ANDRÉ FONTANEZ BRAVO

A DOUBLE DEEP Q-LEARNING APPROACH TO THE CONTAINER RELOCATION PROBLEM

São Paulo
2025

ANDRÉ FONTANEZ BRAVO

A DOUBLE DEEP Q-LEARNING APPROACH TO THE CONTAINER RELOCATION PROBLEM

Graduation Thesis presented to Escola Politécnica da Universidade de São Paulo to obtain a Degree in Production Engineering.

São Paulo
2025

ANDRÉ FONTANEZ BRAVO

A DOUBLE DEEP Q-LEARNING APPROACH TO THE CONTAINER RELOCATION PROBLEM

Graduation Thesis presented to Escola Politécnica da Universidade de São Paulo to obtain a Degree in Production Engineering.

Supervisor:
Leonardo Junqueira

São Paulo
2025

“Questions you cannot answer are usually far better for you than answers you cannot question.”

-- Yuval Noah Harari

ABSTRACT

Efficient management of container stacks in maritime terminals is critical to global supply chains, as relocating “blocking” containers incurs substantial time and cost penalties. This work addresses the static, distinct, deterministic, restricted Container Relocation Problem (CRP), in which containers with predetermined retrieval priorities must be relocated only when they directly block the current target. Following a novel approach, the CRP is modelled as a Markov Decision Process, with bay configurations as states, relocation moves as actions, and a unit penalty per relocation. To learn effective relocation policies without domain-specific heuristics, a Double Deep Q-Learning (DDQN) agent is implemented that leverages separate online and target networks to evaluate the layout of containers and decide the best action to take. Given the sensitivity of Deep Reinforcement Learning to hyperparameters, Bayesian Optimization (BO) is implemented to automatically tune crucial hyperparameters in a sample-efficient manner. Computational experiments on standard benchmark instances demonstrate that the approach obtains near-state-of-the-art results, with low computational times, indicating the methods used are quite promising in the context of CRP. This study is, to the best of the author’s knowledge, the first to apply DDQN with Bayesian Optimization to the CRP, opening a promising avenue for machine-learning-driven yard management.

Keywords – Container Relocation Problem, Blocks Relocation problem, Double Deep Q-Learning, Bayesian Optimization, Container Reshuffling.

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ACO Ant Colony Optimization.

B&B Branch and Bound.

BAP Berth Allocation Problem.

BO Bayesian Optimization.

BRP Blocks Relocation Problem.

CNN Convolutional Neural Network.

CRP Container Relocation Problem.

CSPP Container Stowage Planning Problem.

DDQN Double Deep Q-Learning.

DQN Deep Q-Learning.

GP Gaussian Process.

LIFO Last In, First Out.

MDP Markov Decision Process.

MHE Material Handling Equipment.

MLP Multilayer Perceptron.

PMP Pre-Marshalling Problem.

QCAP Quay Crane Assignment Problem.

QCSP Quay Crane Scheduling Problem.

ReLU Rectified Linear Unit.

RL Reinforcement Learning.

SGD Stochastic Gradient Descent.

CONTENTS

1 INTRODUCTION

This chapter presents the context and motivation for this work, the specific objectives pursued, and the overall structure of the thesis.

1.1 Context and Motivation

Maritime transport carries over 80% of global trade, making ports and container terminals pivotal nodes in international supply chains (WORLD BANK, 2023). Containers arrive on ships, trains and trucks and are stored at container yards until they can be further transported towards their final destination.

At the yards, the containers are stacked on top of each other to maximize space utilization, as shown in Figure 1.1. However, stacking also restricts access, as only the top container in each stack can be retrieved directly. Whenever a target container is blocked beneath others, those blocking containers must be moved elsewhere, in what is known as a relocation. Furthermore, these relocations can themselves block additional containers, requiring further relocations in the near future.

Such unproductive moves significantly degrade yard productivity (YANG; KIM, 2006) and contribute to port inefficiency, shipment delays, and increased logistics costs (WORLD BANK, 2024). The combinatorial challenge of sequencing container moves to minimize total relocations is known as the Container Relocation Problem (CRP) or Blocks Relocation Problem (BRP) (CASERTA; SCHWARZE; VOSS, 2012). Despite several studies, using exact, heuristic, machine learning, and hybrid approaches, state-of-the-art methods still handle only small to medium-sized instances efficiently (LERSTEAU; SHEN, 2022). For this reason, this thesis proposes a novel approach to the problem, leveraging modern Reinforcement Learning (RL) techniques and aiming to open a new avenue that may lead to better terminal efficiency.

Figure 1.1: Container terminal of Deepwater Container Terminal Gdańsk SA

Source: Moszyk, Deja, and Dobrzynski (2021).

1.2 Objectives

The main objective of this thesis is to investigate whether a Double Deep Q-Learning (DDQN) agent (VAN HASSELT; GUEZ; SILVER, 2016), with no prior domain-specific knowledge, is capable of learning an efficient relocation policy to solve the static, distinct, deterministic, restricted CRP and minimize the number of relocations in a reasonable timeframe. In this version of the problem, no new containers arrive during the operation (static), the retrieval priority of each container is unique (distinct) and fully known from the beginning (deterministic), and, at any given moment, only containers blocking the container of highest priority may be relocated (restricted).

In greater detail, the objectives are to:

- Formally model the CRP as a Markov Decision Process (MDP), in which states represent bay configurations, actions correspond to relocating or retrieving a container, and rewards penalize each relocation.
- Implement a Double Deep Q-Learning (DDQN) (VAN HASSELT; GUEZ; SILVER, 2016) agent that uses deep neural networks to evaluate states and decide the next action.
- Employ Bayesian Optimization (BROCHU; CORA; DE FREITAS, 2010; FRAZIER, 2018) to tune the DDQN hyperparameters automatically and efficiently.

- Compare the DDQN agent's performance against state-of-the-art exact and heuristic methods on benchmark instances, assessing both solution quality (number of relocations) and computational effort.

1.3 Thesis Structure

This thesis is organized in a total of six chapters, beginning with this introduction. Chapter 2 explains the theoretical background, including the context of the problem in container terminals and the theory behind the methods used in this work, as well as key papers that have contributed to the field of CRP. Chapters 3 and 4 detail the modeling and practical implementation of the theoretical concepts, respectively. Then, Chapter 5 analyzes the results obtained with this work's method. Lastly, Chapter 6 concludes this thesis by summarizing the work done and the results obtained, and suggesting paths for further research.

2 THEORETICAL BACKGROUND

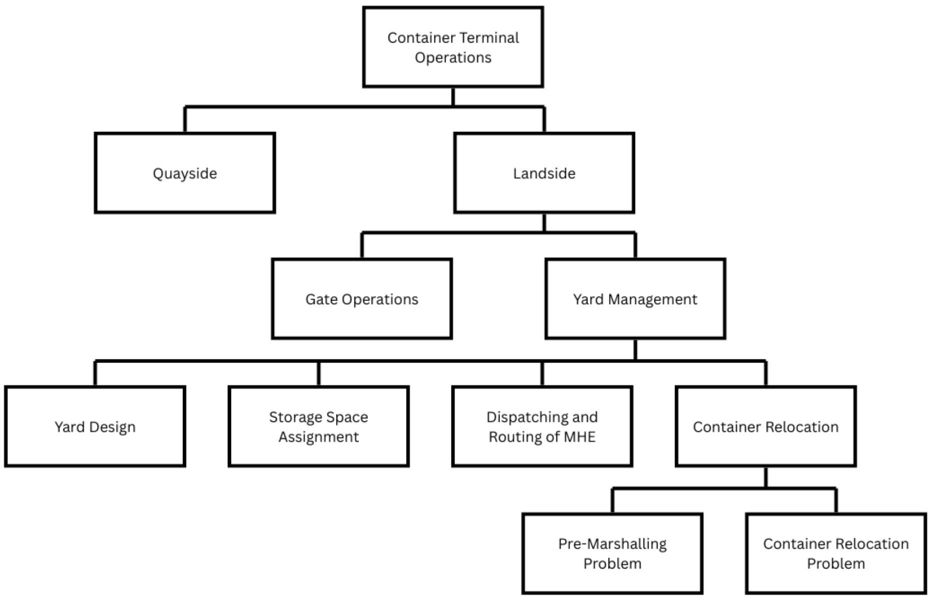
Port terminals are facilities where cargo is transferred from one mode of transportation to another. There are different types of terminals, specialized in handling various types of cargo, including solid bulk (e.g., coal, grain), liquid bulk (e.g., crude oil, chemicals), and containers (WANKE; BARBASTEFANO; HIJJAR, 2011). This study focuses on container terminals in ports, where containers are both transferred from trucks and trains to ships and vice versa.

As containers are transported by different ships, trains, and trucks from multiple companies, origins, and destinations, perfectly synchronizing all inbound and outbound moves is highly complex and practically infeasible (KIZILAY; ELIIYI; VAN HENTENRYCK, 2018; HSU et al., 2017; CARLO; VIS; ROODBERGEN, 2014). Hence, the transfer between modes of transportation can often not be performed immediately, so terminals must also act as temporary buffers in the supply chain, storing containers until they can be transported further. This eliminates the need for synchronization (CASERTA; SCHWARZE; VOSS, 2020).

As described in Chapter 1, container terminals are crucial for the complex, global supply chains of the modern day. Consequently, there is a constant effort to solve the problems that arise in these facilities and improve their efficiency. The Container Relocation Problem (CRP) is one of these problems, and finding better ways of solving it would be a great step towards the improvement of container terminals.

In light of this, this chapter provides the theoretical background for the CRP. It is divided into four main sections. Section 2.1 explains the basic structure and operations of a container terminal and discusses the Operations Research problems that arise in its different areas. The goal is to provide an overview of container terminal operations, including the Container Relocation Problem, as well as other decision problems that are connected to it. Then, Section 2.2 dives into the details of the CRP, presenting its different variants as well as a mathematical formulation. Section 2.3 reviews the literature on solution approaches that have been used by previous authors to solve the CRP. Lastly,

Figure 2.1: Different areas and subdivisions of a typical container terminal



Source: Own representation based on the classification systems of multiple authors.

Section 2.4 explains the theory behind the techniques that are used in Chapter 4, namely Double Deep Q-Learning (DDQN) and Bayesian Optimization (BO), to propose a new solution approach for the Container Relocation Problem.

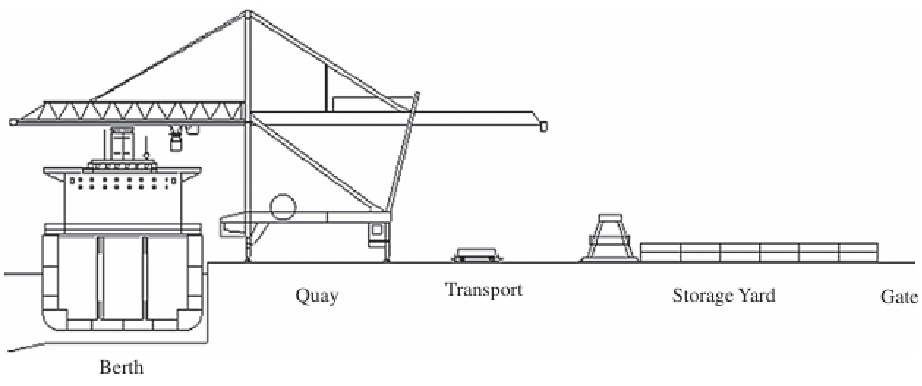
2.1 Overview of a Container Terminal

Following the structure depicted in Figure 2.1, this section starts with a broader overview of container terminals, then progressively narrows its scope, explaining their different areas and the operations performed in them, until it reaches the Container Relocation Problem (CRP).

Typically, a vessel carrying hundreds or thousands of containers is docked along the quay, a platform next to the water which can be divided into multiple berths (CARLO; VIS; ROODBERGEN, 2014). The containers are then unloaded from and loaded onto the vessel by the quay cranes. The unloaded containers are transported by trucks to the yard, where they are stored in stacks. Figure 2.2 provides a simplified illustration of this structure.

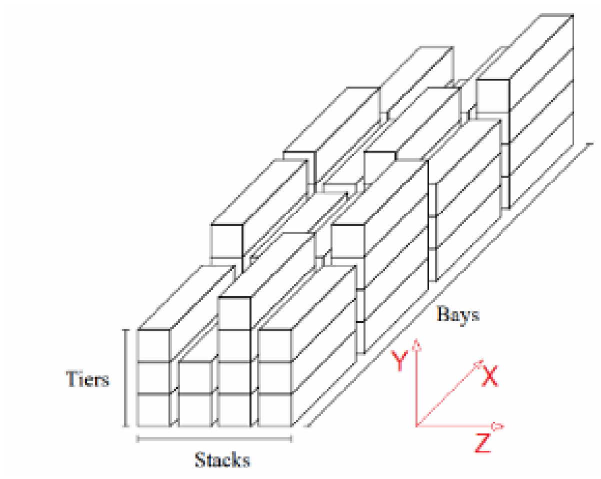
When containers are stored in stacks, either at the yard or on a vessel, there is a specific terminology to describe their position. As illustrated in Figure 2.3, containers are placed on top of each other in multiple tiers, forming a stack. A line of stacks is called a

Figure 2.2: Schematic view of a container terminal



Source: Carlo, Vis, and Roodbergen (2014).

Figure 2.3: Terminology of container storage



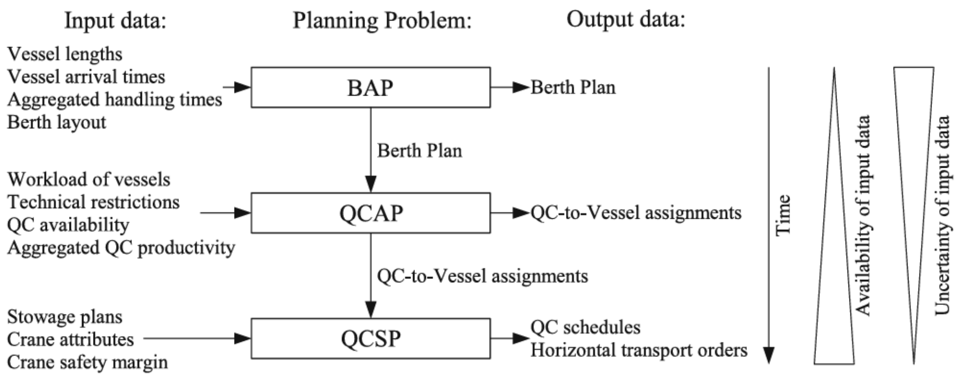
Source: Wei et al. (2021).

bay (WEI et al., 2021). Multiple bays constitute a container block. The set of locations for each container (e.g., container 1 is in bay 2, stack 3, and tier 4, container 2 is in bay 1, stack 2, and tier 1, etc.) is referred to as a configuration or layout. If any container is moved, retrieved or added, it constitutes a new configuration.

When the time comes, containers are removed from the yard, and loaded onto trucks or trains to follow their path inland. Naturally, this process also takes place in the reverse order: a container may arrive at the terminal by train or truck and be stored at the yard until it is loaded onto a vessel.

A container terminal is subdivided into quayside and landside. The former refers to the area where there is direct interaction with the vessels (i.e. berth and quay in Figure 2.2), whereas the latter encompasses activities related to the arrival of trucks and trains,

Figure 2.4: Sequential planning of quayside operations



Source: Bierwirth and Meisel (2010).

as well as the storage of containers at the yard (i.e. storage yard and gate in Figure 2.2) (WEERASINGHE; PERERA; BAI, 2024).

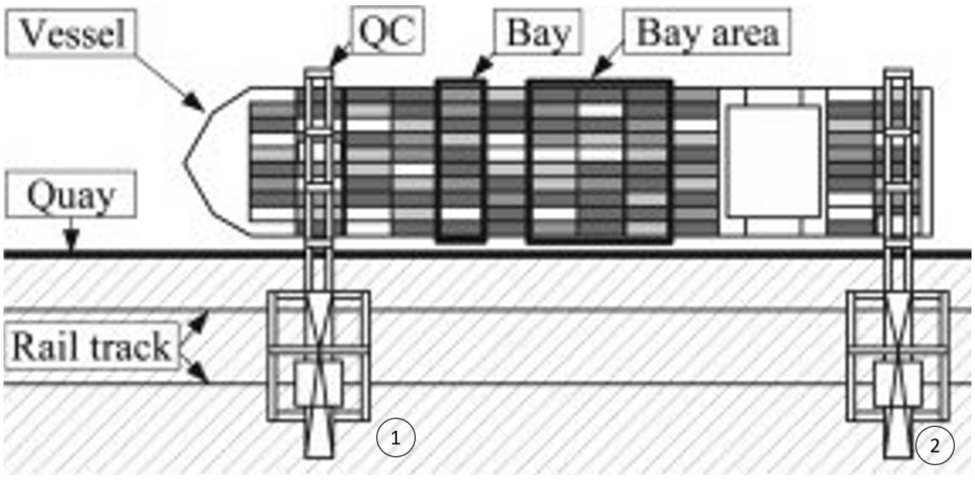
Due to the large number and high complexity of operations performed at container terminals, it is extremely difficult to optimize all of them jointly. Therefore, each Operations Research problem that arises in this context is most often solved either separately or alongside only a few others (KIZILAY; ELIIYI; VAN HENTENRYCK, 2018; HSU et al., 2017; CARLO; VIS; ROODBERGEN, 2014).

2.1.1 Quayside Problems

Quayside problems are directly related to the loading and unloading of containers onto and from vessels, respectively. Here, there are four main problems to tackle: berth allocation (BAP), quay crane assignment (QCAP) and scheduling (QCSP), and container stowage (CSPP) (WEERASINGHE; PERERA; BAI, 2024). As shown in Figure 2.4, the BAP, QCAP, and QCSP can be solved sequentially for planning quayside operations, as the result of each one is necessary as input data for the next one, whereas the CSPP can be solved separately, as it tackles where to store each container, without considering crane availability.

A quay is a platform built alongside a waterway where vessels are docked to be loaded and unloaded. The quay is usually subdivided into berths. The problem of, given a layout of berths and a set of vessels to be served, assigning a berth position and time to each vessel is known as the Berth Allocation Problem (BIERWIRTH; MEISEL, 2010). The goal is to service vessels as well as possible, which is usually formulated as minimizing the sum of waiting and handling times of vessels.

Figure 2.5: Illustration of two quay cranes servicing a vessel



Source: Adapted from Meisel and Bierwirth (2011).

Quay cranes (sometimes referred to as QCs) are mounted on a single rail along the quay to service vessels. This means that, on the one hand, every crane can reach any position in the quay. On the other hand, quay cranes may not cross and switch places, e.g., in Figure 2.5, crane number 2 may not be assigned to a position to the left of crane number 1.

Given a berth plan, which results from solving the BAP, and a set of identical quay cranes, the Quay Crane Assignment Problem (QCAP) consists of assigning cranes to service vessels, while minimizing crane productivity losses, such as crane setups at vessels and crane travel times (BIERWIRTH; MEISEL, 2010). The problem further depends on the loading and unloading requirements of each vessel, as well as on the maximum number of quay cranes that may serve each vessel simultaneously.

