Master's Thesis

Solving the Dynamic Container Relocation Problem to Minimize Transportation Costs using Approximate Dynamic Programming

R.R. Bosch

Sep 2022

Supervisors University of Twente:
Martijn Mes
Eduardo Lalla

Supervisors Cofano:
Leon de Vries

UNIVERSITY OF TWENTE.

COFANO

Address:
Drienerlolaan 5
7522NB
Enschede

Address:
Capitool 9
7521PL
Enschede

Management Summary

Problem definition

This research on container logistics is done for Cofano, a software-as-a-service company for many logistics companies. One of their products is a Terminal Operating System, which among other things provides advice on where to stack containers within a terminal. Container terminals are busy interfaces between different modes of transport for containers, where they are stored in stacks until they are retrieved. Transport companies want to be able to drop off and retrieve their containers as quickly as possible, so efficient handling of containers within a terminal is important. The decision where to locate a terminal is an important problem, as the decision has long-term effects on the layout of the terminal. This is because retrieving a container requires the relocation of all containers above it, called a reshuffle. Each reshuffle introduces inefficiency, so it is important to anticipate on which stack you put a container. Additionally, exact information about the arrival and departure times of containers is not known in advance, and each container has many possible destinations, making it infeasible to evaluate each option in-depth.

Methods

To solve this problem, we formulate the container terminal and schedule of arriving and departing containers as a Markov Decision Problem. Each state represents the layout of the terminal, and in each stage a batch of containers arrives or departs. The actions that are required are the allocation of each arriving container to the terminal, and the reshuffling of all containers that block the departing container from doing so.

To solve the Markov Decision Problem, we use Approximate Dynamic Programming. This method is good at optimizing complex stochastic Markov Decision Problems by offering a solution to the “curses of dimensionality”, which makes solving these problems impossible due to the number of states, choices and outcomes that are possible in these kinds of problems. It achieves this through iterative learning and separation of the consequences of actions in a particular state from randomness. The algorithm attempts to learn the optimal decision at each stage by going through the problem multiple times and approximating the future costs at each stage using a value function approximation. We use a Basis Feature Function, which uses characteristics of the terminal layout as input, called features. These features give an indication of the goodness of a particular layout at a specific time, and these values are combined in a weighted sum to arrive at the expected future costs. These weights are learned iteratively by updating the weights so that the predicted costs approach the costs realized in each iteration. In addition, this research tests a neural network as an alternative to the linear approximation used in the basis feature function.

In this particular problem we also have to deal with finding the optimal decision in a single stage, as the available decisions in any given state are too many to evaluate completely. We solve this problem using two methods: a partial search tree and a mixed integer program. The partial search tree evaluates only the x most promising moves, which are determined using the value function approximation and direct costs. The mixed integer program turns the single-stage optimization problem into a set of decision variable and linear constraints, which allow for powerful algorithms to solve this problem efficiently.

Results

The proposed solutions are tested on a set of 13 generated problems. These are terminals with characteristics ranging from 10 to 40 stacks, 40% to 80% occupation, and different length-of-stay for containers and batch sizes. We tested multiple sets of features, different weight-updating algorithms for the basis feature function, choice policies and pre-training using heuristics. All results are benchmarked against two heuristics from literature: the min-max heuristic and the reshuffle-index heuristic.

We find fast convergence towards the optimal performance of the algorithm (<200 iterations). For all of the problem instances we compare the performance of the algorithm to the best performing benchmark heuristics and find that the proposed solution results in a reduction of costs of on average 6.8%, with a standard deviation of 4.8%.

When testing the Neural Network as an alternative to the linear approximation of a basis feature function we find that the Neural Network requires a significantly larger training time (5000 iterations compared to 200), and performs on par or slightly better, resulting in 2% less costs. This is not a robust improvement however, as the Neural Network performance fluctuates more over training iterations.

In testing single-stage solution methods, the Linear Program outperformed the partial search tree by on average …, but the search tree outperformed the algorithm used in (Boschma, 2020) by 3.2%. The Linear Program was prohibitively slow to train with however, so some solution for this would need to be found before it can be used. In the meantime, the partial search tree already provides an improvement over existing methods.

Recommendations

To conclude, the developed solution resulted in an improvement of on average 6.8% reduction in costs compared to benchmark heuristics and converges quickly for an Approximate Dynamic Programming algorithm. However, while this model is able to solve the problem it was tested on better than any tested heuristic, it is not applicable to realistic problems as is. The problems on which it was tested are too small to be realistic terminals, as the algorithm as it is written in this thesis took a prohibitively long time to solve on realistic problem instances (300+ stacks compared to 40). However, improvements in computation time are possible by using a different programming language, using a more powerful computer, and other optimizations in the implementation.

Additionally, the performance of the solution can be further improved. We suggest studying other methods to implement the information process in the Markov Decision Problem, such that the amount of uncertainty in container arrivals can be better represented. We also recommend looking at other decision pruning methods, and further exploring possible features for the basis feature function.

Acknowledgements

"Then I saw all that God has done. No one can comprehend what goes on under the sun. Despite all their efforts to search it out, no one can discover its meaning. Even if the wise claim they know, they cannot really comprehend it."

