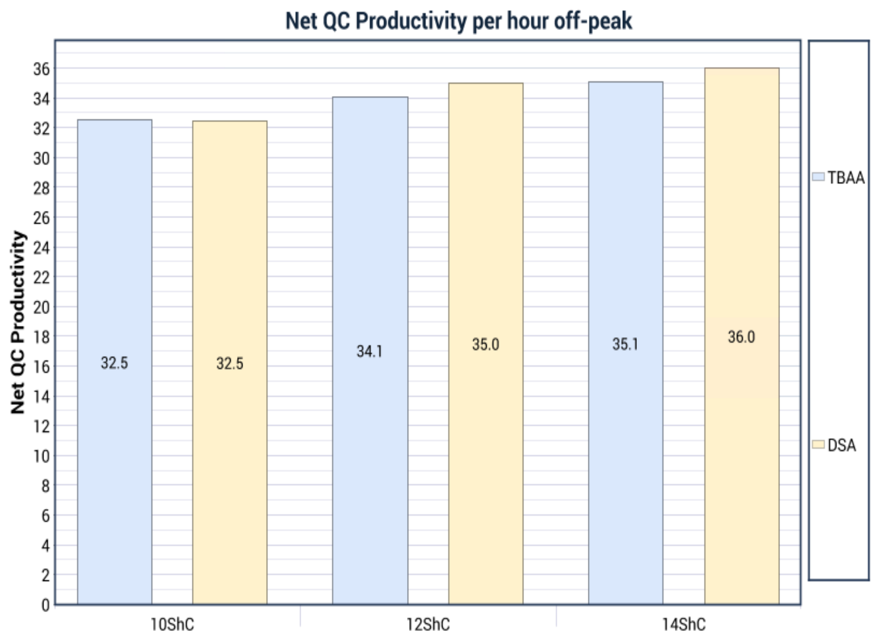


Figure B.23: Net QC productivity in number of containers per hour for off-peak simulation experiments with 10, 12 and 14 ShCs. Comparing simulation results of DSA (yellow) and TBAA (blue).

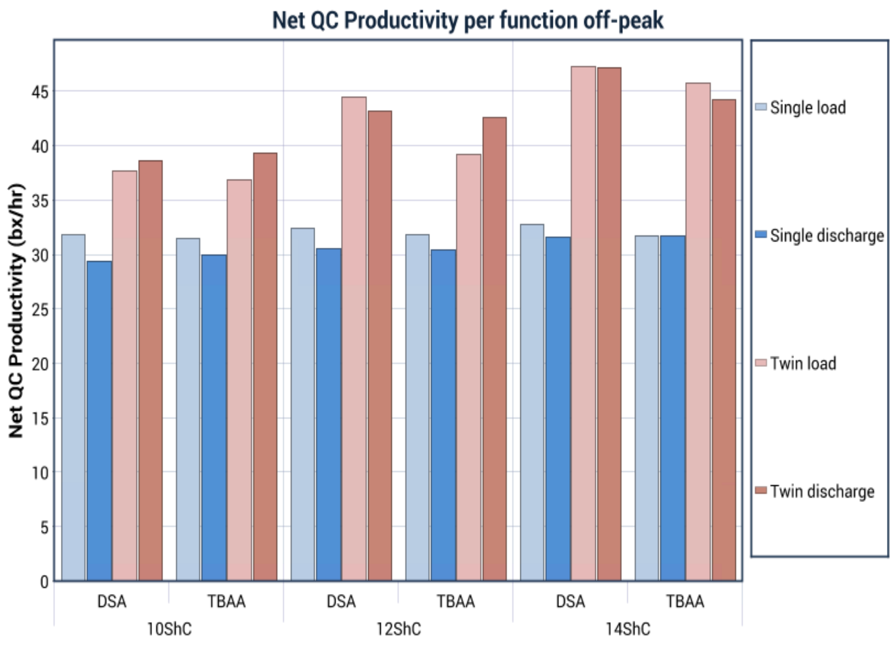


Figure B.24: Net QC productivity per function in number of containers per hour for off-peak simulation experiments with 10, 12 and 14 ShCs. Comparing simulation results of DSA and TBAA.