After solving the BAP and the QCAP, there are many loading and unloading tasks that must be performed. Determining which quay crane must perform each task at what time is known as the Quay Crane Scheduling Problem (QCSP) (AL-DHAHERI; DIABAT, 2015). The goal is to minimize the completion time of operations, while respecting all restrictions, such as the assignment of QCs to vessels and the fact that cranes may not cross. The tasks can be defined as loading or unloading a bay area (which consists of multiple bays), a single bay, a stack, a group of containers (i.e. a set of containers of similar characteristics, such as destination or priority), or a single container. Defining more granular tasks increases the complexity of the problem, but allows for better solutions to be found (BIERWIRTH; MEISEL, 2010; AL-DHAHERI; DIABAT, 2015).

Lastly, the Container Stowage Planning Problem (CSPP), also known as Master

Bay Plan Problem (MBPP) or Vessel Stowage Planning (VSP), consists of determining how to stow each container, given the available positions on the vessel (AMBROSINO; SCIOMACHEN; TANFANI, 2004; TWILLER et al., 2024). The stowage plan, which results from solving this problem, must take into account the container configuration with which the vessel arrives, the list of containers that must be loaded onto and unloaded from it, as well as requirements regarding the terminal and the vessel itself, e.g., the vessel's weight stability. The goals here are to maximize the capacity utilization of the vessel, minimize operational costs, and deal with uncertainty, such as changes in the containers to be loaded. Additionally, the CSSP may be solved for multiple ports, as the stowage decisions made in one port affect the container configuration that arrives at subsequent ports (TWILLER et al., 2024).

2.1.2 Landside Problems

Landside operations include everything that does not involve directly servicing vessels. The landside is typically subdivided into gate and yard.

2.1.2.1 Gate Operations

The gate is the area that connects internal terminal operations with external stakeholders on land, represented by trucks and trains. Therefore, the gate operations aim to facilitate inbound and outbound cargo flows on these modes of transportation.

The workload at container terminals varies throughout the day based on the arrival schedule of vessels, trains and trucks. If the arrival of many of them coincides, the workload peaks and the service quality may be compromised. To mitigate this problem, a Truck Appointment System (TAS) can be developed (ZEHENDNER; FEILLET, 2014).

In a TAS, terminal operators set up time slots with a maximum number of truck arrivals based on the expected workload for each period. The truck companies can then book a time slot to pick up and/or deliver their cargo (CABALLINI et al., 2020). In this context, terminal operators must decide how many time slots to offer in a day, and how long they should be, considering the impact on the overall service quality (ZEHENDNER; FEILLET, 2014).

2.1.2.2 Yard Management

The yard is a temporary storage area where containers are buffered before being loaded onto ships, trucks, or trains to further progress in the supply chain. Some authors consider the yard as part of the landside, while others consider it a third area, distinct from both the quayside and the landside (WEERASINGHE; PERERA; BAI, 2024; CASERTA; SCHWARZE; VOSS, 2020).

At the yard, containers are stacked on top of each other. This maximizes the space utilization, allowing more containers to be stored in the same area. However, this also makes it so that only the top containers of each stack are directly accessible to the cranes. To access a container that is not on top of the stack, all containers above it (referred to as blocking containers) must first be moved to different stacks, an operation known as relocation or reshuffling.

These relocations are unproductive moves, as they incur additional time and energy costs without directly contributing to the terminal's primary throughput (WU; TING, et al., 2010; GÜVEN; TÜRSEL ELIIYI, 2019). Hence, yard operations focus on storing containers as efficiently as possible to minimize relocations.

Carlo, Vis, and Roodbergen (2014) divide the decision problems in yard management into four typologies, which are usually solved sequentially, namely:

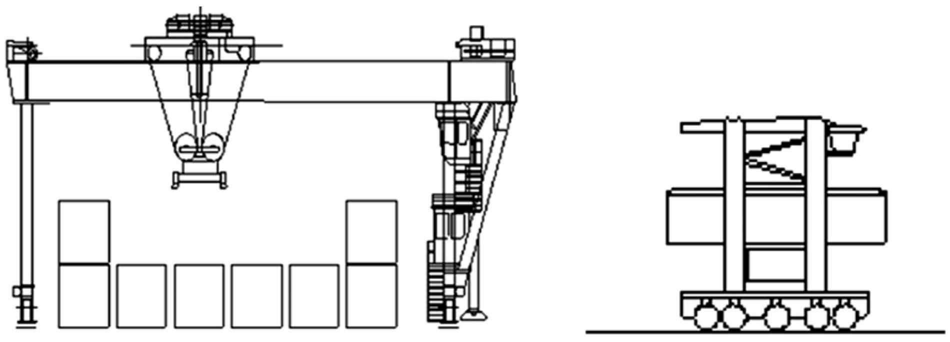
- Yard design
- Storage space assignment for containers
- Dispatching and routing of Material Handling Equipment (MHE)
- Optimizing the relocation of containers

Yard Design

In general, considering the expected throughput of the terminal, the capital investment, and the operating costs, one chooses the level of automation for the terminal. Then, based on the level of automation and the type of containers to be handled, the MHE is selected. Finally, given all that information, the layout of the yard is defined, with a certain number of container blocks, bays, and tiers (CARLO; VIS; ROODBERGEN, 2014).

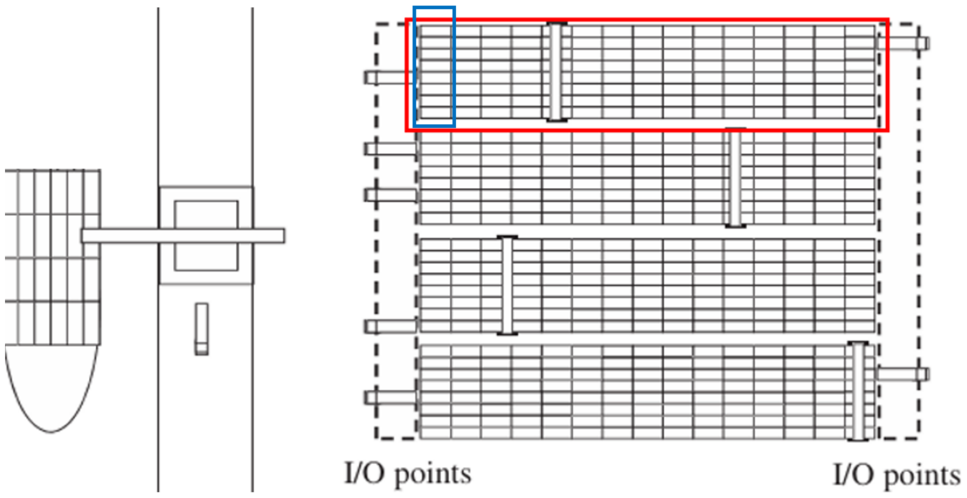
A full review of different Material Handling Equipment and yard layouts is beyond the scope of the present work (for more details, see Carlo, Vis, and Roodbergen (2014)

Figure 2.6: Illustration of a gantry crane (left) and a straddle carrier (right)



Source: Carlo, Vis, and Roodbergen (2014).

Figure 2.7: Illustration of a yard layout using gantry cranes



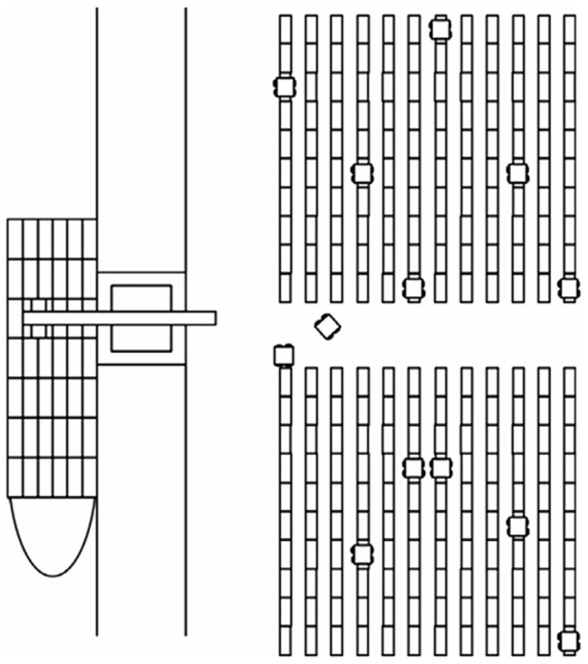
Source: Adapted from Carlo, Vis, and Roodbergen (2014).

and Wiese, Suhl, and Kliewer (2011)). Figure 2.6 illustrates two of the most common MHEs, namely the gantry crane and the straddle carrier.

As depicted in Figure 2.7, gantry cranes are capable of moving both within a bay (highlighted in blue) and across different bays, so they service an entire container block (highlighted in red). The gantry crane transports containers to and from the trucks positioned at the input/output points on each end of the block. These trucks then connect the yard to the quay and to the gate.

Straddle carriers are versatile transport vehicles, used for both handling containers inside the yard and transporting containers between the quay and the yard. However, unlike gantry cranes, they are not capable of reaching over a stack to another behind it. Therefore, they need direct access to each individual stack. As shown in Figure 2.8, a typical layout for straddle carriers must include corridors.

Figure 2.8: Illustration of a yard layout using straddle carriers



Source: Carlo, Vis, and Roodbergen (2014).

Storage Space Assignment

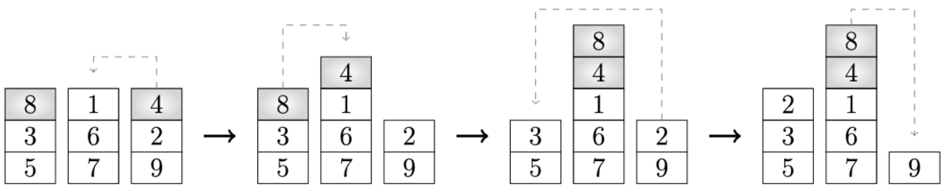
According to Carlo, Vis, and Roodbergen (2014), storage space assignment is about determining the best location inside the yard to store containers, i.e., in which container block, in which bay, and on which stack. The main goal is to reduce the cycle time of yard operations, e.g. by storing a container close to where the vessel that will transport it will be berthed.

This decision is based on information regarding the arrival and departure times of containers. The more reliable the information is, the better space assignment decisions can be made, which makes container handling overall more efficient. Individual container characteristics such as weight, dimensions and content may also influence the assignment.

Dispatching and Routing of Material Handling Equipment

In the dispatching problem, individual storage and retrieval tasks are assigned to specific pieces of equipment (e.g., gantry cranes or straddle carriers), to balance workload across container blocks, respect equipment availability, and maximize overall throughput or minimize completion times. In contrast, the routing problem determines the sequence in which each assigned crane or carrier will execute its tasks, aiming to minimize travel distance or vehicle waiting time (CARLO; VIS; ROODBERGEN, 2014). For these problems, common constraints include avoiding interference (e.g., by maintaining a minimum

Figure 2.9: Illustration of a solution to the Pre-Marshalling Problem



Source: Lersteau and Shen (2022).

safe distance between equipments) and respecting equipment capacity.

Container Relocation

Container relocation, also referred to as reshuffling, is a short-term, operational decision performed at the yard in order to gain access to certain target containers by moving the blocking containers on top of them (CARLO; VIS; ROODBERGEN, 2014).

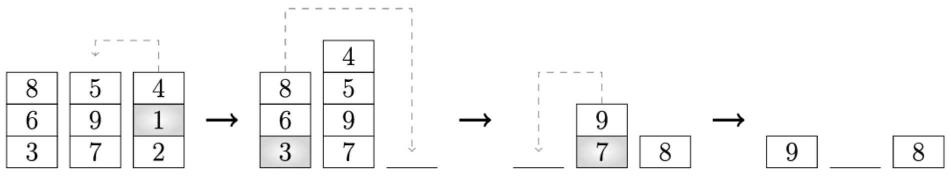
For this type of decision problems each container in a bay is assigned a unique priority label from 1 to N, where N is the total number of containers. A lower numerical label indicates a higher retrieval priority. For instance, container 1 must be retrieved first, followed by container 2, and so on, until the bay is cleared. The priority can be determined by multiple factors, such as container category, estimated departure time, size and weight (CASERTA; SCHWARZE; VOSS, 2020). Then, the goal is to minimize the number of relocations performed when retrieving the containers.

Container relocation can be performed at two moments: (i) before containers are retrieved from the bay, as in the Pre-Marshalling Problem, or (ii) at the same time as containers are retrieved, as in the Container Relocation Problem.

The Pre-Marshalling Problem (PMP) consists of rearranging the containers in a bay, so that, later on, all items can be retrieved without additional relocations (LERSTEAU; SHEN, 2022; CASERTA; SCHWARZE; VOSS, 2020). In this case, as illustrated in Figure 2.9, no containers are retrieved during the operation, so the number of containers remains constant. In the final solution, there must be no misplaced containers, i.e., no containers of lower priority blocking containers of higher priority.

The Container Relocation Problem (CRP), also referred to as Blocks Relocation Problem (BRP), is formulated as the task of finding the optimal set of relocations that allows all containers in a yard to be retrieved in a pre-defined order with as few relocations as possible (CASERTA; SCHWARZE; VOSS, 2012). Here, unlike in the Pre-Marshalling

Figure 2.10: Illustration of a solution to the Container Relocation Problem



Source: Lersteau and Shen (2022).

Problem, containers are retrieved during the operation. This means that, given an initial configuration of container stacks, the goal is to empty the storage area by retrieving all containers in the correct order. As shown in Figure 2.10, in the first step, container 4 is relocated, so that containers 1 and 2 may be retrieved. Then, containers 8 and 6 are relocated, so that container 3 may be retrieved, and so on. This problem was proven to be $\mathcal{NP}$-hard by Caserta, Schwarze, and Voss (2012).

2.2 Detailed Review of CRP

After the brief overview provided in Section 2.1, this section presents the details of the Container Relocation Problem, including its different versions, forms and a mathematical formulation for it.

2.2.1 Variations of the CRP

Several variations of the CRP have been tackled in the literature (LERSTEAU; SHEN, 2022; CASERTA; SCHWARZE; VOSS, 2020), including:

- Static/Dynamic CRP
- Distinct/Duplicate CRP
- Deterministic/Stochastic CRP
- Restricted/Unrestricted CRP

If new containers arrive and must be stored during the instance, the CRP is called dynamic (LERSTEAU; SHEN, 2022). In this case, the solution approach must not only deal with the initial containers, but also with newly-arrived ones and how to store them. Otherwise, the CRP is static.

Regarding the priority labels, if each container has its own unique label, the CRP is distinct; whereas, if there are multiple containers with the same label, it is a case of duplicate CRP (LERSTEAU; SHEN, 2022). In the latter, the containers may have the same priority due to certain similarities (e.g. in weight, size, destination, or type of cargo), and the method must therefore have a mechanism to choose which target should be focused next.

It may also be the case that only some of the priority labels are known at each point in time. For example, in a bay with 20 containers, perhaps only the priorities of the next 10 containers to be retrieved are known, while the priority labels of the other ones are revealed later. This is known as a stochastic CRP, as opposed to a deterministic CRP, where the priorities of all containers are fully known from the start (LERSTEAU; SHEN, 2022).

In the unrestricted version of CRP, any container on top of any stack may be relocated. However, the assumption of a restricted CRP is quite common in the literature. In this case, at any given time, only the container on top of the current target's stack may be relocated (CASERTA; SCHWARZE; VOSS, 2012; LERSTEAU; SHEN, 2022). This restriction significantly reduces the space of possibilities at each move, at the expense of, in many cases, reducing the quality of the final solution.

Finally, as mentioned before, there are different objective functions that can be optimized. The most common ones are the number of unproductive moves (relocations) (CASERTA; SCHWARZE; VOSS, 2012) and crane operation time or distance traveled (SILVA FIRMINO; ABREU SILVA; TIMES, 2016), but other possibilities include optimizing crane fuel consumption (HUSSEIN; PETERING, 2012).

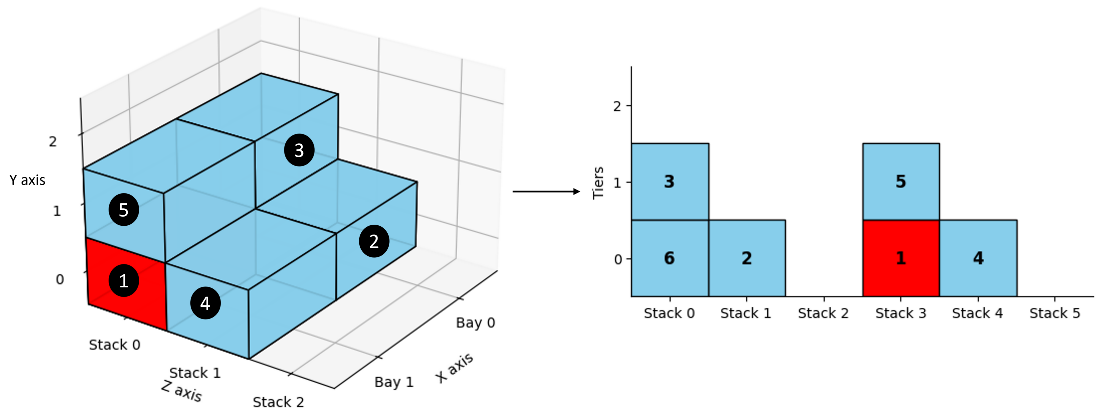
This study tackles the static, distinct, deterministic, and restricted version of the Container Relocation Problem, minimizing the number of relocations.

2.2.2 Dimensions of the CRP

Although container yards are three-dimensional, the CRP is frequently modeled as a two-dimensional problem to simplify the analysis. Indeed, when the objective is to minimize the number of relocations, a 3D yard, with multiple bays, can be equivalently represented as a single 2D bay, provided that the total number of stacks and tiers remains unchanged.

As an example, one may consider a 3D yard with 3 tiers, 2 bays and 3 stacks per bay,

Figure 2.11: Equivalent three-dimensional and two-dimensional representations for the CRP



Source: Own representation.

yielding a total of 6 stacks, as illustrated in Figure 2.11. By numbering these stacks from 0 to 5, one can model the yard as a single bay consisting of 6 stacks, each with 3 tiers. In this 2D representation, if container 5 is relocated from stack 3 to stack 1, the relocation count is incremented by one. The resulting configurations of stack 3 and stack 1 (defined as the ordered sequences of containers) are unaffected by the original spatial arrangement of the stacks.

It is important to note, however, that this equivalence holds only when the relocation cost is measured solely by the number of moves (as is the case in the present work). If the objective function were to incorporate additional factors, such as the distance traveled by the crane, then the spatial layout of the yard would be significant, and the 2D simplification would no longer be valid.

In summary, in the example of relocating container 5 from stack 3 to stack 1, the distance travelled is different in the 2D and in the 3D representations. However, the number of relocations remains the same. Therefore, when minimizing the number of relocations, the 2D model may be used as a simpler alternative to the 3D model.

2.2.3 Integer Programming Formulation for the CRP

Several mathematical formulations have been proposed for the Container Relocation Problem (see Subsection 2.3.1). This thesis tackles the two-dimensional, static, distinct, deterministic, restricted Container Relocation Problem. Galle, Barnhart, and Jaillet (2018) formulated this version of the CRP as an Integer Programming Problem using an improved version of the binary encoding introduced by Caserta, Schwarze, and

Voss (2009). This section provides a brief overview of this formulation to better explain the problem at hand. The authors' notation has been adapted to better fit the thesis at hand. Please refer to Galle, Barnhart, and Jaillet (2018) for more detailed explanations and proofs.

2.2.3.1 Basic Notation and Definitions

Let $\mathcal{W} = \{1, \ldots, W\}$ be the set of stacks, $\mathcal{H} = \{1, \ldots, H\}$ the set of tiers, and $\mathcal{N} = \{1, \ldots, N\}$ the set of containers which must be retrieved in order, from 1 to N. The target container (i.e., the container of highest priority at the moment) is denoted $n \in \mathcal{N}$.