Ecclesiastes 8:17, NIV

Before you lies my master thesis 'Solving the Dynamic Container Relocation Problem to Minimize Transportation Costs using Approximate Dynamic Programming', which is the result of a year of work at the company of Cofano. It also marks the end of my study Industrial Engineering and Management at the University of Twente. While it has been a challenging year, I feel like a learned a lot from an academic and a personal point of view. I learned a lot about doing scientific research, about modelling and programming, and got a sneak peek into the world of logistics companies. I want to thank my colleagues at Cofano for the easy welcome they gave me, and in particular I want to thank Leon, who orchestrated the assignment, and Quirijn, for the biweekly update meetings and interest in my research.

I also want to thank my supervisors at the university, Martijn and Eduardo. Their knowledge about the problem and solution method I used were indispensable, and their insights and remarks about my research kept me in the right direction.

I'm also very grateful for the support of my friends and family. I want to thank my housemates for listening to my rants when I encountered one too many bugs in my code, and my parents for supporting me throughout my entire study. I specifically want to thank Matthijs, Eveline and Jedidja, whose friendship and study-related advice have been immensely helpful when I had no idea what to do. Lastly, I want to thank Mette, who was a blessing and great support for me throughout the process of my thesis.

During my life as first a Civil Engineering student and then as an Industrial Engineering student I learned much about modelling, logistics, algorithms, teamwork, writing and thinking in a scientific manner in general. I had a great time, and I'm looking forward to the future, whatever it may bring.

Robbert Bosch,
September 2022

Table of Contents

Nomenclature and Abbreviations

Table 1 - Abbreviations used in this Thesis

Term	Meaning
20DV	20-foot Dry Van
20RF	20-foot Reefer
40DV	40-foot Dry Van
40RF	40-foot Reefer
ADP	Approximate Dynamic Programming
ASH	Average Non-Empty Stack Height
B&B	Branch & Bound
BD	Blocking Degree
BFF	Basis Feature Function
BLB	Blocking Lower Bound
BLD	Batch Label Difference
BRP	Blocks Relocation Problem
CRP	Container Relocation Problem
DLEBLB	Dynamic-Lookahead Expected Blocking Lower Bound
EBLB	Expected Blocking Lower Bound
EEBLB	Expanded Expected Blocking Lower Bound
FIC	Future Incoming Costs
FOC	Future Outgoing Costs
HUSP	Highest Used-Space Percentage
ILP	Integer Linear Program
LB	Lower Bound
LoS	Length-of-Stay
LP	Linear Program
MAE	Mean Absolute Error
MDP	Markov Decision Process
MMH	Min-Max Heuristic
MMV	Min-Max Value
MSE	Mean Square Error
MWSP	Minimum Wrong-Stacking Penalty
NES	Non-Empty Stacks
NIC	Non-Ideal Containers
NIS	Non-Ideal Stacks
PST	Partial Search Tree
R^2	R-squared
RIH	Reshuffle-Index Heuristic
RL	Reinforcement-Learning
SOS	Semi-Ordered Stacks
SSH	Squared Stack Height
TDLB	Travel Distance Lower Bound
TEU	Twenty-Foot Equivalent Unit
UB	Upper Bound

US	Unordered Stacks
USP	Used-Space Percentage
VFA	Value Function Approximation

Table 2 - Symbols used in this thesis

Symbol	**meaning**
α	Values of the cost-function
γ	discounting factor for future costs
ω	Random information
$\phi(S_t)$	basis feature function
π	policy
θ	weights of the VFA
a	arrival batch
b	order within arrival batch
c	stack
C	constant
$C_t(S_t, x_t)$	direct contribution to the cost
d	departure batch
e	exit
f	order within departure batch
i	Iteration or container id
n	simulation
n	entrance
p	tier
S_t	State at time t
S_t^x	Post-decision state S^x at time t
t	timestep
$V_t(S_t)$	Value of state S at time t
W_t	random information at time t
x_t	decision at time t

1. Introduction

In this chapter, we introduce the company of Cofano and their business, the problem of container handling at container terminals, and identify the problem Cofano experiences the problem this thesis will cover. Afterwards we determine the scope and research questions.

1.1. Company Introduction

Cofano is a company founded in 2010 that provides software-as-a-service in a variety of logistics sectors. They offer services for terminals, air transport, shipping, warehousing, and rail transport. With two offices and 50 employees, they are a small company, but their clients have used Cofano's software to handle over 2.3 million containers in the last year. In the sector for terminals, Cofano offers Terminal Operating Systems that help with the entire cargo handling process, from order management to physical handling of cargo.

One of the aspects that clients of Cofano are looking for in their Terminal Operating Systems is advise on the optimal placement of containers. Cofano offers in their software a basic container handling system, where all containers of one client can be designated to one stack, or where simple rules can be applied (e.g., 'do not stack a container on top of one that is needed today'). This software helps, but still depends on manual configuration, meaning the solution quality depends on the planner. As terminals are judged on KPIs such as the length of stay of trucks or ships, Cofano's clients would like to see as efficient container handling as possible, to improve the performance of their terminals.