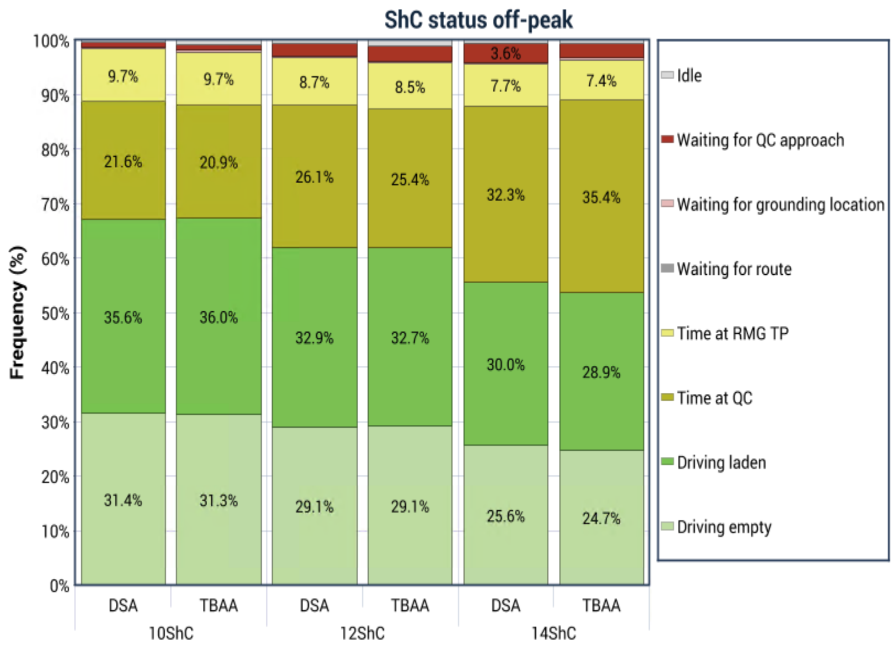


Figure B.25: ShC status as a percentage of time for off-peak simulation experiments with 10, 12 and 14 ShCs. Comparing simulation results of DSA and TBAA.

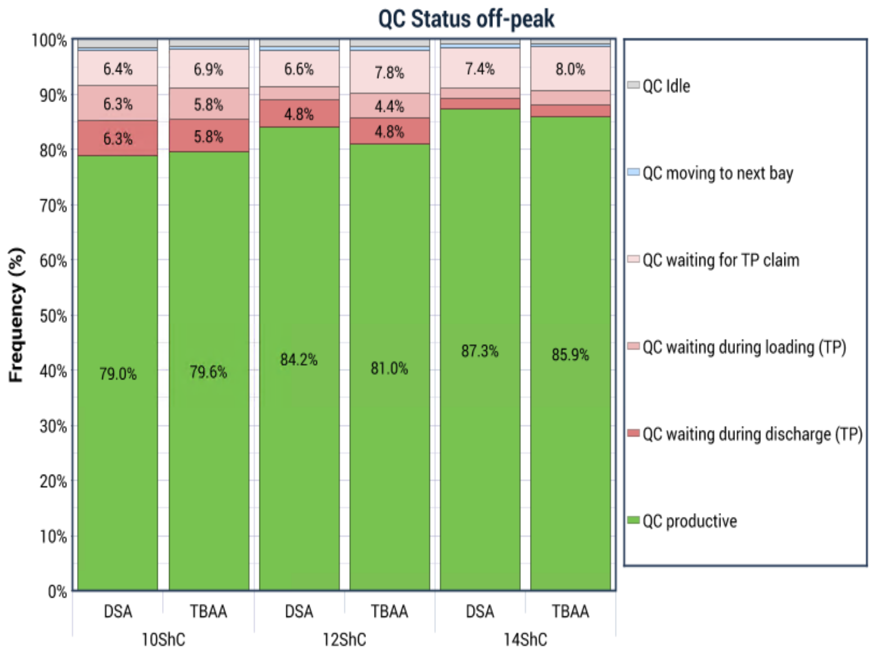


Figure B.26: QC status as a percentage of time for off-peak simulation experiments with 10, 12 and 14 ShCs. Comparing simulation results of DSA and TBAA.