To represent a bay configuration, i.e., the layout in which the containers are organized in the bay, a binary matrix $A \in \{0,1\}^{(N+W)\times N}$ is defined. In this matrix, each element $A_{i,j}$ of row i and column j indicates whether container i is below (but not necessarily directly below) container j, as shown in Expression (2.1).

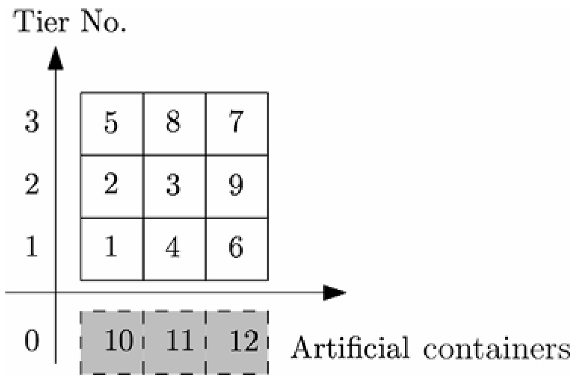
$$A_{i,j} = \begin{cases} 1, & \text{if } i \text{ is below } j \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

Matrix A contains $N + W$ rows and N columns. Rows $N + 1, \ldots, N + W$ represent artificial containers, which sit below each stack and are used to track in which stack each container is located. For example, if artificial container $N + w$ is below container j (i.e., $A_{N+w,j} = 1$), then container j is in stack $N + w - N = w$.

Figure 2.12 shows an example of a bay configuration with $W = 3$ stacks, $H = 3$ tiers, and $N = 9$ containers. Three artificial containers (10, 11, and 12) are added to represent each stack. Expression (2.2) shows how this configuration can be represented using the binary encoding matrix.

In this example, the target container is $n = 1$. The sum of all elements in row 1 of matrix A shows that there are 2 containers on top of container 1, namely containers 2 and 5, as $A_{1,2} = A_{1,5} = 1$. As $A_{10,1} = 1$, container 1 is above artificial container 10, which means container 1 is in stack $10 - N = 1$.

Figure 2.12: Illustration of a CRP configuration with artificial containers



Source: Caserta, Schwarze, and Voss (2009).

$$A = \begin{pmatrix} 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \end{pmatrix} \tag{2.2}$$

2.2.3.2 Logical rules for a feasible CRP

To ensure each bay configuration and its respective encoding follow the rules of the Container Relocation Problem, the following rules are set:

1. If n is the target container, then all containers $i < n$ have already been retrieved. Therefore, at any given moment, only containers $n, ..., N$ remain in the configuration. Hence, only rows $i \in \{n, ..., N + W\}$ and columns $j \in \{n, ..., N\}$ of matrix A have to be considered, i.e., $A \in \{0, 1\}^{(N+W-n+1)\times(N-n+1)}$, because all rows and columns corresponding to containers $i < n$ are zero (Expressions (2.3) and (2.4));

$$A_{i,j} = 0, \forall i \in \{1, ..., n-1\}, \forall j \in \{1, ..., N\} \tag{2.3}$$

$$A_{i,j} = 0, \forall i \in \{1, ..., N+W\}, \forall j \in \{1, ..., n-1\} \tag{2.4}$$

2. Each container must be placed in one stack. Therefore, each container must be above exactly one artificial container;

$$\sum_{i=N+1}^{N+W} A_{i,j} = 1, \forall j \in \{n, ..., N\} \tag{2.5}$$

3. A container may not be below itself, therefore the main diagonal of matrix A must be null;

$$A_{i,i} = 0, \forall i \in \{n, ..., N\} \tag{2.6}$$

4. If i is above j, then j may not be above i;

$$A_{i,j} + A_{j,i} \leq 1, \forall i, j \in \{n, ..., N\} \tag{2.7}$$

5. If two distinct containers i and j are in the same stack w, then either i is above j or j is above i. Here, there are three possible cases:

   (a) If i and j are both in stack w, then $A_{i,j} + A_{j,i} \geq 1$. With rule 4, this becomes $A_{i,j} + A_{j,i} = 1$;

   (b) If one is in stack w and the other is not, then $A_{i,j} + A_{j,i} \geq 0$, which always holds;

   (c) If neither i nor j is in stack w, then $A_{i,j} + A_{j,i} \geq -1$, which always holds;

$$A_{i,j} + A_{j,i} \geq A_{N+w,i} + A_{n+w,j} - 1, \forall w \in \{1, ..., W\}, \forall i, j \in \{n, ..., N\}, i \neq j \tag{2.8}$$

6. If two distinct containers i and j are in different stacks, then neither may be above the other. Here, there are four possible cases:

   (a) If container i is in stack w but j is not, then $A_{i,j} + A_{j,i} \leq 0$, i.e., i and j may not be above each other;

(b) If both i and j are in w, then $A_{i,j} + A_{j,i} \leq 1$, which is the same as rule 4;

(c) If neither i nor j is in w, then $A_{i,j} + A_{j,i} \leq 1$, which is the same as rule 4;

(d) If j is in w but i is not, then $A_{i,j} + A_{j,i} \leq 2$, which always holds;

$$A_{i,j} + A_{j,i} \leq 2 - A_{N+w,i} - \sum_{k=1,k\neq w}^{W} A_{N+k,j}, \forall w \in \{1, ..., W\}, \forall i, j \in \{n, ..., N\}, i \neq j \tag{2.9}$$

7. The number of relocations to retrieve target container n is the number of containers blocking n, i.e. $\sum_{j=n+1}^{N} A_{n,j}$;

8. The maximum height of each stack is H. The height of a stack w is given by the number of containers above the artificial container $N + w$;

$$\sum_{j=n+1}^{N} A_{N+w,j} \leq H, \forall w \in \{1, ..., W\} \tag{2.10}$$

9. A blocking container is one that sits above at least one container of higher priority. The total number of blocking containers in a configuration A is given in Expression (2.11). $B_j \in \{0, 1\}$ indicates whether container j is a blocking container. It is important to note that the number of blocking containers is not $\sum_{i=n}^{N-1} \sum_{j=i+1}^{N} A_{i,j}$ because, even if a container blocks multiple containers of higher priorities, it should count as only one blocking container in the configuration;

$$b(A) = \sum_{j=n+1}^{N} B_j \tag{2.11}$$

Additionally, it is crucial to guarantee that the rules of CRP are respected when moving from one configuration to the next one. In this context, for $n \in \{1, \ldots, N\}$ the following notation is defined:

- $A \in \{0, 1\}^{(N+W-n+1)\times(N-n+1)}$ is the binary encoding for the bay configuration after $n - 1$ retrievals (i.e., when n is the target container);

- $A' \in \{0, 1\}^{(N+W-n)\times(N-n)}$ is the binary encoding for the bay configuration after n retrievals (i.e., when $n + 1$ is the target container);

Then, A and A' must obey the following rules:

10. Restricted assumption: if container j is not blocking target container n, then j may not be relocated. Therefore, column j must be identical in both A and A'. In Expressions (2.12) and (2.13), if j does not block n, then $A'_{i,j} = A_{i,j}$. Otherwise, $A_{i,j} - 1 \leq A'_{i,j} \leq A_{i,j} + 1$, which always holds;

$$A'_{i,j} \geq A_{i,j} - A_{n,j}, \forall i, j \in \{n+1, ..., N\} \tag{2.12}$$

$$A'_{i,j} \leq A_{i,j} + A_{n,j}, \forall i, j \in \{n+1, ..., N\} \tag{2.13}$$

11. A relocated container can not remain in the same stack. In this case, if j is in stack w and blocks n, then $A_{n,j} = A_{N+w,j} = 1$, which forces $A'_{N+w,j} = 0$. Otherwise, $A'_{N+w,j} \leq 1$, which always holds;

$$A_{n,j} + A_{N+w,j} + A'_{N+w,j} \leq 2, \forall j \in \{n+1, ..., N\}, w \in \{1, ..., W\} \tag{2.14}$$

12. As a consequence of the combination of the Last In, First Out (LIFO) principle of stacks and the restricted assumption, if distinct containers i and j both block target n and j is above i in A, then j cannot be above i in the next configuration A'. According to Expression (2.15), if those conditions are met, $A_{n,i} = A_{n,j} = A_{i,j} = 1$, which ensures $A'_{i,j} = 0$. Otherwise, $A'_{i,j} \leq 1$, which always holds;

$$A_{n,i} + A_{n,j} + A_{i,j} + A'_{i,j} \leq 3, \forall i, j \in \{n+1, ..., N\}, i \neq j \tag{2.15}$$

2.2.3.3 The Minimum Number of Relocations

Let $A^{(i)}$ be the binary matrix encoding for the bay configuration after $i-1$ retrievals. Additionally, let $r(A)$ be the minimum number of relocations necessary to retrieve all containers from configuration A, and $r(A^{(1)}, A^{(i)})$ the minimum number of relocations necessary to go from configuration $A^{(1)}$ to $A^{(i)}$. Then, if the number of containers N is lower or equal to the number of stacks W, the minimum number of relocations $r(A)$ to solve configuration A is equal to the number of blocking containers $b(A)$ (Lemma 1 from Galle, Manshadi, et al. (2018)).

In light of Lemma 1, Galle, Barnhart, and Jaillet (2018) define $K = N - W + 1 > 1$, so that, after $K-1$ relocations, the number of containers remaining in the configuration is equal to the number of stacks, i.e., $A^{(K)} \in \{0,1\}^{(N'+W)\times N'}$, where $N' = N - K + 1 = W$. Therefore, Lemma 1 can be applied to $A^{(K)}$, resulting in Expression (2.16).

$$r(A^{(K)}) = b(A^{(K)}) \tag{2.16}$$

Hence, as shown by Expression (2.17), if the goal is to find the minimum number of relocations $r(A^{(1)})$, it is only necessary to solve $r(A^{(1)}, A^{(K)})$, i.e., the first $K-1$ retrievals.

$$r(A^{(1)}) = r(A^{(1)}, A^{(K)}) + r(A^{(K)}) = r(A^{(1)}, A^{(K)}) + b(A^{(K)}) \tag{2.17}$$

2.2.3.4 The Binary Integer Programming Formulation

Now, the rules based on the binary encoding matrix presented in Subsection 2.2.3.2 are converted to a binary integer programming formulation. Given a feasible triplet of stacks, tiers, and containers (W, H, N) and an initial configuration $A^{(1)}$, let $a_{n,i,j}$ be the binary variable that tracks whether container i is below container j when container n is the target (see Expression (2.18)), and b_j be the binary variable that marks whether container j is a blocking container, i.e., whether it is above at least one container of higher priority (see Expression (2.19)).

$$a_{n,i,j} = \begin{cases} 1, & \text{if container } i \text{ is below } j \text{ when } n \text{ is the target} \\ 0, & \text{otherwise} \end{cases}, \quad \forall n \in \{1, ..., K\}, i \in \{n, ..., N+W\}, j \in \{n, ..., N\} \tag{2.18}$$

$$b_j = \begin{cases} 1, & \text{if container } j \text{ is blocking after } K-1 \text{ retrievals} \\ 0, & \text{otherwise} \end{cases}, \quad \forall j \in K+1, ..., N \tag{2.19}$$

Then, Galle, Barnhart, and Jaillet (2018) formulate the Container Relocation Problem as follows:

$$\min \left(\sum_{n=1}^{K-1} \sum_{j=n+1}^{N} a_{n,n,j} + \sum_{j=K+1}^{N} b_j \right) \tag{2.20}$$

s.t.

$$a_{1,i,j} = A^{(1)}_{i,j}, \forall i \in \{1, \dots, N+W\},\ j \in \{1, \dots, N\} \tag{2.21}$$

$$\sum_{w=1}^{W} a_{n,N+w,j} = 1, \forall n \in \{2,\dots,K\},\ j \in \{n,\dots,N\} \tag{2.22}$$

$$a_{n,i,i} = 0, \forall n \in \{2,\dots,K\},\ i \in \{n,\dots,N\} \tag{2.23}$$

$$\begin{gathered} a_{n,i,j} + a_{n,j,i} \leq 1, \\ \forall n \in \{2,\dots,K\},\ i \in \{n,\dots,N\},\ j \in \{n,\dots,N\} \setminus \{i\} \end{gathered} \tag{2.24}$$

$$\begin{gathered} a_{n,i,j} + a_{n,j,i} \geq a_{n,N+w,i} + a_{n,N+w,j} - 1, \\ \forall n \in \{2,\dots,K\},\ w \in \{1,\dots,W\},\ i \in \{n,\dots,N\}, j \in \{n,\dots,N\} \setminus \{i\} \end{gathered} \tag{2.25}$$

$$\begin{gathered} a_{n,i,j} + a_{n,j,i} \leq 2 - a_{n,N+w,i} - \sum_{\substack{r=1 \\ r\neq w}}^{W} a_{n,N+r,j}, \\ \forall n \in \{2,\dots,K\},\ w \in \{1,\dots,W\},\ i \in \{n,\dots,N\}, j \in \{n,\dots,N\} \setminus \{i\} \end{gathered} \tag{2.26}$$

$$\sum_{j=n}^{N} a_{n,N+w,j} \leq H, \forall n \in \{2,\dots,K\},\ w \in \{1,\dots,W\} \tag{2.27}$$

$$b_j \geq a_{K,i,j}, \forall j \in \{K+1,\dots,N\},\ i \in \{K,\dots,j-1\} \tag{2.28}$$

$$\begin{gathered} a_{n+1,i,j} \leq a_{n,i,j} + a_{n,n,j}, \\ \forall n \in \{1,\dots,K-1\},\ j \in \{n+1,\dots,N\}, i \in \{n+1,\dots,N+W\} \setminus \{j\} \end{gathered} \tag{2.29}$$

$$\begin{gathered} a_{n+1,i,j} \geq a_{n,i,j} - a_{n,n,j}, \\ \forall n \in \{1,\dots,K-1\},\ j \in \{n+1,\dots,N\}, i \in \{n+1,\dots,N+W\} \setminus \{j\} \end{gathered} \tag{2.30}$$

$$\begin{gathered} a_{n,n,j} + a_{n,N+w,j} + a_{n+1,N+w,j} \leq 2, \\ \forall n \in \{1,\dots,K-1\},\ j \in \{n+1,\dots,N\},\ w \in \{1,\dots,W\} \end{gathered} \tag{2.31}$$

$$a_{n,n,i} + a_{n,n,j} + a_{n,i,j} + a_{n+1,i,j} \leq 3, \\ \forall n \in \{1, \ldots, K-1\},\ i \in \{n+1, \ldots, N\}, j \in \{n+1, \ldots, N\} \setminus \{i\} \tag{2.32}$$

$$a_{n,i,j} \in \{0,1\}, \forall n \in \{1, \ldots, K\},\ i \in \{n, \ldots, N+W\}, j \in \{n, \ldots, N\} \tag{2.33}$$

$$b_j \in \{0,1\}, \forall j \in \{N+1, \ldots, N\} \tag{2.34}$$

The objective function (2.20) minimizes the number of relocations. The first term counts the number of containers blocking the target n, which corresponds to $r(A^{(1)}, A^{(K)})$ (see rule 7), while the second term computes $r(A^{(K)})$ as $b(A^{(K)})$ (see Lemma 1 in Expression (2.17)).

Constraints (2.21) initialize the binary variable $a_{n,i,j}$ for $n = 1$ using the input configuration $A^{(1)}$. Constraints (2.22), (2.23), (2.24), (2.25), (2.26), (2.27) correspond to rules 2, 3, 4, 5, 6, and 7, respectively. Constraints (2.28) are related to rule 9 and determine that, after $K-1$ retrievals, if container j is above at least one container $i < j$, then j must be considered a blocking container, i.e., $b_j \geq 1$. Constraints (2.29) and (2.30) apply rule 10, while constraints (2.31) and (2.32) apply rules 11 and 12, respectively. Lastly, constraints (2.33) and (2.34) are the binary constraints.

This integer programming formulation clearly defines the static, distinct, deterministic, restricted Container Relocation Problem. In Chapter 3, a different formulation is proposed to tackle the CRP using a Reinforcement Learning framework.

2.3 Review of Solution Approaches to CRP

As mentioned in Subsection 2.1.2.2, the Container Relocation Problem has been proven to be $\mathcal{NP}$-hard. Considering its importance for logistics and the challenges it poses, different approaches have been proposed to solve it.

In this Section, a brief overview of key representative methods is provided. Subsection 2.3.1 handles exact approaches that guarantee optimal solutions to the CRP. Subsection 2.3.2 discusses heuristic methods, i.e., relatively simple, computationally efficient strategies to obtain high-quality, but not necessarily optimal solutions. Lastly, Subsection 2.3.3 examines recent machine learning-based approaches to the problem.

2.3.1 Exact Methods

Caserta, Schwarze, and Voss (2012) provided the first mathematical formulation for the complete set of features of the BRP as a binary linear program, named BRP-I. However, the model was too complex to solve instances of realistic size. By applying the assumption of a restricted BRP (see Subsection 2.2.1), the authors then formulated BRP-II. This assumption is practically reasonable and significantly reduces the solution space, making larger instances possible to solve in reasonable time, albeit at the risk of eliminating optimal solutions.

However, BRP-II exhibited a couple of issues, as identified by Expósito-Izquierdo, Melián-Batista, and Moreno-Vega (2015). Specifically, BRP-II suffers from two main issues: (i) in many cases, multiple containers blocking the target may not be relocated to the same destination stack, and (ii) containers below the current target may also be relocated. Due to these problems, BRP-II would sometimes lead to suboptimal solutions or not be able to solve a feasible configuration (e.g. if a solution had to involve moving multiple blocking containers to the same stack). Expósito-Izquierdo, Melián-Batista, and Moreno-Vega (2015) addressed this in their BRP-II* formulation by adapting the constraints of BRP-II.

Despite fixing the formulation problems of its predecessor, BRP-II* also suffers from a high computational complexity, limiting its application to small and medium-sized instances. Building on the binary encoding framework of Caserta, Schwarze, and Voss (2009), Galle, Barnhart, and Jaillet (2018) proposed an alternative integer programming formulation, termed CRP-I. Their approach introduced a more compact representation in terms of variables and constraints, improving computational efficiency for small and medium-sized instances compared to earlier formulations.

As it is common for $\mathcal{NP}$-hard problems, exact approaches are generally only capable of solving small and medium-sized instances and scale poorly for large problem instances, where solution times grow exponentially with problem size. While exact methods remain valuable for formally defining the problem and deriving theoretical insights, practical large-scale applications often require heuristic or machine learning-based approaches to achieve feasible solution times.

2.3.2 Heuristic Methods

Caserta, Schwarze, and Voss (2012) proposed a simple heuristic rule, known as Min-Max heuristic, which serves both to generate fast, high-quality solutions and to provide an upper bound for their exact formulations. Let i be a blocking container, w a stack, and $\min\{w\}$ the most urgent container in stack w. Then, in this heuristic, the decision rule is as follows:

- Case 1: If there is at least one destination stack w such that $\min(w) > i$, relocate i to one of those stacks, choosing the one with the lowest $\min(w)$
- Case 2: If no destination stack fits Case 1, relocate i to the stack with the highest $\min(w)$

The idea is that in the first case, the relocation will not force any additional relocations in the future because the relocated container i is more urgent than the containers below it, so it will be retrieved before them. In the second case, the relocation will force an additional relocation in the future anyway, so the heuristic tries to minimize its immediate impact by placing i on a stack with containers that will only be retrieved later.