1.2. Context Description

In modern logistics, containers have become an essential method of transporting goods; in 2020 over 9 billion USD worth of goods were transported using containers (Statista Research Department, n.d.). The fact that they are relatively uniform boxes whose contents do not need to be unpacked at each point of transfer makes them the standard for freight transport. Container sizes are expressed in terms of the twenty feet long standard container, also known as the Twenty Feet Equivalent Unit (or TEU).

Deep-sea general cargo is heavily containerized, with 849 million TEUs being brought through ports in 2021 (Statista Research Department, 2022). This means that these containers are loaded from or onto a ship twice or more, depending on whether they are first transported from a smaller port to a larger international port. The large and increasing number of container shipment causes high demands on seaport container terminals to keep the flow of incoming cargo smooth and fast. Seaports compete between each other for sea traffic, and the competitiveness of a port is mostly determined by the time in port for ships (transshipment time) and low prices for loading and unloading. This means that a crucial advantage for seaports is the rapid turnover for containers, which leads to minimization of the time a ship spends in port, and thus more satisfied customers and better utilization of seaport equipment.

One phenomenon that introduces inefficiency in the handling of containers is reshuffling: containers are stored on-site between arrival and departure, and containers are often stacked on top of each other to preserve space and minimize moving distance. Stacked containers have a Last-in-First-out retrieving order, which means that if a container needs to depart before other containers on top it, those need to be relocated somewhere else. The removal of containers blocking another container is called a reshuffle, and this move is inherently inefficient, particularly when this occurs during the loading of a ship. Reducing the number of reshuffles is a classic optimization problem called the Container Relocation Problem (CRP).

The CRP has many different variants and extensions, and all variants on the problem have solution spaces large enough to make exact solutions to this problem infeasible (Lehnfeld & Knust, 2014). In addition, the CRP problem has many variants depending on the type of problem and the objective. There exists a tradeoff between the ease of computation of solutions and applicability; most classic CRPs try to reduce the number of reshuffles and consider each reshuffle as identical in cost, but in reality, there is a difference in time cost between long-distance moves and short-distance moves. Additionally, some CRPs only consider one loading or unloading scenario, while in terminal operations a series of loading and unloading problems exist where the solution to one problem influences the terminal layout for the next problem. Furthermore, knowledge about the order and timing of arrivals is limited as the priority of containers within a ship, called the stowage plan, is mostly unknown before a ship comes to port. Some containers may even change destination last-minute, meaning that containers can be added or removed from the schedule. To deal with these complex aspects of the CRP, sophisticated algorithms are required, which is discussed in Chapter 3.

1.3. Problem Cluster and Problem Statement

This section identifies the action problem and the structure of problems that underly that problem, then we identify the core problem and lastly, we give the problem statement. The problem cluster (Figure 1) starts with the action problem that is being experienced by Cofano (no. 11); their software for scheduling container handling could offer a better service, so clients would like better tools to help reduce costs, which would lead to more customer satisfaction for Cofano. In the problem cluster given in Figure 1, this problem is demarcated in red.

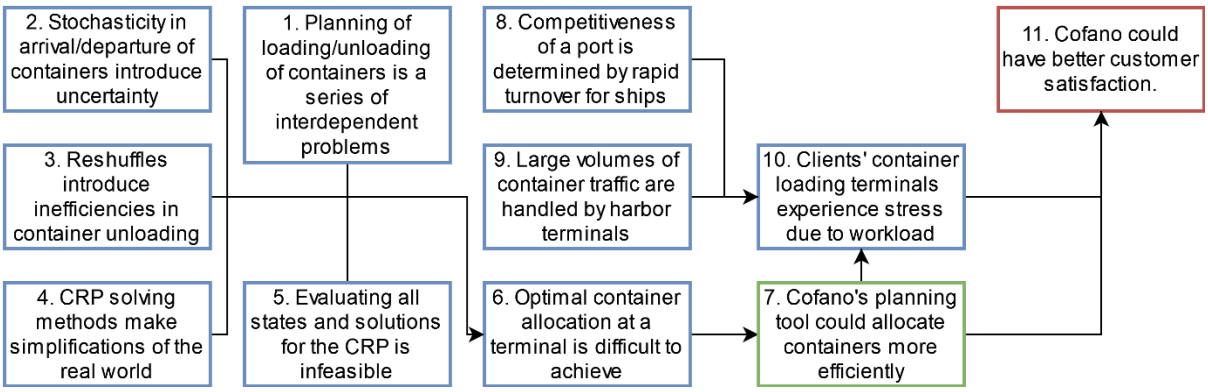


Figure 1 - Problem Cluster of Cofano

The action problem is caused by the lack of an efficient container planning methodology that Cofano offers within their tools (no. 7) coupled with the situation at their clients' terminals (no. 10): shipping terminals are strained in their workflow due to the amount of traffic that goes through them, and the fact that container loading is not sufficiently efficient. The volume of traffic going through terminals is not a problem we can solve, as this is a direct result of the number of containers handled by the terminal, i.e., their core business. We can however improve efficiency.