This heuristic achieved high-quality results with low processing times on small- and medium-sized configurations.

Building on this work, Jovanovic and Voss (2014) proposed a simple improvement to the Min-Max. They adjusted the decision rule in situations where a relocation would cause the destination stack to be full. Additionally, the authors developed a chain heuristic that extends the decision-making process: instead of considering only the current blocking container r_t, this method applies a basic heuristic to also evaluate subsequent containers (e.g., r_{t+1}, and beyond). By integrating this forward-looking mechanism, with the improved Min-Max heuristic as the basic heuristic, the method achieves superior overall performance with only a negligible increase in computational effort.

In addition to their BRP-II* formulation, Expósito-Izquierdo, Melián-Batista, and Moreno-Vega (2015) proposed a Branch and Bound (B&B) approach. In their method, each level i of the search tree contains nodes that represent bay layouts in which all containers of priority label lower or equal to i have been retrieved. When exploring a node, the algorithm enumerates all feasible ways of relocating the blocking containers on top of the target. To mitigate the significant computational and memory costs associated with exploring every possibility, the authors define a parameter α. After exploring a node at level i, only the α most promising nodes at level $i+1$ are retained for further exploration. The attractiveness, in turn, is measured by a heuristic rule that estimates the number of additional relocations needed to reach a terminal state.

While this B&B approach yielded very good results for instances of up to size 4x6 (4 stacks and tiers), the α parameter, although effective in making the algorithm significantly more computationally and memory efficient, sacrifices the optimality guarantee. This trade-off is common in heuristic methods applied to $\mathcal{NP}$-hard problems, such CRP.

Jovanovic, Tuba, and Voss (2019) applied an Ant Colony Optimization (ACO) algorithm to solve both the restricted and the unrestricted CRP. This algorithm is based on a colony of artificial ants that expand partial solutions based on a probabilistic transition rule. The authors achieved solutions of optimal or near-optimal quality for small and medium-sized instances, and outperformed state-of-the-art methods for larger sizes with short computing times.

2.3.3 Machine Learning Methods

The Container Relocation Problem has been studied for decades, with traditional optimization techniques serving as the primary solution methods (LIU et al., 2025). In recent years, some researchers have begun incorporating machine learning techniques into their approaches. This section reviews a few notable contributions in this emerging field.

Zhang et al. (2020) proposed a strategy that integrated machine learning, optimization, and heuristic methods. In their approach, a random forest model is employed as a branch pruner within both Branch and Bound and Beam Search frameworks. For a given configuration, two heuristic metrics $R1(s)$ and $R2(s)$ are computed for each stack s. Then, a random forest model, trained on exact solutions from small and medium-sized instances, classifies each stack into different categories using these metrics. Based on the current state of the bay, an appropriate stack category is selected to proceed to the branching process, while the stacks of the remaining categories are pruned. This pruning method significantly reduces the search space, leading to improved computational performance.

Wei et al. (2021) applied a Reinforcement Learning framework, using Proximal Policy Optimization, to estimate an optimal relocation policy for the unrestricted CRP. The authors achieved solutions close to the theoretical optimums, but with much higher computational efficiency. Notably, the authors found the difficulty in tuning hyperparameters to be a disadvantage of the method.

Tang et al. (2024) leveraged Deep Reinforcement Learning to address the three-dimensional stochastic CRP. Their study considered the operation of a 3D container yard over the course of multiple days, where only some containers are retrieved each day, and information about container priority is only revealed on the retrieval day. Their goal was to minimize both the relocation rate and the distance covered during relocations. The authors' strategy consisted of first predicting the pickup dates for the containers, and then using these predictions to derive an optimal retrieval policy.

More recently, Liu et al. (2025) presented a Q-learning-based approach to solve the restricted BRP with duplicate priorities, where different containers may have the same priority label. Their method incorporates a heuristic algorithm for initializing Q-values more effectively and an optimal action-filtering rule for reducing the state-action space, consequently accelerating convergence. These innovations help mitigate the challenges of applying reinforcement learning to combinatorial optimization problems where the number of possibilities grows exponentially with the size of the environment.

Despite these recent developments, the application of machine learning methods to the Container Relocation Problem is still incipient, with most studies focusing on hybrid techniques that integrate machine learning, heuristic and classical optimization methods. With the goal of expanding the research in this field, the present work investigates the applicability of a Double Deep Q-Learning (DDQN) strategy, which has proved itself quite successful in similar contexts (VAN HASSELT; GUEZ; SILVER, 2016), for this problem. More specifically, it is evaluated whether a DDQN agent, with no prior domain-specific knowledge, is capable of learning an efficient relocation policy to solve the static, distinct, deterministic, restricted CRP and minimize the number of relocations in a reasonable timeframe. To the best of the author's knowledge, this is the first time such a research question has been investigated.

2.4 Methodological Foundation

This section presents the theoretical foundation for the techniques that are used in the solution approach (see Chapter 4). Subsection 2.4.1 gives a brief overview of Reinforcement Learning in general and explains the method of Double Deep Q-Learning. Subsection 2.4.2 addresses neural networks and some improvements to their classical structure. Lastly, Subsection 2.4.3 explains Bayesian Optimization, which was used in the present work to optimize machine learning hyperparameters.

2.4.1 Reinforcement Learning

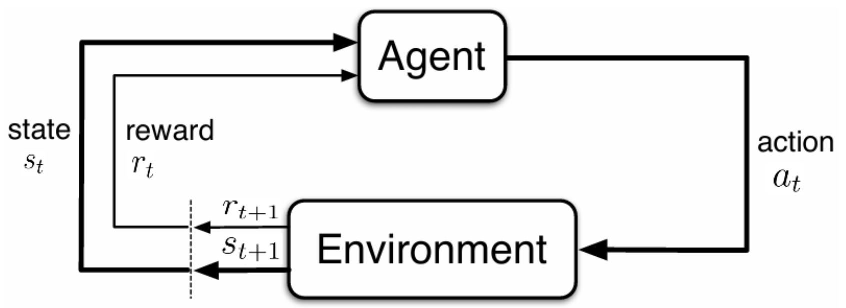
Reinforcement Learning (RL) is one of the three main paradigms that subdivide the broad field of Machine Learning (SUTTON, 2018). In contrast to Supervised Learning, where a model is trained on a labeled dataset to classify data, and Unsupervised Learning, which focuses on identifying hidden structures or patterns in a set of unlabeled data, Reinforcement Learning involves an agent that interacts with an environment to reach a clearly-defined goal. The agent learns an optimal behaviour by using its own experience, evaluating the consequences of past actions, to maximize cumulative rewards (SUTTON, 2018).

2.4.1.1 Markov Decision Processes

To solve a problem using Reinforcement Learning, it is common to model it as a Markov Decision Process (MDP). Sutton (2018) defines an MDP as a classical framework for sequential decision-making problems. Let S be the discrete set of all possible states and $A(s)$ the discrete set of all possible actions when the agent is in state s, then the interaction between an agent and its environment can be described through three key signals, as depicted in Figure 2.13:

1. **State Observation:** At (discrete) time t, the agent observes the current state $s_t \in S$ of the environment.
2. **Action:** Based on s_t, the agent selects an action $a_t \in A(s_t)$.
3. **State Transition and Reward:** Following action a_t, the environment transitions to a new state s_{t+1} and provides a reward r_{t+1}.

Figure 2.13: Interaction between agent and environment in an MDP



Source: Adapted from Sutton (2018).

At each step, the next state s_{t+1} depends exclusively on the current state s_t and the action taken a_t. This is known as the Markov property. The obtained reward r_{t+1} can depend on the current state s_t, the action taken a_t and on the state reached s_{t+1}.

In stochastic cases, an MDP is also characterized by a transition model $T(s, a, s')$, which gives the probability $P(s'|s, a)$ of reaching s', when taking action a in state s. However, in the context of the present work, the MDP is deterministic, i.e., any action is guaranteed to lead to the intended result.

"Environment" is a broad term that encompasses all possible states, actions, and associated rewards. The agent interacts with its environment, taking actions and changing the state, until it reaches a terminal state, which means its task is over. When the agent takes an action a at state s, one can say the agent has visited the state-action pair (s, a). The period from the initial state until the terminal state is called an episode. In the context of the work at hand, for example, the episode spans from the initial configuration, with all containers, until all containers have been retrieved and the bay is empty.

A policy π is a mapping $\pi : S \to A$ that, given a state, returns an action to be taken. The Q-value of a state-action pair (s, a) under policy π, denoted $Q_\pi(s, a)$, represents the expected sum of discounted rewards obtained when starting at state s, taking action a and following policy π until the end of the episode (see Expression (2.35)). The Q-value gives a sense of the utility of taking action a at state s. The discount factor $\gamma \in [0, 1]$ is a parameter that determines the relative importance of future rewards compared to immediate rewards.

$$Q_\pi(s, a) = E\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1}(s_t, a_t)\right] \tag{2.35}$$

The goal in an MDP is to find the optimal policy $\pi^*(s)$ that maximizes the expected cumulative reward (see Expression (2.36)).

$$\pi^*(s) = \operatorname{argmax}_a E\left[\sum_{t=0}^{\infty} \gamma^t r_{t+1} Q(s_t, a)\right] \tag{2.36}$$

2.4.1.2 Double Deep Q-Learning

There exist multiple Reinforcement Learning approaches to find the optimal policy for an MDP. In the present work, Double Deep Q-Learning (DDQN) (VAN HASSELT; GUEZ; SILVER, 2016) was used to tackle the Container Relocation Problem. To understand DDQN, one must first understand classical Q-Learning and Deep Q-Learning (DQN).

Q-Learning

Q-Learning is an off-policy, temporal-difference (TD) learning algorithm introduced by Watkins (1989) that iteratively updates the Q-values for state-action pairs. The term "off-policy" refers to the fact that the algorithm learns the Q-values for the optimal policy π^*, even if the agent is following a different (exploratory) policy; whereas Temporal-Difference Learning is a method of updating Q-value estimates based on the difference between consecutive predictions.

In tabular Q-Learning, the agent stores a table with the Q-value $Q(s, a)$ of every state-action pair. These values are typically initialized arbitrarily and updated as the agent interacts with the environment. The update rule is given in Expression (2.37), where α is the learning rate and the error term $\delta = \tilde{Q}(s, a) - Q(s, a)$ reflects the discrepancy between the current estimate $Q(s, a)$ and the target estimate $\tilde{Q}(s, a)$.

$$Q(s, a) \leftarrow Q(s, a) + \alpha[\tilde{Q}(s, a) - Q(s, a)] \tag{2.37}$$

Essentially, this creates a control loop in which the target, that should be the actual optimal Q-value for the state-action pair, is not known. $\tilde{Q}(s, a)$ estimates this target according to Expression (2.38) using the reward r_{t+1} and the best Q-value for the next step, which is obtained by choosing the action a that maximizes the Q-value at s_{t+1}. This mechanism, where the update rule uses current estimates to improve future ones, is the essence of bootstrapping in temporal-difference learning (SUTTON, 2018).

$$\tilde{Q}(s_t, a_t) \equiv r_{t+1} + \gamma \max_a Q(s_{t+1}, a) \tag{2.38}$$

The algorithm continuously refines its Q-value estimates by comparing the current estimate with the newly observed target. This iterative process eventually leads the Q-values to converge to their optimal values under certain conditions. Watkins and Dayan (1992) proved that, for discrete environments, Q-learning converges to the optimal Q-values provided that all state-action pairs are visited infinitely often and the learning rate is appropriately decayed.

The pseudocode in Algorithm 1 (adapted from (SUTTON, 2018)) summarizes the Q-Learning algorithm. The epsilon-greedy strategy is explained in detail in Subsection

2.4.1.3.

Algorithm 1 Q-Learning Algorithm, adapted from Sutton (2018)

```
1: **Input:** $\alpha, \epsilon$
2: Initialize a table with random Q-values $Q(s, a)$
3: **for** each episode **do**
4:     Initialize the environment
5:     Let $t = 1$
6:     **while** state $s$ is not terminal **do**
7:         Choose $a \in A(s)$ using an $\epsilon$-greedy strategy
8:         Take action $a$
9:         Observe $r_{t+1}$ and $s_{t+1}$
10:        $Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$
11:        $t \leftarrow t + 1$
12:    **end while**
13: **end for**
```

Function Approximation and Deep Q-Learning

In complex environments, tabular Reinforcement Learning methods face two main challenges due to the large number of possible states and actions: (i) storing a Q-value for every state-action pair can be too computationally expensive, and (ii) it may be infeasible to visit every single state-action pair often enough to ensure reliable estimates and guarantee convergence (SUTTON, 2018).

To address these challenges, function approximation (also known as value approximation) consists of using, instead of a table, a function $Q(s, a; \theta)$ of parameters θ as an approximator to the Q-value of a state-action pair. These approximators can take various forms, such as linear functions, decision trees, or, most notably, deep neural networks (SUTTON, 2018).

The main advantage of function approximation lies in the fact that, typically, the number of parameters that must be saved to memory for the chosen function is much lower than the number of values stored with tabular methods. Additionally, function approximation allows the estimation of Q-values for state-action pairs that have not been explicitly visited during training, and parameter updates can simultaneously affect the estimates for many pairs.

Mnih et al. (2015) incorporated the ideas of experience replay and a target neural network to the Q-learning framework to address the instability issues commonly encountered when using non-linear value function approximators in RL. This laid the foundation for Deep Q-Learning (DQN).

Ideally, the agent should learn on independent data points. However, subsequent observations made by the agent while exploring the environment are highly correlated. The experience replay, replay memory or replay buffer is essentially a dataset that stores transitions. Each transition is a tuple containing the current state s_t, the action chosen a_t, the resulting next state s_{t+1} and the reward obtained r_{t+1}. Instead of using a sequence of correlated transitions, the agent is trained on a randomly selected minibatch of transitions from the experience replay. This makes it so that the training data is quasi-independent, improving the learning process (MNIH et al., 2015).

In DQN, there are two neural networks. The online network, also known as main network, with parameters θ is used for selecting actions during the interaction with the environment. Meanwhile, the target network with parameters θ^- is used for computing the bootstrapped target Q-value $\tilde{Q}(s,a)$, as shown in Expression (2.39).

$$\tilde{Q}^{\text{DQN}}(s_t,a_t) \equiv r_{t+1} + \gamma \max_a Q(s_{t+1},a;\theta^-) \tag{2.39}$$

The online parameters θ are updated frequently using a neural network optimization strategy, such as Stochastic Gradient Descent (SGD) (AMARI, 1993), with the current network output $\hat{y} = Q(s,a;\theta)$ compared against the expected output $y = \tilde{Q}(s,a)$. Meanwhile, the target network's parameters θ^- are updated less frequently by copying the online parameters θ. This update mechanism contributes to stabilize learning (MNIH et al., 2015).

Double Deep Q-Learning

Van Hasselt, Guez, and Silver (2016) demonstrated that the DQN algorithm is prone to overestimate Q-values. This phenomenon, denoted overestimation bias, arises because the maximum operator uses the same estimates to select and evaluate actions, which is likely to lead to overly optimistic value estimates, degrading performance. To solve this, the authors proposed a relatively simple change when bootstrapping $\tilde{Q}(s,a)$: to use the online network (with parameters θ) for choosing the best action and the target network (with parameters θ^-) for evaluating the state-action pair. Consequently, the new bootstrap rule is as shown in Expression (2.40).

$$\tilde{Q}^{\text{DDQN}}(s_t,a_t) \equiv r_{t+1} + \gamma Q(s_{t+1}, \operatorname{argmax}_a Q(s_{t+1},a;\theta);\theta^-) \tag{2.40}$$

This adjustment helps reduce overestimation by ensuring that the action selection and evaluation processes are handled by distinct networks, thereby improving the stability and overall performance of the learning algorithm.

2.4.1.3 Epsilon-Greedy Policy

According to Sutton (2018), to maximize its rewards, an agent must focus on actions that it has already tried and worked well, a behaviour known as exploitation. However, to discover actions that work well, the agent must try new actions, i.e., explore the environment. This constitutes the classic trade-off between exploration and exploitation in Reinforcement Learning.

Epsilon-greedy policies provide a straightforward method to balance this trade-off during training. According to this policy:

1. **Exploration:** with probability ϵ, the agent selects a random action, which prompts it to try new actions and explore new states.
2. **Exploitation:** with probability $1 - \epsilon$, it chooses the action that maximizes the Q-value, leveraging prior experience to obtain high rewards.

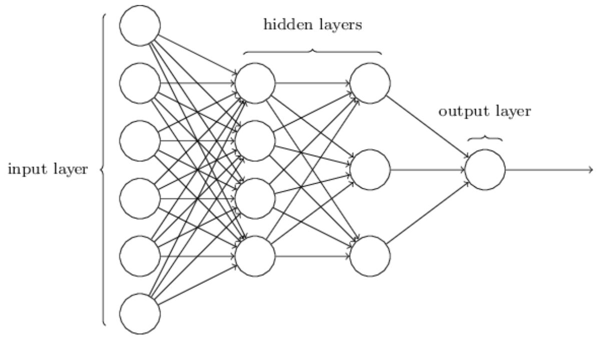
The parameter $\epsilon \in [0, 1]$, that serves as a probability threshold to balance exploration and exploitation, is set as a hyperparameter for the training process. Typically, ϵ starts quite high, to encourage exploration, and slowly decays throughout training, to focus on exploitation, as the agent's knowledge improves.

2.4.2 Neural Networks

Goodfellow, Bengio, and Courville (2016) define a neural network as a function $\hat{f}(x, \theta)$ that learns the best parameters θ to approximate an unknown objective function $f(x)$. In the past decades, neural networks have proven themselves quite useful, especially to approximate extremely complex functions.

A simple fully-connected, feedforward neural network, also known as Multilayer Perceptron (MLP) (ROSENBLATT et al., 1962), consists of layers of interconnected neurons, as shown in Figure 2.14. Each neuron takes an input vector x, computes its dot product with a weight vector w, and adds a bias vector b. Then, the result is typically passed through a non-linear activation function σ. Expression (2.41) summarizes the operation each neuron applies to the input.

Figure 2.14: Representation of a fully-connected, feedforward network with one input layer, two hidden layers, and one output layer



Source: Nielsen (2015).

$$\sigma(w \cdot x + b) \tag{2.41}$$

Let $X = \{x_1, \ldots, x_n\}$ be the set of n training input vectors, $\hat{Y} = \{\hat{f}(x_1, \theta), \ldots, \hat{f}(x_n, \theta)\}$ the set of their network outputs for parameter vector θ, and $Y = \{f(x_1), \ldots, f(x_n)\}$ the set of the corresponding expected outputs. For each example, the expected output $f(x)$ and the output $\hat{f}(x, \theta)$ for the current network parameters are compared using a loss function (NIELSEN, 2015). The loss function for the entire network is computed by averaging the loss function for each example (or for a random sample of those examples).

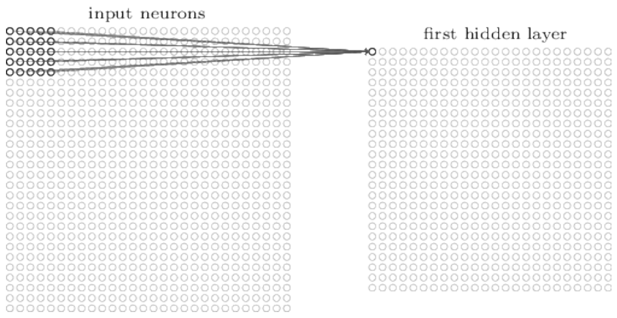
In the work at hand, the Huber Loss Function, introduced by Huber (1964), is used (see Expression (2.42) for a single training example i). This loss function applies the Mean Squared Error function when the difference between actual and expected outputs is small, and the Mean Absolute Error function when this difference is large. Consequently, the Huber loss function is more robust to outliers.

$$\mathcal{L}_i = \begin{cases} \frac{1}{2}(\hat{f}(x_i, \theta) - f(x_i))^2, \text{ for } |\hat{f}(x_i, \theta) - f(x_i)| \leq 1 \\ |\hat{f}(x_i, \theta) - f(x_i)| - \frac{1}{2}, \text{ otherwise.} \end{cases} \tag{2.42}$$

The result of the loss function is then used to update the network's weights and biases using the backpropagation algorithm (NIELSEN, 2015).

It has been proven that neural networks with at least one hidden layer and a non-linear activation function are capable of approximating any continuous function with arbitrary precision (HORNIK; STINCHCOMBE; WHITE, 1990). Despite demonstrating the potential of neural networks, this proof, known as the Universal Approximation Theorem, does not determine what architecture should be used for each objective function, nor how

Figure 2.15: Representation of a kernel applied to one local receptive field



Source: Nielsen (2015).

long it would take for the network to converge. Therefore, many strategies have been developed to improve the basic MLP design (NIELSEN, 2015). In the next subsections, three strategies relevant for the present work are introduced.

2.4.2.1 Convolutional Layers

Nielsen (2015) describes convolutional layers as a special type of neural network architecture that takes advantage of the spatial structure in the data. In the two-dimensional case, each neuron in the next layer is connected to a rectangular subset of the neurons in the previous layer, instead of being connected to all of them, as in the fully-connected case of Figure 2.14. The connections represent the operation in Expression (2.41).

The subset of neurons in the previous layer that are connected to a certain neuron in the next layer are called that neuron's local receptive field. Figure 2.15 illustrates the connections of a single hidden neuron (i.e., a neuron in the hidden layer) to its local receptive field in the input layer. In this case, the receptive field for the hidden neuron immediately to its right would be similar (with the same 5x5 size, same weights and biases), but dislocated one column to the right. The same logic applies for the hidden neuron immediately below it: the local receptive field would be dislocated one row down. In this example, the stride length is one, i.e., moving one neuron to the right/down, also moves the local receptive field by 1 in the appropriate direction. The layer which results from this convolution operation (e.g. the "first hidden layer" in Figure 2.15) is often called a feature map.

One important aspect of this operation is the fact that the shape of the input is reduced. For example, in Figure 2.15, a 5x5 kernel is applied to a 28x28 input, resulting in a 24x24 feature map. When it is desirable to maintain the original shape, one may apply padding to the input, i.e., augment its shape with dummy inputs at the edges, so that the output feature map retains the input's shape.

Neural networks that leverage convolutional layers as one of their main components are often referred to as Convolutional Neural Networks (CNNs).

2.4.2.2 Batch Normalization

Batch normalization was first proposed by Ioffe and Szegedy (2015) to address issues caused by the internal covariance shift, a change in the distribution of the activations in a neural network as the parameters of previous layers are updated during training. This shift can slow down convergence and hinder training.

Batch normalization consists of normalizing the activations output by a given layer to zero mean and constant standard deviation before passing them as input to the next layer. This technique has been successfully applied in thousands of studies, and has been empirically demonstrated to accelerate the convergence of neural networks, and improve their accuracy in many scenarios. However, despite its widespread practical success, there is still little theoretical consensus regarding the reason for such improvements (see Bjorck et al. (2018) and Santurkar et al. (2018)).

2.4.2.3 Residual Connections

He et al. (2016) proposed the concept of residual connections to tackle the problem of accuracy degradation in deep neural networks, where increasing network depth can lead to higher training error. Given a certain block of layers that applies a transformation $F(x)$ to an input x, the residual connection adds the input x itself to the final output of the block y (see Expression (2.43) and Figure 2.16).

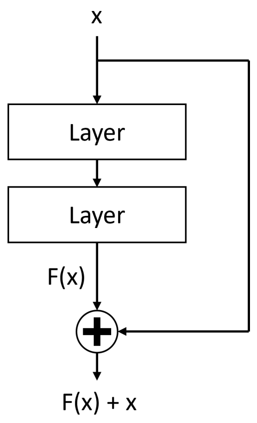
$$y = F(x) + x \tag{2.43}$$

Residual networks have since been successfully applied in a wide range of applications, facilitating optimization and allowing deeper networks to achieve higher accuracies.

2.4.3 Bayesian Optimization

Bayesian Optimization (BO) is a global optimization method for black-box functions, i.e., functions for which no closed-form expression or gradient information is available (BROCHU; CORA; DE FREITAS, 2010; FRAZIER, 2018). Typically, BO is most effec-

Figure 2.16: Illustration of a residual block



Source: Adapted from He et al. (2016).

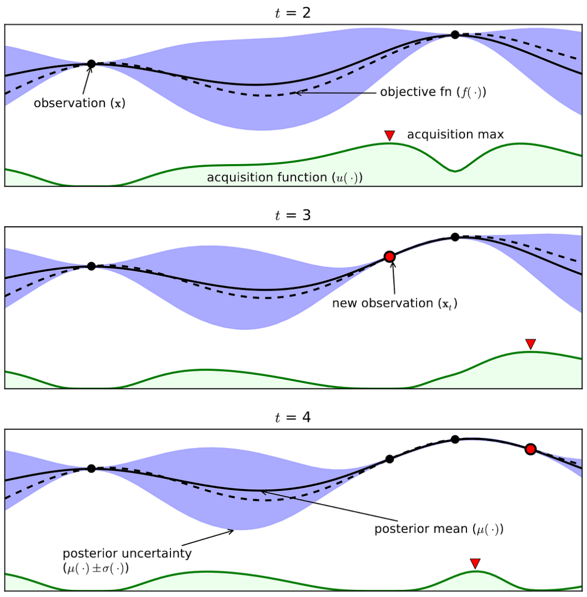
tive for computationally expensive functions, i.e., when evaluating the objective function hundreds of times is not feasible, and a sample-efficient optimization strategy is needed.

Essentially, BO works by assuming that the objective function f can be modeled as a Gaussian Process (GP) prior. This prior places a distribution over the function space, and the GP posterior is updated with every new observation $(x, f(x))$ using Bayes' Theorem. Hence, Bayesian Optimization can also be thought of as fitting a surrogate model to approximate the objective function, refining this fit as more data is obtained. Figure 2.17 illustrates this process, where the mean function of the GP posterior, i.e., the surrogate model, is represented by the full black line and the blue shaded area shows the uncertainty.

After each update to the posterior, the next point to be evaluated, denoted x_{n+1}, is chosen by maximizing an acquisition function $u(x)$ (represented by the green curve in Figure 2.17). There are many different options for acquisition functions, such as Expected Improvement (EI) and Probability of Improvement (PI). However, in essence, all acquisition functions estimate the quality of every possible new evaluation point (BROCHU; CORA; DE FREITAS, 2010; FRAZIER, 2018). These functions address the trade-off between evaluating areas near good points that have already been found (exploitation) and areas with high uncertainty (exploration). Unlike the objective function, acquisition functions are designed so that finding their optimum is fairly simple.

In the present work, Bayesian Optimization is used to tune the hyperparameters of the Double Deep Q-Learning approach. As training a Deep Reinforcement Learning model can be computationally expensive and time-consuming, BO helps reduce the total number

Figure 2.17: Example of three steps of Bayesian Optimization in a simple 1D problem



Source: Brochu, Cora, and De Freitas (2010).

of training runs required to find effective hyperparameters. This approach is detailed in Section 4.4.

3 PROBLEM MODELING

Considering the theoretical background presented in Chapter 2, this work focuses on the static, distinct, restricted, deterministic Container Relocation Problem, as a crucial yard management operation for the efficiency of the terminal. Given N containers stored in W stacks of maximum height H, the goal is to retrieve all containers in a pre-defined order, with as few relocations as possible. This chapter presents how this problem was modelled as a Markov Decision Process (MDP), so that it could be solved using a Double Deep Q-Learning approach.

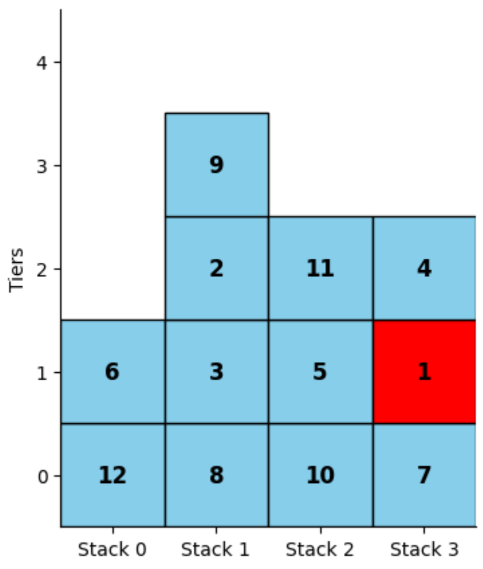
3.1 Environment

As detailed in Subsection 2.2.2, the environment of this study is a two-dimensional container bay. Let W be the number of stacks, H the number of tiers and N the number of containers in this bay. The stacks are numbered from 0 to $W-1$, the tiers from 0 to $H-1$, whereas the priority labels range from 1 to N. Figure 3.1 shows an example of a bay with $W = 4$ stacks, $H = 5$ tiers and $N = 12$ containers. The target container, i.e., the one with the highest priority, is highlighted in red. In this example, it is important to note that, despite being currently empty, tier 4 can still be used.

For any instance of CRP, the feasibility condition, shown in Constraint (3.1) must be satisfied. This ensures there are at least $H-1$ empty slots to fit a maximum of $H-1$ blocking containers on top of the target, thereby guaranteeing the instance's feasibility (CASERTA; SCHWARZE; VOSS, 2012).

$$W \times H \geq N + (H-1) \tag{3.1}$$

Figure 3.1: Illustration of a bay for the CRP with the target container highlighted in red



Source: Own representation.

3.1.1 State Representation

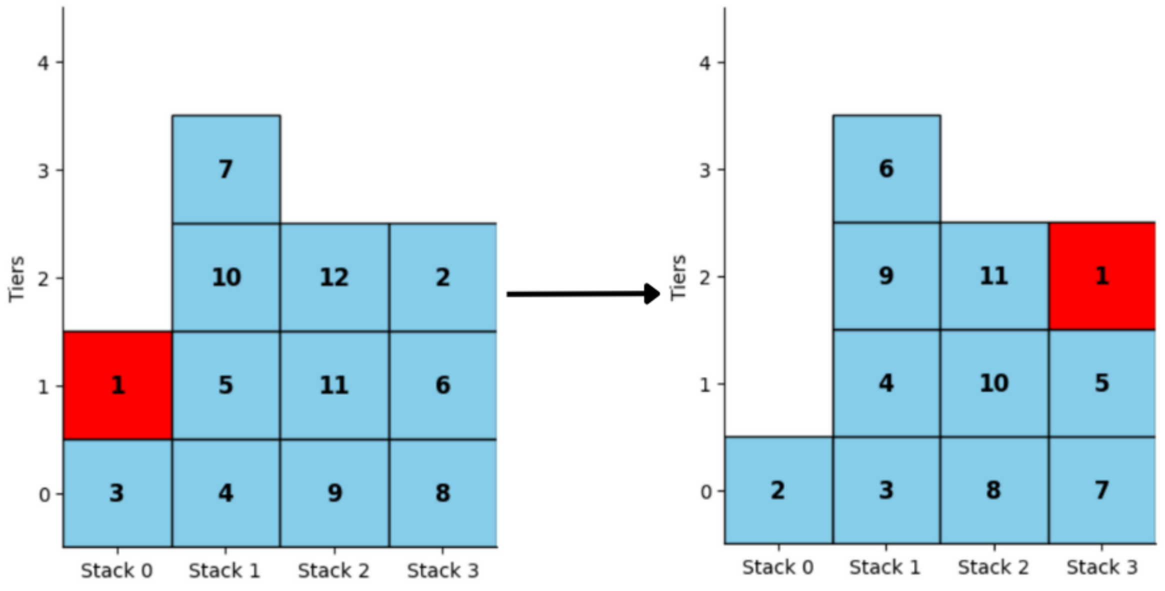
In Reinforcement Learning, it is often the case that the agent only has partial information on the environment. Here, however, the environment is fully observable, i.e., the agent has full knowledge of the position and priority of every container at each step.

In this formulation, container stacks are represented by arrays. The first element corresponds to the bottom container (in tier zero), the second element to the container in tier 1, and so on until the top container. Thus, a bay is represented by a $W \times H$ two-dimensional array of stack arrays, indexed from 0 to $W - 1$. Empty slots are marked zero. The bay in Figure 3.1 is described as:

$$[[12, 6, 0, 0, 0], [8, 3, 2, 9, 0], [10, 5, 11, 0, 0], [7, 1, 4, 0, 0]]$$

Once the container with priority label 1 is retrieved, only containers 2 through N remain, and container 2 becomes the target. However, that is equivalent to a configuration where all containers have their priority labels reduced by 1. Therefore, as shown in Figure 3.2, after every retrieval, the priority labels of all remaining containers are reduced by one, so that the target always has priority label 1. This transformation simplifies the agent's learning task by eliminating the need to identify a new target after each retrieval, without compromising the overall applicability of the model.

Figure 3.2: Updating the priority labels of all containers after a retrieval



Source: Own representation.

3.1.2 Set of Actions

For a container bay of size $W \times H$, the set of possible actions at a state s is represented as a one-dimensional vector of length $2W$, where each action a_i is indexed from 0 to $2W-1$.

$$A(s) = a_0, a_1, ..., a_{2W-1}$$

The first half of the vector represents relocations, while the second one represents retrievals:

1. **Relocations:** for $0 \leq i \leq W-1$, a_i is a relocation from the stack with the current target container to stack i.
2. **Retrievals:** for $W \leq i \leq 2W-1$, a_i is a retrieval of the top container in stack $i-W$.

It is important to note that, for a given state s, the actions in the set $A(s)$ are not all legal. Taking the bay illustrated in Figure 3.1 as an example, the set of possible states is:

$$A(s) = \{a_0, a_1, a_2, a_3, a_4, a_5, a_6, a_7\}$$

Here, a_3 is illegal because it would mean relocating container 4 from stack 3 back to stack 3, whereas a_4, a_5, a_6, a_7 are also illegal, as retrieving a container is not allowed while the target is blocked. Conversely, if the current target container is at the top of its stack, such as in Figure 3.2, the only legal action is to retrieve it. In the proposed approach, whenever the agent chooses an action, whether that is done randomly (exploration) or by maximizing Q-values (exploitation), the illegal actions are first filtered out, so that only legal actions are available.

3.1.3 Rewards

As discussed by Sutton (2018), the success of an RL model strongly depends on how well the reward frames the agent's goal. Moreover, the reward signal should not give the agent any information on how to perform a task (for that would mean interfering with its learning process), but rather provide feedback once the agent achieves its goal or fails it.

In the Container Relocation Problem, the objective is straightforward: to retrieve all containers in a predefined order with as few relocations as possible. Hence, the reward signal is defined as follows:

- **Retrieval reward:** the agent receives a reward of +1 for correctly retrieving a container.
- **Relocation penalty:** the agent receives a reward of -1 for every relocation.

This simple reward structure directly encourages the agent to minimize unnecessary relocations and focus on achieving retrievals.

4 SOLUTION APPROACH

This chapter details how the concepts explained in Section 2.4 are applied to investigate the applicability of a Double Deep Q-Learning approach to the Container Relocation Problem.

Specifically, Section 4.1 describes the two neural network models designed. Section 4.2 discusses the training loop that the agent goes through. Section 4.3 elucidates the method used to test the quality of the agents trained using the CM dataset (CASERTA; SCHWARZE; VOSS, 2009) as benchmark. Lastly, Section 4.4 details how the hyperparameters were tuned using Bayesian Optimization.

4.1 Neural Network Architectures and Components

For the Double Deep Q-Learning agent, two different neural network architectures were compared: (i) one simpler, baseline fully connected architecture, denoted Simple-Net, and (ii) a more complex network, denoted Complex-CNN, that leverages the concepts of convolutional layers, residual connections, and batch normalization. The motivation for this comparison is to investigate not only the DDQN framework for the CRP, but also the impact of a more complex architecture on performance.

For both networks, the Huber loss function (HUBER, 1964) was used. This loss function is more robust to outliers, which may appear especially in the early stages of training, when Q-value estimates are still quite noisy (see Subsection 2.4.2).

Additionally, Rectified Linear Unit (ReLU), one of the most common and most successful choices of activation function for Deep Learning (LI et al., 2021), was used in both architectures.

Finally, both networks also use Adam, a standard optimizer in the field of Neural Networks (LI et al., 2021).

Table 4.1 summarizes the similarities and differences between the employed Simple-

Table 4.1: Main differences and similarities between Simple-Net and Complex-CNN

	Simple-Net	Complex-CNN
Activation Function	ReLU	ReLU
Loss Function	Huber Loss	Huber Loss
Optimizer	Adam	Adam
Convolutional Layers	Absent	Present
Residual Connections	Absent	Present
Batch Normalization	Absent	Present

Source: Own representation.

Net and Complex-CNN. Subsections 4.1.1 and 4.1.2 discuss the particular details of each architecture.

4.1.1 Simple-Net Architecture

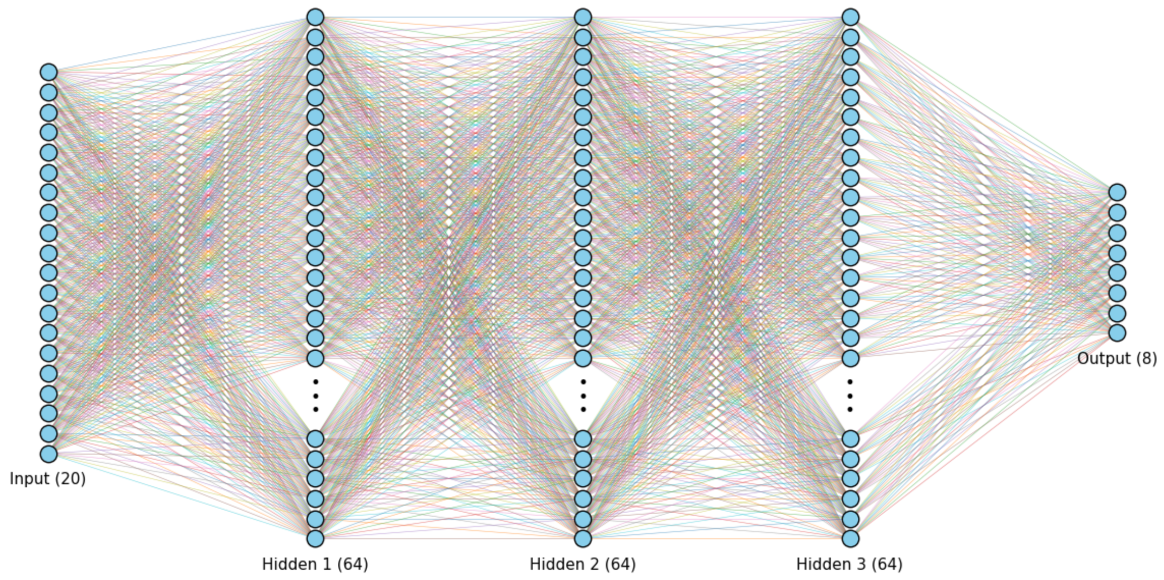
As the name suggests, Simple-Net has a simple, fully connected architecture. The network receives the flattened bay observation, i.e. a one-dimensional vector of length $W \times H$, as input. This input is processed by three fully-connected hidden layers, each with 64 neurons. The network outputs a one-dimensional vector of length $2W$, with the Q-values for each possible action, indexed in the same manner as $A(s)$ (see Subsection 3.1.2). Figure 4.1 illustrates the architecture of Simple-net for a bay with $W = 4$ stacks and $H = 5$ maximum height.