The source of inefficiencies are reshuffles, as discussed in Section 1.2. The necessity for container stacking and thus the source of reshuffles, cannot be prevented without a massive decrease in terminal capacity, but finding more efficient container handling schedules can alleviate the problem. The problem bundle ends with different reasons why current solutions to the CRP are not sufficient (no. 1 - 5). These characteristics are inherent to the real-life application of the CRP to terminals, so these cannot be changed, only their effects can be diminished. Following this logic, the core problem that can be addressed in this problem bundle is 'Solution methods to the CRP problem are not optimal'

(no. 7), which is what this research will focus on improving. With the core problem identified, we formulate the following problem statement:

> *"Cofano lacks an algorithm to create near-optimal container handling schedules for their clients, leading to a situation where unnecessary reshuffles are executed and containers travel unnecessarily long distances, introducing delay in transport, a reduction in terminal capacity, and an increase in handling cost."*

1.4. Scope

The goal of this research is to develop an algorithm that creates well-optimized schedules for container handling in terminals. To accurately define what is and is not part of the problem and this research, we line out the following assumptions and restrictions to arrive at the scope:

1. *The CRP is solved at an operational level; layout and machinery are assumed to be fixed, and demand is treated as unchangeable input.*

The handling of containers can be managed on different planning ranges: strategic (e.g., terminal design), tactical (e.g., client management) and operational (day-to-day physical handling of containers). This research focuses only on the operational level of container handling, so the terminal layout is assumed to be fixed. Client- and demand management is also on a higher planning range, so this is assumed to be unchangeable.

2. *The CRP is considered as a continuous combined loading and unloading problem.*

The CRP comes in several different variants, so it is important to distinguish what is and what is not considered part of the operational planning problem. A classification system of CRPs is given in (Lehnfeld & Knust, 2014), where part of the classification is the problem type. In alignment with the wishes of Cofano we treat the CRP as a continuous combined loading/unloading problem; a situation where several loading and unloading problems take place after each other.

3. *Objective of this research is to minimize transportation costs, expressed in the form of the number of reshuffles and distance travelled per container.*

Commonly in CRPs, the objective is to minimize the number of reshuffles. This goal keeps the problem instance abstract and thus easily generalizable. This means however that all moves are considered to take up an equal amount of effort, which is not the case in reality. An alternative objective is to minimize the transportation costs, which can be expressed as the weighted sum of the number of reshuffles and distance travelled per container (Li et al., 2019).

4. *This problem focuses on the restricted CRP, meaning that proactive container moves are not allowed.*

In literature on the CRP, two variants of the problem are distinguished: the restricted and unrestricted CRP. In the unrestricted CRP, any container may be moved at any time. This can be useful, as a container may be reshuffled voluntarily in order to accommodate a better destination for a container that is currently blocking a target container. In the restricted variant, only the top container of the stack containing the current target container may be moved. While this can lead to suboptimal solutions for a given stack layout, this restriction heavily reduces the available moves at any time (Jin, 2020). For practical purposes, this research will limit itself to the restricted CRP.

5. *Stochasticity in the arrival of containers is taken into account by assuming containers arrive and depart in batches; container priority within batches is only known upon arrival of said batch or ship.*

Arrival and departure of containers is semi-planned; ships and their contents are largely known in advance, but a stowage plan is only known hours beforehand. Additionally, the departure time for containers is not always known beforehand, as routes are subject to change and trucks that deliver or retrieve containers can be early or late depending on traffic conditions. To reflect this partial knowledge in a model, all containers are assumed to be arriving in a certain batch. Even if the order of containers within a ship is partly known, the content of a ship can simply be treated as a sequence of batches. The order of batches (as well as if they are inbound or outbound) is known in advance, but the order of containers within this batch is unknown until the batch itself arrives. Additionally, containers may arrive or depart earlier or later due to fluctuations in the arrival time of the truck or vessel that carries the container. This is reflected by sometimes changing the batch in which a container arrives or departs. In this way, this research will limit the knowledge that is available beforehand to reflect real-life scenarios.

6. *The types and lengths of containers that are considered are any common standard type of container present at the terminals of clients of Cofano.*

Most approaches to CRP assume all containers are uniform blocks. This massively simplifies the problems' state space and thus allows for faster solutions to the CRP. In reality, while there exists a standard size for container shipping (the Twenty-foot Equivalent Unit or TEU), containers can also be twice as long, forming a 2 TEU container. Thus, containers are not uniform in size and type, meaning they cannot all be stacked on top of one another. Additionally, there are restrictions on where containers are allowed to be depending on their contents (e.g., containers containing dangerous chemicals often have to be put in designated areas).

7. *The model is not designed for one particular terminal, but instead such that by adjusting the input and output of the model, the model can be trained and used by any terminal layout.*

Cofano wants to have a model that is applicable to multiple terminals, in order to have a product that can be sold to multiple clients. In designing a model for terminals, a trade-off needs to be made between generalizability and performance. Functions that can approximate values of certain states need to be different depending on the shape and size of a given terminal. Additionally, the schedule of unloading a ship is different when there are multiple cranes available as opposed to one. A cutoff needs to be made here. As most of Cofano's clients operate small to medium sized terminals, where one crane is available per ship, this assumption will be made for the remainder of this research.

1.5. Research Objective and Questions

With the problem statement and scope defined, we now define the main research objective and the research questions that need to be answered to achieve the research objective. We define the research objective as follows:

"Develop a model that creates container handling schedules for a container terminal that minimizes the transportation costs and ship transshipment times, while taking long-term costs into account."

To be able to develop this model, several research questions need to be answered. The first objective is to perform an analysis of the context of container handling at terminals, by looking at the logistics behind container terminals and the state of the art in how container handling schedules are made. The accompanying research question is:

1. What does container handling at terminals look like?
    a. What are the important places, movements and equipment at a terminal?

   b. How is the handling of containers at terminals planned in general and at Cofano?
   c. What is the nature of delays and information about delays at container terminals?

The second objective is to perform a literature review of state-of-the-art solution methods for the CRP and variants that are relevant for our research. A specific focus is on data-driven methods such as Approximate Dynamic Programming (ADP) and Reinforcement Learning (RL), as these methods are well-suited to solve problems that are too large to solve exactly, especially when the environment is stochastic (Powell, 2011). This gives the following research question:

2. What can we learn from literature on methods to solve CRPs?
   a. How are CRPs categorized?
   b. What methods exist for solving different types of CRPs?
   c. What methods exist for solving the CRP treated in this research?
   d. How can Approximate Dynamic Programming be used to solve the CRP treated in this research?

The third step in this research is to use the knowledge gained from the context analysis and literature review to design the model that can create container handling schedules for terminals in the manner as described in the Research Objective. The research question that covers this objective is:

3. How is the problem and the corresponding solution method treated in this research formulated?
   a. What are the characteristics and assumptions underlying the problem treated in this research?
   b. How is the Markov Decision Process formulated from the problem description?
   c. How are future costs of the Markov Decision Process accounted for when using Approximate Dynamic Programming?
   d. How is the optimal single-stage action found at each stage of the Markov Decision Process?
   e. What methods can be used to speed up the Approximate Dynamic Programming algorithm?

The next objective is to compare the performance of the new model against existing solutions in different contexts. The research question that addresses this objective is:

4. What is the performance of the proposed solution?
   a. What features are optimal to use to approximate future costs in a container terminal?
   b. How do parameters for the ADP algorithm influence the performance of the proposed solution?
   c. How can you best find the optimal way to allocate a single batch of containers?
   d. What is the best function to use to approximate future costs in a container terminal?
   e. How sensitive is the performance of the proposed solution to different circumstances?

This thesis has a chapter devoted to each of these research questions, with additionally the customary Introduction and the chapter Conclusions and Recommendations.

2. Context Analysis

This chapter answers the first research question: *"What does container handling at terminals currently look like?"*. We discuss the process of container handling at container terminals, terminal layout and equipment present at terminals, and introduce container scheduling and related problems in literature.

2.1. Introduction

From a logistics standpoint, container terminals are open systems of material flow with two external interfaces: the quayside, where ships are loaded and unloaded, and the landside (also known as hinterland), where containers arrive and depart via train or truck. The terminal contains equipment to move containers, as well as places to store containers between their arrival and departure, see Figure 2. Upon arrival at a port, a ship is assigned to a berth within a terminal, where cranes are assigned to load and unload containers onto the wharf behind it.

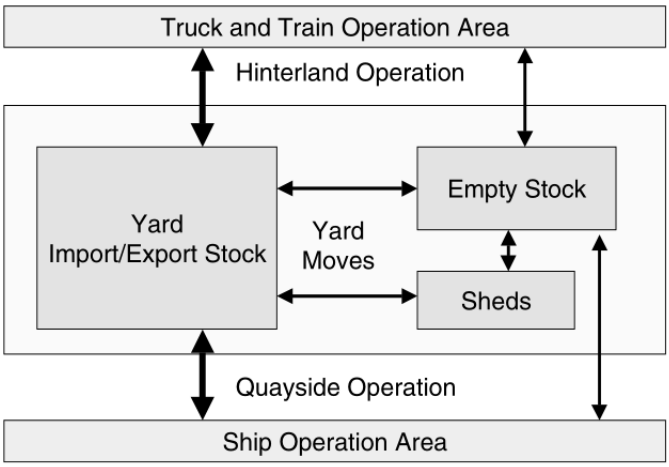


Figure 2 - Flow of transport within and around a container terminal (Steenken et al., 2004)

The yard of a terminal is the place where containers are stored, and is often divided into stacks, which are often grouped into blocks of stacks. These blocks usually have containers of the same length, or all contain empty containers belonging to the same company, for practical purposes. Blocks are further divided in the x direction into rows and in the y direction into bays, as shown in Figure 3. A particular [x, y] place containing containers is a stack, whereas a tier refers to the position in the direction of z.