4.1.2 Complex-CNN Architecture

Convolutional layers are most commonly applied to image processing (NIELSEN, 2015). However, similar to the pixels in an image, the input setting of the Container Relocation Problem (a bay configuration) also has a relevant spatial structure, i.e., not only is the position of a container relevant, but so is its relative position to the other containers. Therefore, Complex-CNN leverages the ability of convolutional layers to extract relevant features from the spatial structure of the bay (see Subsection 2.4.2.1).

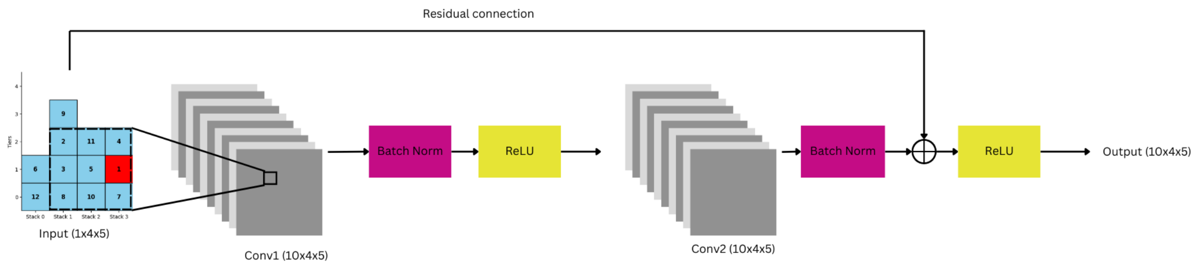
The main component in this architecture is the Residual Block, illustrated in Figure 4.2 with a 4x5 bay as input. In the Residual Block, the first convolutional layer (Conv1) processes the input with a 3x3 kernel and stride length 1 to create 10 feature maps. These feature maps then pass through batch normalization and the ReLU activation function. In the second part, 10 new feature maps are created, once again using a 3x3 kernel with

Figure 4.1: Architecture of Simple-Net for a 4x5 bay



Source: Own representation.

Figure 4.2: Architecture of a Residual Block from Complex-CNN



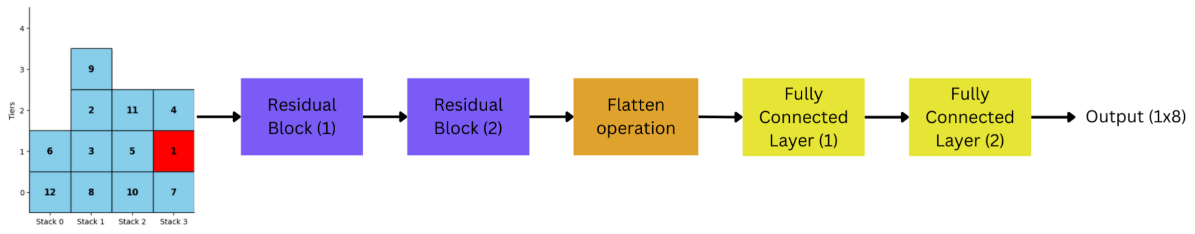
Source: Own representation.

stride length 1. After batch normalization, the input to the Residual Block is added through a residual connection, and ReLU is applied.

As convolutional layers are typically employed in the context of image processing, it is common to use them along with pooling layers (KHAN et al., 2020), which reduce the size of the input and contribute to the network's capacity to generalize features. However, the CRP inputs are much smaller than high-resolution images. Hence, reducing their size would entail sacrificing information to gain (unnecessary) efficiency. Therefore, Complex-CNN does not use pooling layers. In fact, before each convolution, the input is padded with a value of -1 to ensure that the operation always creates feature maps with the same size as the input.

The architecture of Complex-CNN first extracts features from the input with two

Figure 4.3: Architecture of Complex-CNN



Source: Own representation.

consecutive residual blocks. Then, the output from the blocks is flattened into a one-dimensional vector and passed on to the fully-connected layers. In the end, similarly to the output of Simple-Net, the one-dimensional vector of Q-values, one for each possible action, is output (as defined in Subsection 4.1.1). Figure 4.3 illustrates this process for the 4x5 bay from Figure 3.1.

4.2 Training the Agent

Given a bay size $W \times H$ and a number of containers N, the Double Deep Q-Learning agent is trained throughout a series of episodes, following the process described in this section. This process is independent of neural network architecture (i.e., it was applied to both Simple-Net and Complex-CNN) and is summarized as pseudocode in Algorithm 2.

This subsection also introduces many hyperparameters that are essential for training. The way the value of each hyperparameter was chosen and, in many cases, optimized is detailed in Section 4.4.

An episode is initiated by generating a random bay configuration of size $W \times H$ with N containers, numbered according to their priorities. Then, for each step, illegal actions are filtered out (see Subsection 3.1.2), and an action a_t is chosen according to the epsilon-greedy policy. This action is performed, and the transition $(s_t, a_t, r_{t+1}, s_{t+1})$ is saved to the replay buffer.

The constants C_{online}, C_{target}, and C_{eval} determine the intervals with which three key operations are performed:

- C_{online}: After every C_{online} steps taken, the online network parameters θ are updated (see Subsection 4.2.1).

- C_{target}: After every C_{target} episodes, the online network parameters θ^- are updated by copying the online parameters θ (see Subsection 4.2.1).
- C_{eval}: After every C_{eval} episodes, the agent is evaluated (see Section 4.3).

After the episode is finished, the parameter ϵ is decayed by a factor of $\lambda_\epsilon \in [0, 1]$. This way, as the training progresses and the agent learns more about the environment, its strategy shifts from a focus on exploration (with a high ϵ) to a focus on exploitation (with a low ϵ). However, ϵ is never decayed below a minimum threshold ϵ_{min}, to ensure the agent continues to explore new actions, even in later stages of training.

This cycle of randomly initializing an instance, taking actions and storing transitions until it is solved, while periodically updating the online parameters θ and the target parameters θ^-, evaluating the model and decaying ϵ, is repeated for the number of episodes determined by N_{ep}.

Algorithm 2 Training Loop Implementation

```
1: Input: $W, H, N, N_{\text{ep}}, C_{\text{online}}, C_{\text{target}}, C_{\text{eval}}, \epsilon_0, \epsilon_{\text{min}}, \lambda_\epsilon$
2: $\epsilon := \epsilon_0$
3: $n_{\text{ep}} := 1$
4: Initialize all counters $c_{\text{online}} := 1$, $c_{\text{target}} := 1$
5: while $n_{\text{ep}} \leq N_{\text{ep}}$ do
6:     Generate a random bay configuration of size $W \times H$ with $N$ containers
7:     $t := 1$
8:     while episode is not done do
9:         Choose action $a_t$ following an $\epsilon$-greedy policy
10:        Perform $a_t$, register the reward $r_{t+1}$ and the next state $s_{t+1}$
11:        Save the transition $(s_t, a_t, r_{t+1}, s_{t+1})$ to the replay buffer
12:        if $c_{\text{online}}$ is a multiple of $C_{\text{online}}$ then
13:            Update the online network parameters $\theta$ (see Algorithm 4)
14:        end if
15:        $c_{\text{online}} := c_{\text{online}} + 1$
16:        $t := t + 1$
17:    end while
18:    $n_{\text{ep}} := n_{\text{ep}} + 1$
19:    if $n_{\text{ep}}$ is a multiple of $C_{\text{eval}}$ then
20:        Evaluate the agent and save the model
21:    end if
22:    $c_{\text{target}} := c_{\text{target}} + 1$
23:    if $c_{\text{target}}$ is a multiple of $C_{\text{target}}$ then
24:        Update the target network parameters $\theta^- := \theta$
25:    end if
26:    $\epsilon := \max(\epsilon \cdot \lambda_\epsilon, \epsilon_{\text{min}})$
27: end while
```

4.2.1 Updating the Online Parameters

Before updating the online parameters, it must first be checked if the replay buffer contains a minimum of N_{replay} transitions stored. If not, the update is postponed until more transitions are simulated. This avoids the risk of fitting the network to a set of transitions that are too correlated, which could destabilize learning.

Once there are enough transitions in the replay buffer, a minibatch of size m is randomly sampled from it. For each transition $(s_t, a_t, r_{t+1}, s_{t+1})$, s_t and s_{t+1} are used by both the online and the target networks to compute the necessary Q-values. Specifically, $\hat{Q}(s_t; \theta)$ is computed for the current observations s_t, and both $\hat{Q}(s_{t+1}; \theta)$ and $\hat{Q}(s_{t+1}; \theta^-)$ are computed for the next observations s_{t+1}. Here, any $Q(s)$, without a, represents a vector of Q-values for all possible actions at state s, e.g. $\hat{Q}(s; \theta) = [\hat{Q}(s, a_0; \theta), \hat{Q}(s, a_1; \theta), \ldots]$.

Now, the input X to the network is the set of observations s_t from the minibatch. The network's current output is $\hat{y} = \hat{Q}(s_t; \theta)$ for all observations s_t in the minibatch. The expected output y is constructed by taking a copy of $\hat{y}$ and replacing, for each transition, the entry corresponding to a_t with $\tilde{Q}(s_t, a_t)$. $\tilde{Q}(s_t, a_t)$ is computed through bootstrapping, according to Expression (4.1) (see Subsection 2.4.1.2 for more details).

$$\tilde{Q}(s_t, a_t) \equiv r_{t+1} + \gamma \hat{Q}(s_{t+1}, \operatorname{argmax}_a \hat{Q}(s_{t+1}, a; \theta); \theta^-) \tag{4.1}$$

Consequently, the expected output y and the current output $\hat{y}$ are identical, except for the entries corresponding to state-action pairs (s_t, a_t) from the minibatch.

With X, $\hat{y}$ and y computed, an optimization step is performed using the Adam optimizer. The Huber loss function is used to calculate the error between $\hat{y}$ and y, and the online parameters θ are updated via backpropagation.

For clarity, Algorithm 3 exemplifies the process of building the expected output y for one transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in a scenario with $2W$ possible actions. Algorithm 4 shows the entire process for updating the online parameters θ.

Algorithm 3 Example of building the expected output y

```
1: **Input:** $\hat{y} = \hat{Q}(s_t; \theta)$
2: Initialize $y$ as a copy of $\hat{y}$, i.e., $y := [\hat{Q}(s_t, a_0; \theta), \hat{Q}(s_t, a_1; \theta), ..., \hat{Q}(s_t, a_{2W-1}; \theta)]$
3: For a given transition, action $a_0$ was chosen at $s_t$, so $\hat{Q}(s_t, a_0; \theta)$ is replaced by $\tilde{Q}(s_t, a_0)$
   in $y$
4: **return** $y = [\tilde{Q}(s_t, a_0), \hat{Q}(s_t, a_1; \theta), ..., \hat{Q}(s_t, a_{2W-1}; \theta)]$
```

Algorithm 4 Updating the online parameters θ

```
1: **Input:** $N_{\text{replay}}, m, \gamma$
2: **if** Replay buffer contains fewer than $N_{\text{replay}}$ transitions **then**
3:     Do not update $\theta$, more simulations are required
4: **end if**
5: Select a random minibatch of size $m$ from the replay buffer
6: Compute $\hat{y} = \hat{Q}(s_t; \theta)$ for all $s_t$ in the minibatch
7: Compute $\hat{Q}(s_{t+1}; \theta)$ and $\hat{Q}(s_{t+1}; \theta^-)$ for all $s_{t+1}$ in the minibatch
8: **for** every transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in the minibatch **do**
9:     Bootstrap $\tilde{Q}(s_t, a_t) \equiv r_{t+1} + \gamma \hat{Q}(s_{t+1}, \text{argmax}_a \hat{Q}(s_{t+1}, a; \theta); \theta^-)$
10:    Build the expected output $y$ (see Algorithm 3)
11: **end for**
12: Take an optimization step using Adam with all $s_t$ in the minibatch as input to the
    network, $\hat{y} = \hat{Q}(s_t; \theta)$ as current output and $y$ as expected output
13: **return** Updated $\theta$
```

4.3 Evaluation

To evaluate the DDQN agents, the CM dataset (CASERTA; SCHWARZE; VOSS, 2009), the most widely used dataset for evaluating solution methods for the CRP, was employed. For a bay of size $W \times H$, the instances in this dataset contain $N = W \times H'$ containers. Initially, all containers are evenly distributed across the stacks, such that the slots in the top two tiers start empty, as shown in Figure 4.4.

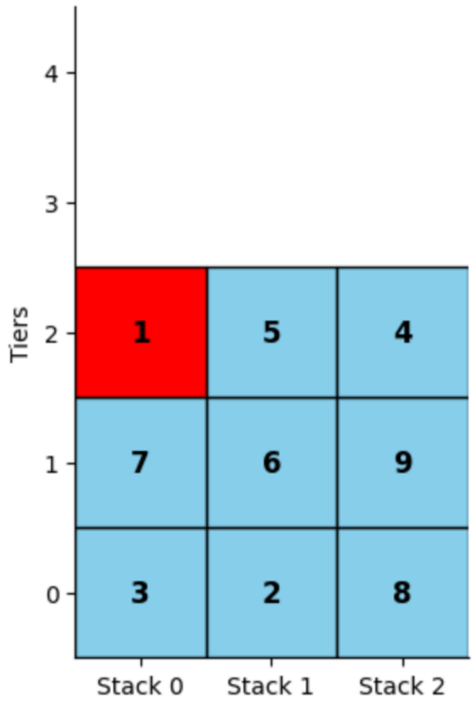
Using Constraint (3.1), one can conclude that these instances are feasible for any bay size where $H \leq 2W + 1$, which is the case for all sizes in the dataset.

The CM dataset contains 21 different bay sizes, each with 40 corresponding instances, for a total of 840 instances. The quality of a solution to an instance is measured by the total number of relocations performed.

During training, the agent is evaluated every C_{eval} episodes, as outlined in Algorithm 2. This works as a validation checkpoint, to assess whether the learning process is actually impacting the agent's ability to solve instances of the CRP. At each checkpoint, the agent solves all 40 instances corresponding to the size $W \times H$, and the average number of relocations performed over all instances is recorded.

Since the agent is trained on randomly generated bay configurations of same size, it is possible that some of the evaluation instances are also used in training. However, each agent is trained on tens of thousands of episodes, and the evaluation instances are only 40 out of millions of possible bay configurations. Therefore, it is reasonable to neglect this possible overlap between training and validation sets.

Figure 4.4: Instance “data3-3-1.dat” from the CM dataset



Source: Own representation of an instance from the CM dataset (CASERTA; SCHWARZE; VOSS, 2009).

4.4 Bayesian Optimization for Hyperparameter Tuning

As discussed in Section 4.2, the training process depends on multiple parameters (summarized in Table 4.2) that, unlike weights and biases in the neural networks, are not automatically optimized. These are named hyperparameters and are often set arbitrarily or based on previous empirical results. In the context of this work, there are several challenges regarding the choice of hyperparameters.

Firstly, to the best of the author's knowledge, this Double Deep Q-Learning approach to CRP has never been tried before, so there is no previous experience from which to draw knowledge on hyperparameters. Secondly, the results from training processes are noisy. Due to the stochastic nature of neural network training and environment exploration, two training processes with identical hyperparameters may produce two different models, which might have significantly different performances. Therefore, testing each hyperparameter configuration just once would not be enough to assess its quality.

Moreover, one must take into account the possibility that different bay sizes might require different hyperparameters, as found by Wei et al. (2021) in a different RL approach.

Table 4.2: Hyperparameters used in training, with their respective search spaces or values

Hyperparameter	Optimized by BO	Search space	Value
Learning rate α	Yes	$[10^{-4}; 0,15]$	-
Discount factor γ	Yes	$[0,9; 1]$	-
Epsilon decay factor λ_ϵ	Yes	$[0,95; 0,999999]$	-
Minimum epsilon ϵ_{min}	Yes	$[0; 0,2]$	-
Interval for online update C_{online}	Yes	$[5; 100]$	-
Interval for target update C_{target}	Yes	$[3; 50]$	-
Initial epsilon ϵ_0	No	-	1
Replay buffer size	No	-	20000
Minimum replay buffer size N_{replay}	No	-	2000
Minibatch size m	No	-	32
Number of episodes N_{ep}	No	-	30000

Source: Own representation of values and search spaces used.

Considering all of this, plus the high number of hyperparameters involved in the DDQN approach, and the high computational cost of training the agents, the best way to tune the hyperparameters is to utilize a robust and efficient optimization technique to acquire the best results with fewer attempts.

Bayesian Optimization (BO), detailed in Subsection 2.4.3, was chosen as the technique for hyperparameter tuning. In this approach, the quality of an agent, as measured in the evaluation explained in Section 4.3, is considered to be the noisy objective function, with the relevant hyperparameters to be optimized.

Even with a technique specialized in the optimization of computationally expensive functions, the number of hyperparameters was still too large, and the optimization process would have required too much time. Hence, hyperparameters deemed most relevant were optimized. Regarding the values set as constant:

- The initial epsilon ϵ_0 controls the probability of the agent choosing a random action (instead of the one with the highest Q-values) in the early stages of training (see Subsection 2.4.1.3). As the Q-value estimates are still quite poor in these stages, maximizing Q-values is meaningless. Therefore, starting with $\epsilon_0 = 1$ is most reasonable.

- The minibatch size of 32 is common in neural network training (see for example Mnih et al. (2015)) and worked well in initial tests for this current work.

- The replay buffer's maximum and minimum size, as well as the number of episodes in the training process were set by the author based on initial tests to train models of decent quality in a reasonable timeframe considering the machine available (details on the machine specifications provided in Chapter 5).

Table 4.2 shows all hyperparameters and their respective values or search ranges. From here on, the word “hyperparameter” is used to refer only to the ones that are optimized via BO.

The BO implementation uses the Python library Ax (OLSON et al., 2025). For a given bay size $W \times H$, an experiment is set up, which corresponds to the BO process. The objective function to be minimized takes the hyperparameters as input and outputs the average number of relocations over the 40 instances of the CM dataset corresponding to size $W \times H$. The objective function is considered to have Gaussian noise with zero mean and unknown standard deviation.

A trial is an observation of the objective function for a certain hyperparameter configuration. The first 12 trials test quasi-random hyperparameter configurations to initially explore the hyperparameter space (using the Sobol model). Then, 8 more trials are run, fitting a surrogate model to the observed data using Gaussian Process Regression (using the BoTorch model) and the Noisy Expected Improvement as the acquisition function to decide the hyperparameters for each next trial (OLSON et al., 2025).

Due to the noise in the objective function, the Bayesian Optimization does not simply return the best trial, but rather an estimate of the best parameters and the optimal objective, based on all the trials performed and the estimated noise.