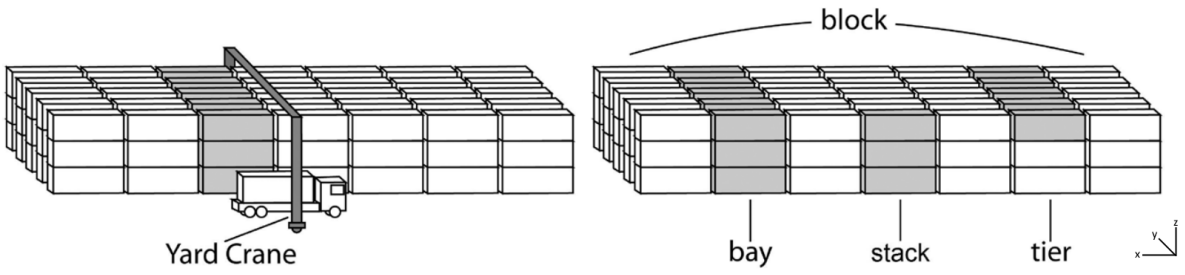


Figure 3 - Overview of the layout of containers in a terminal (Zhu et al., 2012)

Two types of ships are handled quayside: Deep-sea vessels (around 2000 TEU) which travel to main ports between countries and continents, and feeder vessels (around 100-1200 TEU), which link smaller

ports with oversea ports via sea, rivers or channels. Landside, trucks (1-2 TEU) and trains (up to 120 TEU) handle transport land inwards. Container handling at quayside and landside are separated; quayside containers are typically handled using quay/gantry cranes, and landside containers are handled using straddle carriers or reach stackers.

2.2. Containers

Containers are the units that go from point A to point B through the terminal, where it is stored while waiting for their retrieval by a different (type of) vessel. The come in different sizes and types; the 'standard size' of a container is defined as the twenty-feet equivalent unit (TEU), and has a length of 20 feet, a height of 8 feet 6 inch and a width of 8 feet. Container sizes can differ however: container length can be 20feet, 40feet or 45feet, with additionally different types having different dimensions. The most common types of containers are:

Table 3 - List of common container types

Abbreviation	Name	Explanation
DV	Dry van	Standard sized container.
HC	Highcube	Container with 1feet extra height.
PW	Pallet Wide	Container with 1feet extra width.
HW	Highcube Pallet Wide	Container with 1feet extra width and height.
RF	Reefer	Container with cooling for perishable products.
TK	Tank	Cylindrical tank with container frame for chemicals.
OT	Open Top	Container lacking a roof, for oversized cargo.

Containers are referred by [length][type] (e.g., 40HC for a 40feet Highcube container). The most common container type in use currently is the 40HC, see Figure 4. An additional important property is whether a container is empty, as any empty container of the same size/type is interchangeable with another, if they both have the same owner. In the distribution of container types, empty containers are denoted by the '_E'-suffix. Empty containers are assigned to a separate bay of empty containers due to their interchangeability and are left out of the research beyond this context analysis.

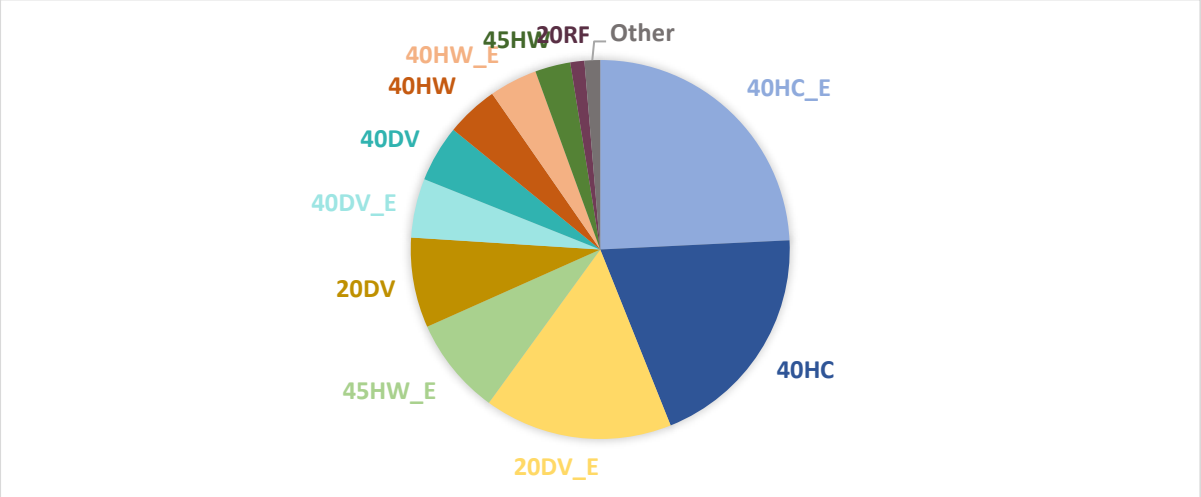


Figure 4 - Distribution of container types at client X

These different container types have restrictions about where they can be placed and how they can be stacked. While stacking different lengths of container is technically possible, in practice different container lengths are stored separately as much as possible, as this reduced planning complexity. Reefers need electricity to keep running, which restricts their placement. Open top containers can technically have others on top of them, but in practice these are used to store oversized cargo, which

means no other containers can be stored on top. DV, HC, PW, HW containers as well as reefers and tanks can be stacked on top of each other, but tanks often contain dangerous chemicals which limits their placement; some are not allowed to be stored near each other, while others have to be stored separately (Steenken et al., 2004). The specifics of these restrictions can change depending on the country, but in general 9 classes of dangerous goods are classified by the Economic Commission for Europe Inland Transport Committee (Committee, 2021).

While many types of containers with specific stacking restrictions exist (open top containers, containers containing dangerous chemicals, etc.), 98% of containers are normal containers with either an extra foot of height and/or width (HC, DV, HW, PW), and of the remaining 2%, the majority consists of reefer containers. These have the stacking constraint that they are confined to stacks where access to electricity is available, so their contents can be cooled. For this reason, only a distinction is made between 20/40ft containers and regular containers and reefers.

2.3.Equipment

The equipment present on a container handling terminal depends on the port, location and the volume of container traffic it needs to process. The port of Shanghai, the largest port in the world (Zhao et al., 2013) since 2008, has over 13km of quay length, 43 berths and 156 quay cranes. On the other end of the spectrum are small river ports that ship cargo to and from deep-sea ports, where cargo is loaded onto bigger, more efficient ships.

Figure 5 - Photo of Shanghai port (left) vs the port of Meppen (right), (MFAME, 2019; RTVDrenthe, 2021)

Equipment on a container terminal can be divided into quayside equipment and landside equipment. Quayside equipment consists of quay cranes, dedicated to loading/unloading ships. Depending on the type of port, different sized ships can visit the port, requiring different sizes of quay cranes. On the landside, containers are transported from quay crane to storage and from storage to the hinterland using straddle carriers, reach stackers or a combination of trucks and gantry cranes. These types of equipment are shown in Figure 6.

Figure 6 - landside equipment for a container terminal: straddle carrier (left), reach stacker (middle), and gantry crane (right)

Depending on the type of equipment used, the restrictions for where containers might be located changes. Gantry cranes can put any container on top of any stack, as long as the stack height does not exceed the height of the crane. Straddle carriers can put a container on top of another one as long as the stack height does not exceed the straddle carrier height and the straddle carrier can drive up to the stack itself (e.g., it is not surrounded by other stacks). Reach stackers lift diagonally, so in addition to the restriction that a stack may not exceed height *n*, at least one neighboring stack in the same bay may not exceed height *n-1*, and the stack next to that *n-2*, etc. These different stacking methods impose restrictions on the layout of a container terminal.

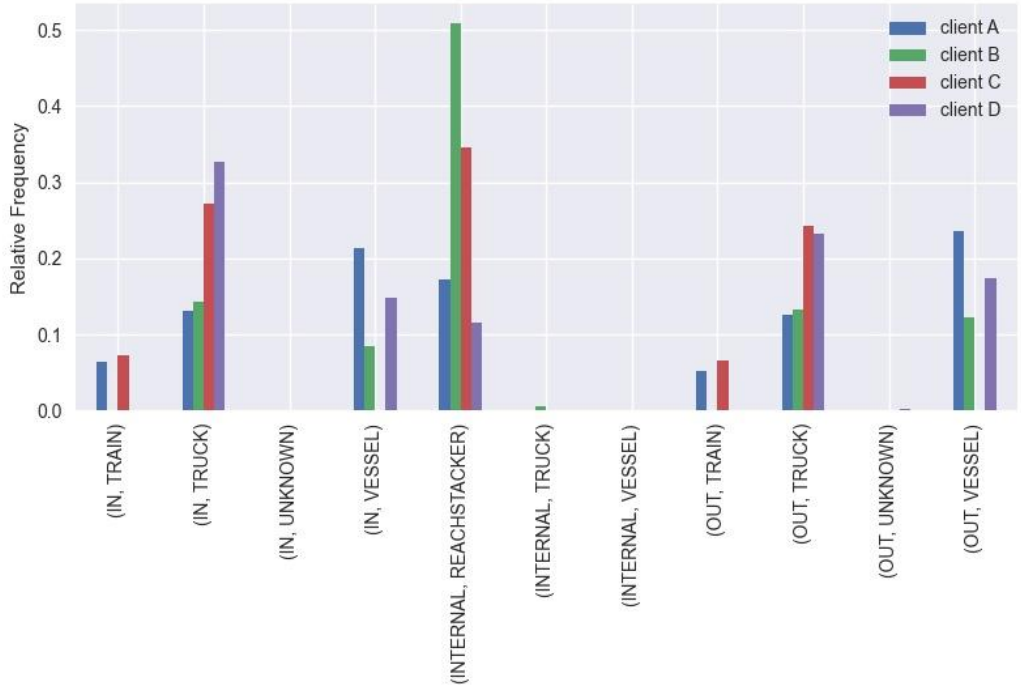


Figure 7 - Histogram of the handling modality and movement type per client

We thus see that the equipment present at a terminal can vary between terminals, which leads to different stacking constraints. Figure 7 shows that almost all internal movements (also known as reshuffles) are performed by reach stackers at Cofano's clients. However, other research done using data from the clients of Cofano has already focused on the stacking restrictions imposed by reach stackers, and using these restrictions limits this research's applicability to terminals with other types of stacking equipment. This research will thus not focus on that aspect of container stacking (Boschma, 2020).