5 COMPUTATIONAL EXPERIMENTS

In this chapter, the results obtained by the DDQN approach, using both Simple-Net and Complex-CNN, to the Container Relocation Problem are detailed and analyzed. The methods described in Chapter 4 were implemented using Python and multiple of its libraries, mainly Numpy, PyTorch, and Ax. The implementation code has been made publicly available (BRAVO, 2025). All training processes and tests were carried out on a Linux machine equipped with an Intel(R) Core(TM) i7-11700K processor (8 cores, 16 threads, 3.6 GHz base), 134GB of RAM and an NVIDIA GeForce GTX 1650 (4GB VRAM), CUDA 12.7. Access to this machine was kindly provided by the chair for Management Sciences and Operations Research at the Technical University of Darmstadt.

For clarity, as it was done throughout this thesis, bay sizes are referenced by their size $W \times H$. H', explained in Section 4.3, is also provided in the tables for completion and for facilitating the comparison with other works that use it as reference instead of H.

In this chapter, Section 5.1 compares the quality of solutions achieved to other benchmark studies. Section 5.2 examines Bayesian Optimization process and the best parameters found. Lastly, Section 5.3 considers training the agents for longer periods.

5.1 Solution Quality

The analysis of overall solution quality is based on the average number of relocations achieved over 40 instances for a variety of bay sizes in the CM dataset (CASERTA; SCHWARZE; VOSS, 2009), as detailed in Section 4.3. In Table 5.1, the Simple-Net and the Complex-CNN agents are compared to a state-the-art method using Ant Colony Optimization (ACO) by Jovanovic, Tuba, and Voss (2019) and to the optimal solutions obtained by Expósito-Izquierdo, Melián-Batista, and Moreno-Vega (2014) using an A^* algorithm.

Table 5.1: Average number of relocations over 40 instances of the CM dataset for multiple bay sizes

Bay size				Average number of relocations			
W	H	H'	N	Simple-Net	Complex-CNN	ACO	Optimal
4	5	3	12	6.72 (6.73)	6.75 (6.82)	**6.18**	**6.18**
5	5	3	15	8.10 (8.11)	7.88 (7.90)	**7.02**	**7.02**
6	5	3	18	10.18 (10.40)	9.45 (9.49)	**8.40**	**8.40**
7	5	3	21	11.35 (11.47)	10.42 (10.50)	**9.28**	**9.28**
8	5	3	24	14.02 (14.11)	12.15 (12.26)	**10.65**	**10.65**
4	6	4	16	11.68 (11.77)	12.58 (12.62)	**10.20**	**10.20**
5	6	4	20	16.02 (16.03)	15.30 (15.55)	**12.95**	**12.95**
6	6	4	24	18.95 (19.01)	17.02 (17.06)	**14.02**	**14.02**
7	6	4	28	23.52 (23.79)	19.62 (19.94)	**16.12**	**16.12**
4	7	5	20	19.35 (19.52)	19.65 (19.72)	**15.42**	**15.42**
5	7	5	25	25.32 (25.44)	24.85 (24.87)	18.95	**18.85**

Source: Own results, Caserta (2017), Jovanovic, Tuba, and Voss (2019) and Expósito-Izquierdo, Melián-Batista, and Moreno-Vega (2014).

5.1.1 Simple-Net and Complex-CNN

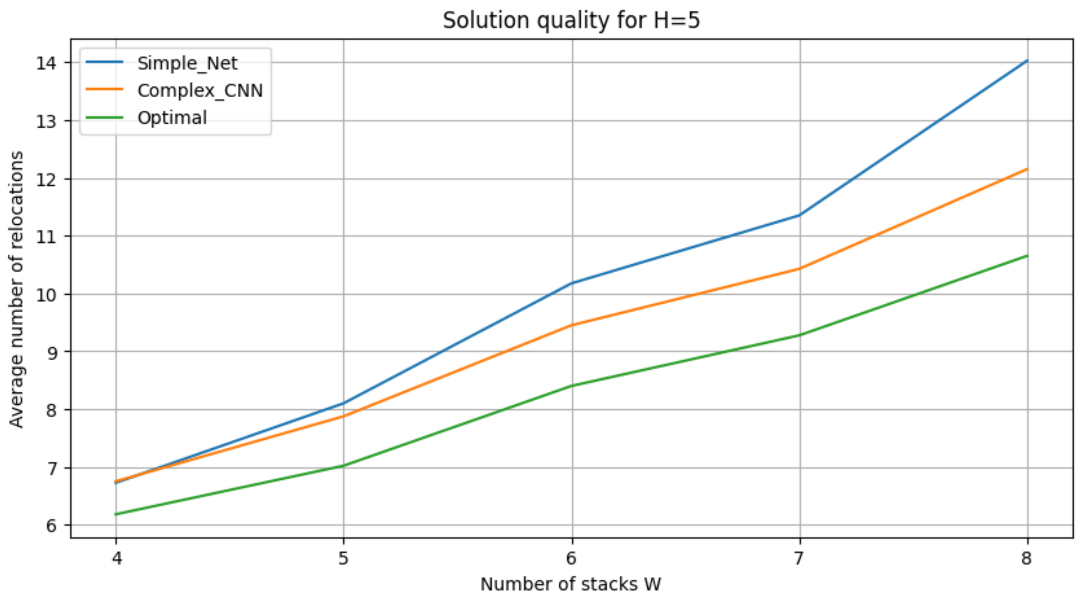
For both Simple-Net and Complex-CNN, the difference between actual and estimated best model, i.e., the difference between the values outside and within the parentheses, is quite small across all bay sizes. This suggests that, despite the stochastic nature of RL training, for this application, the noise associated with a given hyperparameter configuration is not very significant, and perhaps could even be neglected in future approaches, to simplify the tuning of hyperparameters.

It is quite clear that Complex-CNN consistently outperforms Simple-Net across the vast majority of bay sizes (all, except for 4x5, 4x6 and 4x7). One could, therefore, hypothesize that the extra features of Complex-CNN bring a significant improvement when there are at least 5 stacks.

It is also important to note that the improvement provided by Complex-CNN over Simple-Net increases as the bay size gets bigger. Figures 5.1 and 5.2 clearly show this trend for bay heights $H = 5$ and $H = 6$, as the blue and orange curves diverge with an increasing number of stacks W.

As for time required, both types of agent took around thirty minutes to be trained on the largest sizes for 30000 episodes, which is not a concern, as training is only performed once. Furthermore, both types of agent solve each evaluation instance in less than one

Figure 5.1: Solution quality comparison of Simple-Net, Complex-CNN and the optimum for an increasing number of stacks W and fixed bay height $H = 5$



Source: Own representation.

second. The increased complexity of Complex-CNN did not have a significant impact on training or on solution times. Therefore, both Simple-Net and Complex-CNN are appropriate for practical applications which require fast solutions.

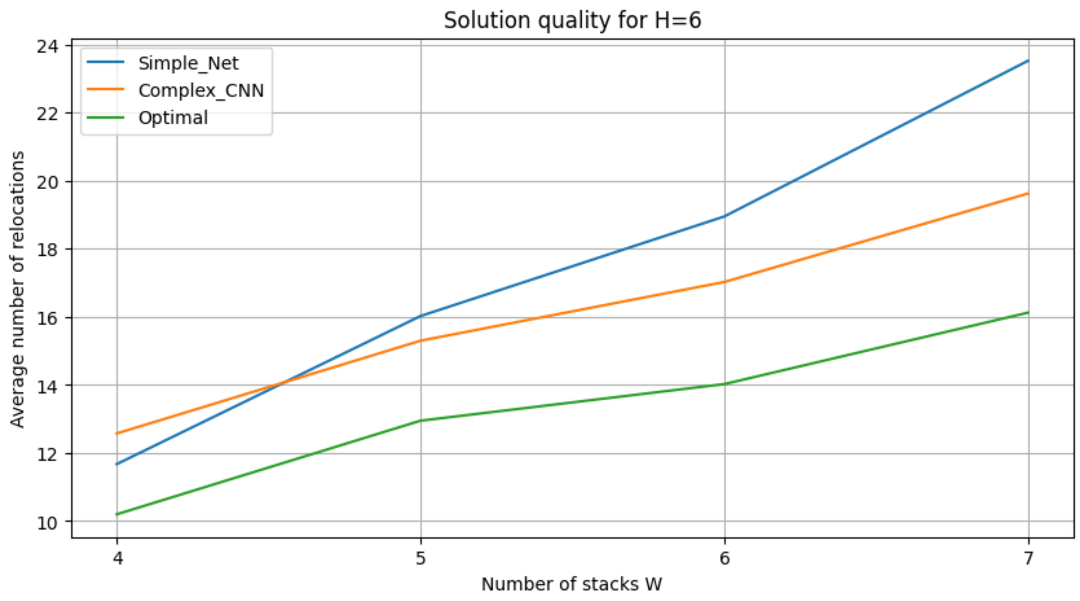
5.1.2 Comparison with other Methods

The ACO method achieves optimal results in nearly all small and medium bay sizes shown in Table 5.1, also in negligible time (JOVANOVIC; TUBA; VOSS, 2019). The Double Deep Q-Learning approaches proposed in this work lag significantly, but not too far, behind. Complex-CNN in particular achieves good results for most bay sizes and, most importantly, maintains its gap to the optimal solutions even as bay sizes increase, as shown by Figures 5.1 and 5.2.

5.2 Analysis of Bayesian Optimization

Table 5.2 shows the Bayesian Optimization process for bay size 7×6 with $N = 28$ containers. During the training process of each trial, the agent was evaluated every 2000 episodes. Column AR gives the lowest average number of relocations achieved for a trial, whereas column EBM shows in which episode that lowest average was achieved.

Figure 5.2: Solution quality comparison of SimpleNet, Complex-CNN and the optimum for an increasing number of stacks W and fixed bay height $H = 6$



Source: Own representation.

This means that trial 12, for example, achieved its best score after 14000 episodes of training, but failed to improve that mark in the following 16 thousand episodes of training. This can be seen quite clearly in Figure 5.3.

On the other hand, for some trials, the best score is achieved close to the end, as is the case for trial 16. In Figure 5.4, there is a clear downward trend in the average number of relocations throughout the entire training process. Thus, it is reasonable to hypothesize that, with longer training, better results could be achieved.

5.2.1 Hyperparameter Tuning

The optimal hyperparameters found by Bayesian Optimization for each bay size are plot as histograms in Figures 5.5 and 5.6. The limits of the X-axis in each histogram are defined by the respective hyperparameter's search range.

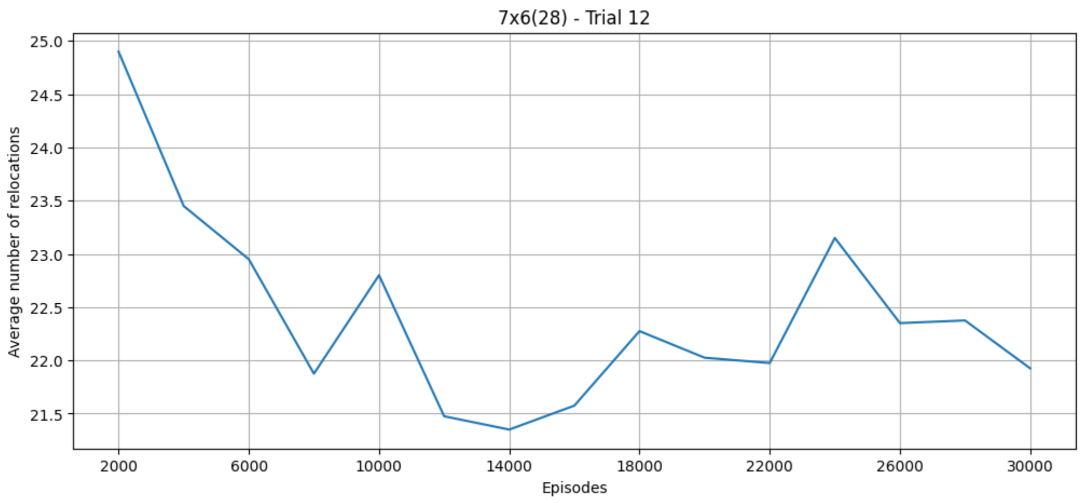
It is noted that, similarly to what was found by (WEI et al., 2021), the best hyper-parameters vary considerably for different bay sizes. The only exception seems to be the online update interval C_{online}.

Ideally, one agent should be trained to solve multiple different bay sizes, so that it can be applied to many practical scenarios. Therefore, the indication that distinct bay sizes may require significantly different hyperparameters would pose a serious challenge to the

Trial	Generation	AR	EBM	α	γ	λ_ϵ	ϵ_{min}	C_{online}	C_{target}
0	Sobol	30.150	30000	2.23e-03	0.958	0.997	0.127	42	17
1	Sobol	34.650	16000	4.02e-02	0.904	0.973	0.090	60	28
2	Sobol	33.300	14000	6.02e-03	0.979	0.976	0.170	90	41
3	Sobol	20.525	28000	4.20e-04	0.932	0.950	0.008	12	5
4	Sobol	25.050	20000	2.24e-04	0.995	0.965	0.028	19	23
5	Sobol	33.875	8000	9.97e-03	0.942	0.989	0.191	83	35
6	Sobol	33.800	26000	1.48e-01	0.967	0.961	0.072	67	47
7	Sobol	21.775	26000	6.73e-04	0.919	0.987	0.110	35	10
8	Sobol	28.725	28000	1.35e-03	0.983	0.957	0.181	53	7
9	Sobol	34.150	14000	7.23e-02	0.927	0.982	0.044	49	42
10	Sobol	34.250	14000	1.97e-02	0.953	0.966	0.113	5	30
11	Sobol	26.800	30000	1.17e-04	0.908	0.991	0.051	97	18
12	BoTorch	21.350	14000	4.45e-04	0.900	0.958	0.000	5	3
13	BoTorch	20.325	22000	4.59e-04	0.917	0.950	0.003	5	48
14	BoTorch	20.225	30000	3.79e-04	0.919	0.950	0.200	5	50
15	BoTorch	23.400	26000	4.27e-04	0.920	0.950	0.089	100	50
16	BoTorch	19.625	28000	3.53e-04	0.926	1.000	0.022	5	50
17	BoTorch	20.550	12000	3.68e-04	0.927	1.000	0.200	5	50
18	BoTorch	19.975	24000	2.97e-04	0.925	0.950	0.000	5	50
19	BoTorch	20.625	28000	2.99e-04	0.926	1.000	0.000	5	3

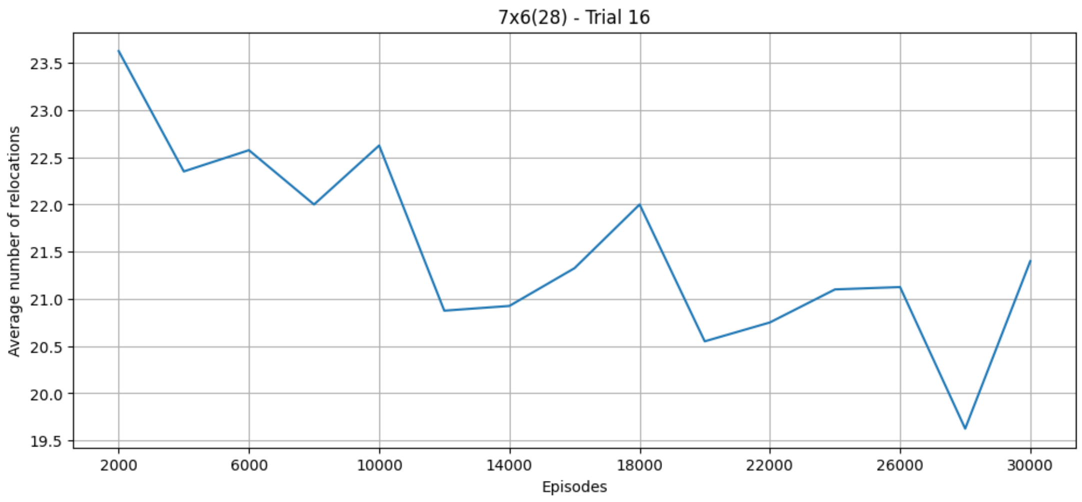
Table 5.2: Bayesian Optimization for bay size 7×6 with $N = 28$

Figure 5.3: Average number of relocations in the CM dataset across training process for bay size 7×6, $N = 28$, BO trial 12



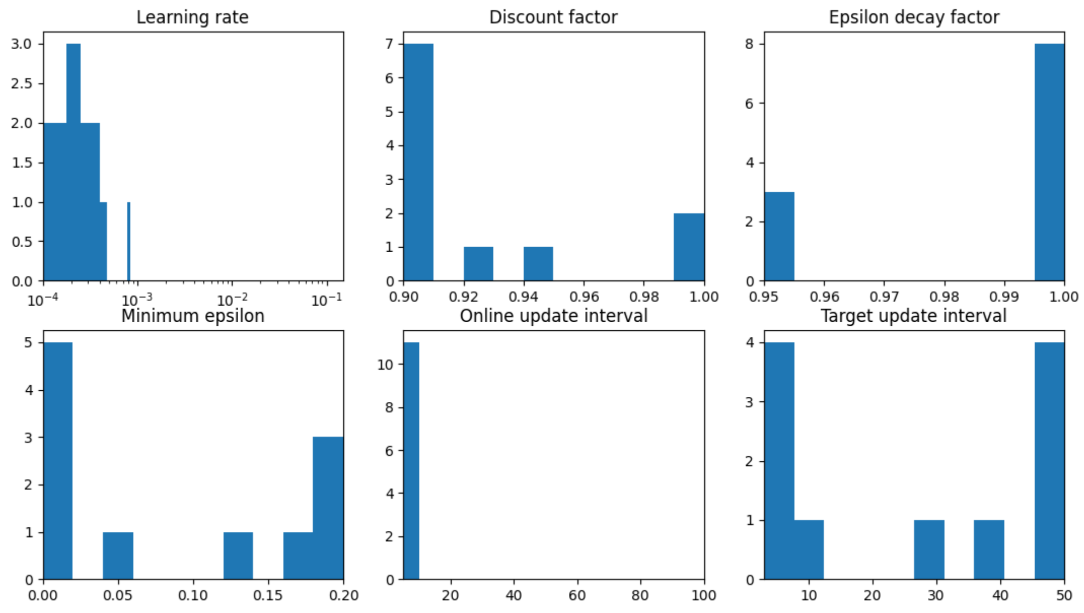
Source: Own representation.

Figure 5.4: Average number of relocations in the CM dataset across training process for bay size 7×6, $N = 28$, BO trial 16



Source: Own representation.

Figure 5.5: Distributions of the best hyperparameters found across bay sizes for SimpleNet



Source: Own representation.

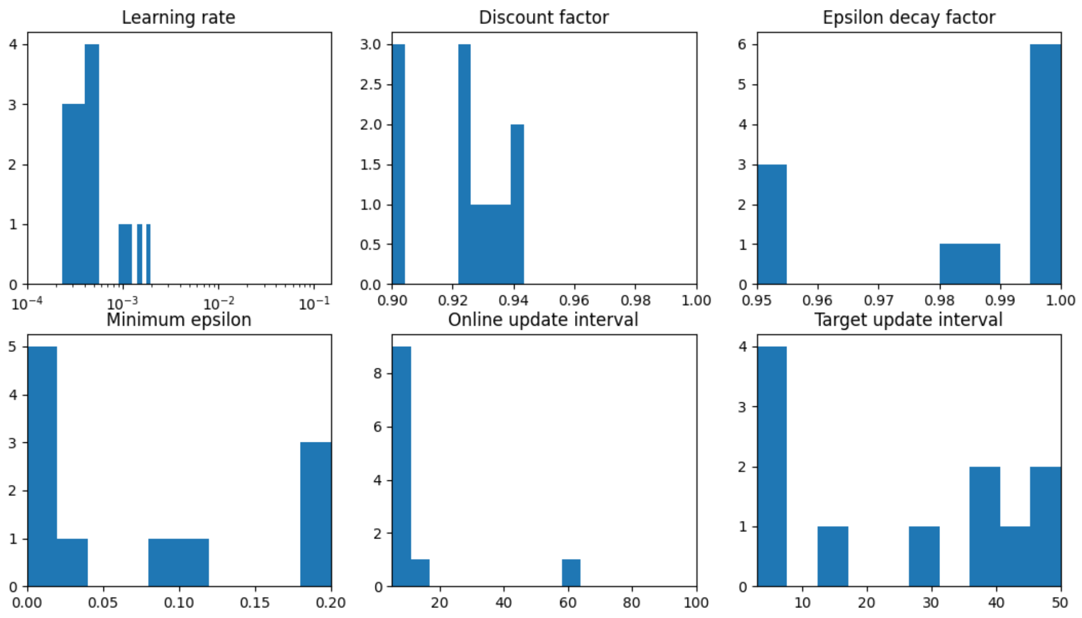
effort towards a flexible agent. However, the evidence presented here does not rule the possibility of finding a hyperparameter configuration that gets good (though not optimal) results for multiple bay sizes.

5.3 Longer Training Periods

In this section, the hypothesis raised in Section 5.2 about possibly obtaining better results with longer training periods is briefly investigated. Two trials that presented a downward trend in the average number of relocations throughout the training process were retrained with the same hyperparameter configuration, but for 50000 episodes, instead of 30000. The trials were number 16 from bay size 7×6 and number 18 from bay size 8×5.

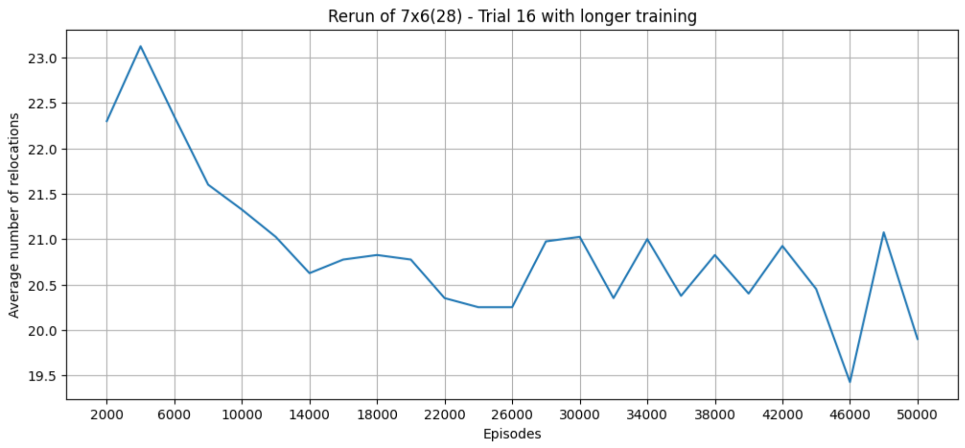
However, as shown by Figures 5.7 and 5.8, learning unfortunately stagnates after around 20000 episodes in both cases. It is possible that learning plateaus in this area before improving once again later, something fairly common in neural network training, but further tests would be necessary to assess this.

Figure 5.6: Distributions of the best hyperparameters found across bay sizes for Complex-CNN



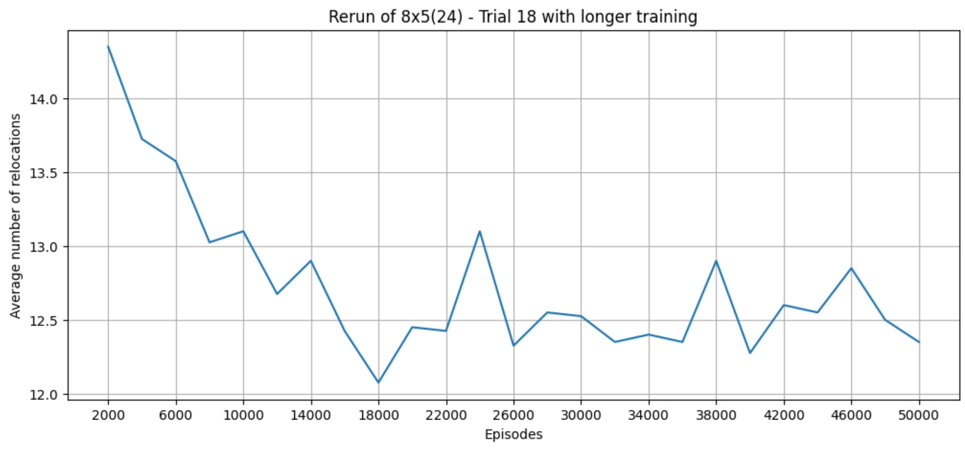
Source: Own representation.

Figure 5.7: Average number of relocations in the CM dataset across a longer training process for bay size 7×6, $N = 28$, rerun of BO trial 16



Source: Own representation.

Figure 5.8: Average number of relocations in the CM dataset across a longer training process for bay size 8×5, $N = 24$, rerun of BO trial 18



Source: Own representation.

6 CONCLUSIONS AND FUTURE RESEARCH

In light of the importance of container handling for terminal logistics and international trade, as well as the optimization challenges imposed by the Container Relocation Problem, this work investigated the applicability of a new Double Deep Q-Learning approach to this problem.

The literature surrounding the CRP was reviewed to understand the details and particularities of the problem, as well as the work that has been done by researchers, including the approaches attempted and the results obtained. Then, a gap was noticed regarding the application of Deep Q-Learning methods, which have found tremendous success in a plethora of applications in recent years. In particular, Double Deep Q-Learning, a method proposed to correct issues with DQN, was chosen as a new solution approach to be investigated.

The theory behind DDQN was studied, alongside other methods that could potentially complement it. DDQN was then implemented with two variations: (i) Simple-Net, a simpler fully connected network; and (ii) Complex-CNN, a more complex network, with convolutional layers, batch normalization, and residual connections, whose goal was to extract better features from the inputs. Furthermore, Bayesian Optimization was used to handle the task of tuning the multiple hyperparameters necessary for the training of the DDQN agents within a reasonable number of trials.

The agents trained are capable of solving instances of the CRP in negligible time, which makes them applicable in practical scenarios. The solutions obtained were good, but not as good as state-of-the-art methods in the literature. Most importantly, this work lays the foundation for future approaches to apply the Double Deep Q-Learning framework to the Container Relocation Problem. This work's findings pave the path for future research that could significantly improve results.

The improvements achieved by increasing neural network complexity, for example, indicate that different architectures and network components can significantly impact performance, and should, therefore, be investigated further. Moreover, Bayesian Optimization has proved itself a valuable tool to systematically handle the tuning of the many hyperparameters involved in a DDQN approach.

Finally, given the history of Deep Q-Learning methods of thriving in extremely large environments (MNIH et al., 2015), the application of such methods to bigger and bigger sizes could potentially bring significantly better results.

REFERENCES

AMARI, Shun-ichi. Backpropagation and stochastic gradient descent method. **Neurocomputing**, Elsevier, v. 5, n. 4-5, p. 185–196, 1993.

AMBROSINO, Daniela; SCIOMACHEN, Anna; TANFANI, Elena. Stowing a containership: the master bay plan problem. **Transportation Research Part A: Policy and Practice**, Elsevier, v. 38, n. 2, p. 81–99, 2004.

BIERWIRTH, Christian; MEISEL, Frank. A survey of berth allocation and quay crane scheduling problems in container terminals. **European Journal of Operational Research**, Elsevier, v. 202, n. 3, p. 615–627, 2010.

BJORCK, Nils et al. Understanding batch normalization. **Advances in Neural Information Processing Systems**, v. 31, 2018.

BRAVO, André. **Andre-F-B/container-relocation-ddqn-thesis: Final Submission**. [S.l.]: Zenodo, Nov. 2025. DOI: `10.5281/zenodo.17705299`. Available from: ¡`https://doi.org/10.5281/zenodo.17705299`¿.

BROCHU, Eric; CORA, Vlad M; DE FREITAS, Nando. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. **arXiv preprint arXiv:1012.2599**, 2010.

CABALLINI, Claudia et al. A combined data mining–optimization approach to manage trucks operations in container terminals with the use of a TAS: Application to an Italian and a Mexican port. **Transportation Research Part E: Logistics and Transportation Review**, Elsevier, v. 142, p. 102054, 2020.

CARLO, Héctor J; VIS, Iris FA; ROODBERGEN, Kees Jan. Storage yard operations in container terminals: Literature overview, trends, and research directions. **European Journal of Operational Research**, Elsevier, v. 235, n. 2, p. 412–430, 2014.

CASERTA, Marco. **BRP**. [S.l.: s.n.], 2017. Accessed: 2025-02-06. Internet Archive: `https://web.archive.org/web/20240419112227/https://github.com/marcocaserta/BRP`. Available from: ¡`https://github.com/marcocaserta/BRP`¿.

CASERTA, Marco; SCHWARZE, Silvia; VOSS, Stefan. A mathematical formulation and complexity considerations for the blocks relocation problem. **European Journal of Operational Research**, Elsevier, v. 219, n. 1, p. 96–104, 2012. DOI: `https://doi.org/10.1016/j.ejor.2011.12.039`.

________. A new binary description of the blocks relocation problem and benefits in a look ahead heuristic. In: SPRINGER. EVOLUTIONARY Computation in Combinatorial Optimization: 9th European Conference, EvoCOP 2009, Tübingen, Germany, April 15-17, 2009. Proceedings 9. [S.l.: s.n.], 2009. P. 37–48.

________. Container rehandling at maritime container terminals: A literature update. **Handbook of Terminal Planning**, Springer, p. 343–382, 2020.

AL-DHAHERI, Noura; DIABAT, Ali. The quay crane scheduling problem. **Journal of Manufacturing Systems**, Elsevier, v. 36, p. 87–94, 2015.

EXPÓSITO-IZQUIERDO, Christopher; MELIÁN-BATISTA, Belén; MORENO-VEGA, J Marcos. A domain-specific knowledge-based heuristic for the blocks relocation problem. **Advanced Engineering Informatics**, Elsevier, v. 28, n. 4, p. 327–343, 2014.

________. An exact approach for the blocks relocation problem. **Expert Systems with Applications**, Elsevier, v. 42, n. 17-18, p. 6408–6422, 2015.

FRAZIER, Peter I. A tutorial on Bayesian optimization. **arXiv preprint arXiv:1807.02811**, 2018.

GALLE, Virgile; BARNHART, Cynthia; JAILLET, Patrick. A new binary formulation of the restricted container relocation problem based on a binary encoding of configurations. **European Journal of Operational Research**, Elsevier, v. 267, n. 2, p. 467–477, 2018.

GALLE, Virgile; MANSHADI, Vahideh H, et al. The stochastic container relocation problem. **Transportation Science**, INFORMS, v. 52, n. 5, p. 1035–1058, 2018.

GOODFELLOW, Ian; BENGIO, Yoshua; COURVILLE, Aaron. **Deep Learning**. [S.l.]: MIT Press, 2016. `http://www.deeplearningbook.org`.

GÜVEN, Ceyhun; TÜRSEL ELIIYI, Deniz. Modelling and optimisation of online container stacking with operational constraints. **Maritime Policy & Management**, Taylor & Francis, v. 46, n. 2, p. 201–216, 2019.

HE, Kaiming et al. Deep residual learning for image recognition. In: PROCEEDINGS of the IEEE Conference on Computer Vision and Pattern Recognition. [S.l.: s.n.], 2016. P. 770–778.

HORNIK, Kurt; STINCHCOMBE, Maxwell; WHITE, Halbert. Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks. **Neural Networks**, Elsevier, v. 3, n. 5, p. 551–560, 1990.

HSU, Hsien-Pin et al. Modeling and solving the three seaside operational problems using an object-oriented and timed predicate/transition net. **Applied Sciences**, MDPI, v. 7, n. 3, p. 218, 2017.

HUBER, Peter J. Robust estimation of a location parameter. In: BREAKTHROUGHS in statistics: Methodology and distribution. [S.l.]: Springer, 1964. P. 492–518.

HUSSEIN, Mazen I; PETERING, Matthew EH. Global retrieval heuristic and genetic algorithm in block relocation problem. In: INSTITUTE OF INDUSTRIAL and SYSTEMS ENGINEERS (IISE). IISE Annual Conference. Proceedings. [S.l.: s.n.], 2012. P. 1.

IOFFE, Sergey; SZEGEDY, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In: PMLR. INTERNATIONAL Conference on Machine Learning. [S.l.: s.n.], 2015. P. 448–456.

JOVANOVIC, Raka; TUBA, Milan; VOSS, Stefan. An efficient ant colony optimization algorithm for the blocks relocation problem. **European Journal of Operational Research**, Elsevier, v. 274, n. 1, p. 78–90, 2019.

JOVANOVIC, Raka; VOSS, Stefan. A chain heuristic for the blocks relocation problem. **Computers & Industrial Engineering**, Elsevier, v. 75, p. 79–86, 2014.

KHAN, Asifullah et al. A survey of the recent architectures of deep convolutional neural networks. **Artificial Intelligence Review**, Springer, v. 53, p. 5455–5516, 2020.

KIZILAY, Damla; ELIIYI, Deniz Türsel; VAN HENTENRYCK, Pascal. Constraint and mathematical programming models for integrated port container terminal operations. In: SPRINGER. INTEGRATION of Constraint Programming, Artificial Intelligence, and Operations Research: 15th International Conference, CPAIOR 2018, Delft, The Netherlands, June 26–29, 2018, Proceedings 15. [S.l.: s.n.], 2018. P. 344–360.

LERSTEAU, Charly; SHEN, Weiming. A survey of optimization methods for block relocation and premarshalling problems. **Computers & Industrial Engineering**, Elsevier, v. 172, p. 108529, 2022.

LI, Zewen et al. A survey of convolutional neural networks: analysis, applications, and prospects. **IEEE Transactions on Neural Networks and Learning Systems**, IEEE, v. 33, n. 12, p. 6999–7019, 2021.

LIU, Liqun et al. A Q-learning-based algorithm for the block relocation problem. **Journal of Heuristics**, Springer, v. 31, n. 1, p. 14, 2025.

MEISEL, Frank; BIERWIRTH, Christian. A unified approach for the evaluation of quay crane scheduling models and algorithms. **Computers & Operations Research**, Elsevier, v. 38, n. 3, p. 683–693, 2011.

MNIH, Volodymyr et al. Human-level control through deep reinforcement learning. **Nature**, Nature Publishing Group UK London, v. 518, n. 7540, p. 529–533, 2015.

MOSZYK, Karol; DEJA, Mariusz; DOBRZYNSKI, Michal. Automation of the road gate operations process at the container terminal—a case study of dct gdańsk sa. **Sustainability**, MDPI, v. 13, n. 11, p. 6291, 2021.

NIELSEN, A. **Neural networks and deep learning**. [S.l.]: Determination Press, 2015.

OLSON, Miles et al. Ax: a platform for adaptive experimentation. In: AUTOML 2025 ABCD Track. [S.l.: s.n.], 2025.

ROSENBLATT, Frank et al. **Principles of neurodynamics: Perceptrons and the theory of brain mechanisms**. [S.l.]: Spartan books Washington, DC, 1962. v. 55.

SANTURKAR, Shibani et al. How does batch normalization help optimization? **Advances in neural information processing systems**, v. 31, 2018.

SILVA FIRMINO, Andresson da; ABREU SILVA, Ricardo Martins de; TIMES, Valeria Cesario. An exact approach for the container retrieval problem to reduce crane's trajectory. In: IEEE. 2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC). [S.l.: s.n.], 2016. P. 933–938.

SUTTON, Richard S. Reinforcement learning: An introduction. **A Bradford Book**, 2018.

TANG, Yin et al. Regulating the imbalance for the container relocation problem: A deep reinforcement learning approach. **Computers & Industrial Engineering**, Elsevier, v. 191, p. 110111, 2024.

TWILLER, Jaike van et al. Literature survey on the container stowage planning problem. **European Journal of Operational Research**, Elsevier, v. 317, n. 3, p. 841–857, 2024.

VAN HASSELT, Hado; GUEZ, Arthur; SILVER, David. Deep reinforcement learning with double q-learning. In: 1. PROCEEDINGS of the AAAI Conference on Artificial Intelligence. [S.l.: s.n.], 2016. v. 30.

WANKE, Peter F; BARBASTEFANO, Rafael Garcia; HIJJAR, Maria Fernanda. Determinants of efficiency at major Brazilian port terminals. **Transport Reviews**, Taylor & Francis, v. 31, n. 5, p. 653–677, 2011.

WATKINS, Christopher JCH; DAYAN, Peter. Q-learning. **Machine learning**, Springer, v. 8, p. 279–292, 1992.

WATKINS, Christopher John Cornish Hellaby. Learning from delayed rewards. King's College, Cambridge United Kingdom, 1989.

WEERASINGHE, Buddhi A; PERERA, H Niles; BAI, Xiwen. Optimizing container terminal operations: a systematic review of operations research applications. **Maritime Economics & Logistics**, Springer, v. 26, n. 2, p. 307–341, 2024.

WEI, Lei et al. Optimization of container relocation problem via reinforcement learning. **Logistics Journal: Proceedings**, v. 2021, n. 17, 2021. DOI: `10.2195/lj_Proc_wei_en_202112_02`.

WIESE, Jörg; SUHL, Leena; KLIEWER, Natalia. Planning container terminal layouts considering equipment types and storage block design. In: HANDBOOK of Terminal Planning. [S.l.]: Springer, 2011. P. 219–245.

WORLD BANK. **Sustainable Development in Shipping and Ports**. [S.l.: s.n.], 2023. Accessed: Feb. 12, 2025. Internet Archive: `https://web.archive.org/web/20240221185659/https://www.worldbank.org/en/topic/transport/brief/sustainable-development-in-shipping-and-ports`. Available from: ¡`https://www.worldbank.org/en/topic/transport/brief/sustainable-development-in-shipping-and-ports`¿.

________. **The Container Port Performance Index 2023: A Comparable Assessment of Performance based on Vessel Time in Port**. [S.l.: s.n.], 2024. Accessed: February 12, 2025. Available from: ¡`https://hdl.handle.net/10986/41707`¿.

WU, Kun-Chih; TING, Ching-Jung, et al. A beam search algorithm for minimizing reshuffle operations at container yards. In: PROCEEDINGS of the International Conference on Logistics and Maritime Systems. [S.l.: s.n.], 2010. P. 15–17.

YANG, Jee Hyun; KIM, Kap Hwan. A grouped storage method for minimizing relocations in block stacking systems. **Journal of Intelligent Manufacturing**, Springer, v. 17, p. 453–463, 2006.

ZEHENDNER, Elisabeth; FEILLET, Dominique. Benefits of a truck appointment system on the service quality of inland transport modes at a multimodal container terminal. **European Journal of Operational Research**, Elsevier, v. 235, n. 2, p. 461–469, 2014.

ZHANG, Canrong et al. Machine learning-driven algorithms for the container relocation problem. **Transportation Research Part B: Methodological**, Elsevier, v. 139, p. 102–131, 2020.