2.4. Container Allocation

The ship planning process takes place at different levels and contains different types of optimization problems. In general, three different levels of planning are distinguished: strategic, tactical and operational. Common decision problems encountered at each level are (Steenken et al., 2004):

1. *Strategic*: equipment acquisition, terminal design
2. *Tactical*: personnel scheduling, berth allocation, stowage planning, crane split
3. *Operational*: container storage logistics

At the strategic level (planning horizon of multiple years), long-term decisions are being made, such as design of new or existing container terminals, and acquisition and maintenance strategies related to the amount and type of equipment. At the tactical level (planning horizon of several days to several months), semi-long term planning decisions take place. These include the scheduling of personnel as well as the allocation of ships to berths in a terminal. Additionally, cranes are assigned to ships and ship sections. There are constraints on the accessibility of cranes at a berth as well as constraints on what ship types a crane can service, and often a range of cranes are present at a terminal due to organic growth of a terminal over time. At the shipside of operations, stowage planning is done, where containers are grouped according to their destination, weight and length, and assigned to minimize the number of shifts required during loading/unloading, maximize utilization of the ship and ensure stability. At the operational level, container stacking logistics takes place. This is also the level of planning where the subject of this research takes place.

Cofano offers a container allocation planning tool. Extensive information on this tool was not available, but a rudimentary explanation of the tool is provided by an employee. When a container is picked up, the operator of the crane or reach stacker gets suggestions for a location of the container. These suggestions are based on restrictions on the location of a container and decision rules (e.g., “do not place a container on top of one that needs to be retrieved today”) and are determined by the client. The final decision on where a container is located however, depends heavily on the operator. The number of reshuffles per container for clients of Cofano is shown in Figure 8. The number of reshuffles differs per client; containers are on average reshuffled 0.24, 1.66, 0.77 and 0.18 times for each client respectively. The maximum number of reshuffles encountered was 35 times.

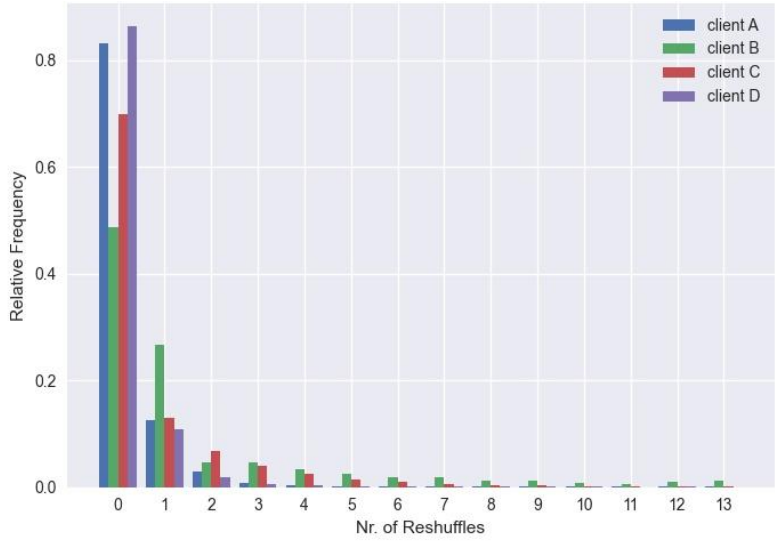


Figure 8 – Histogram of the number of reshuffles per container for each client

The container arrival process and length of stay is also highly dynamic. The intensity with which containers arrive differs per client and can change heavily depending on the time of day and weekday, as demonstrated in Figure 9. Containers mostly arrive on weekdays between 7:00 and 18:00, but outliers exist as well. Length-of-stay of containers (shown in Figure 10) has a wide range and is also dependent on the client. While no data is available on the occupation of terminals at any given time, the fact that containers arrive in irregular intensity while their length of stay is also irregular suggest that the occupation of a terminal can vary wildly over time.

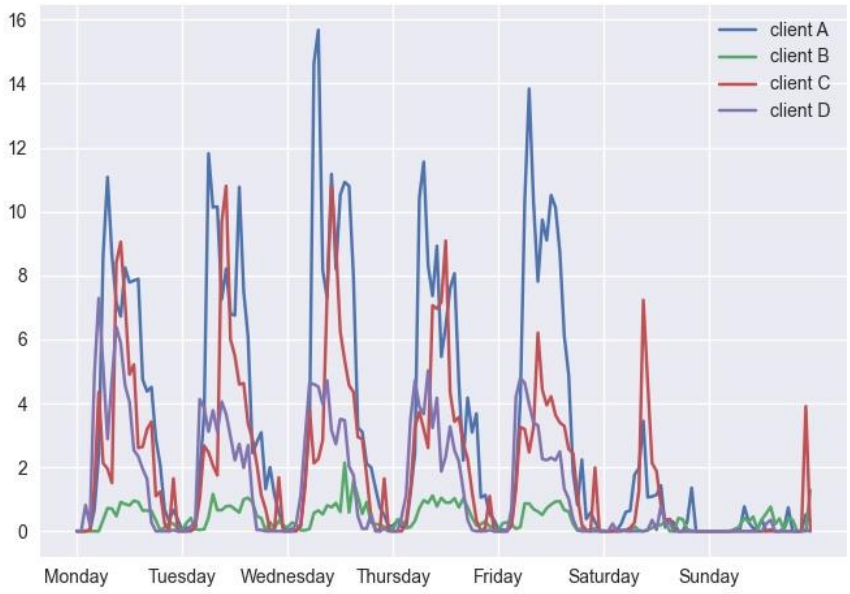


Figure 9 - Arrival frequency over a week for each client

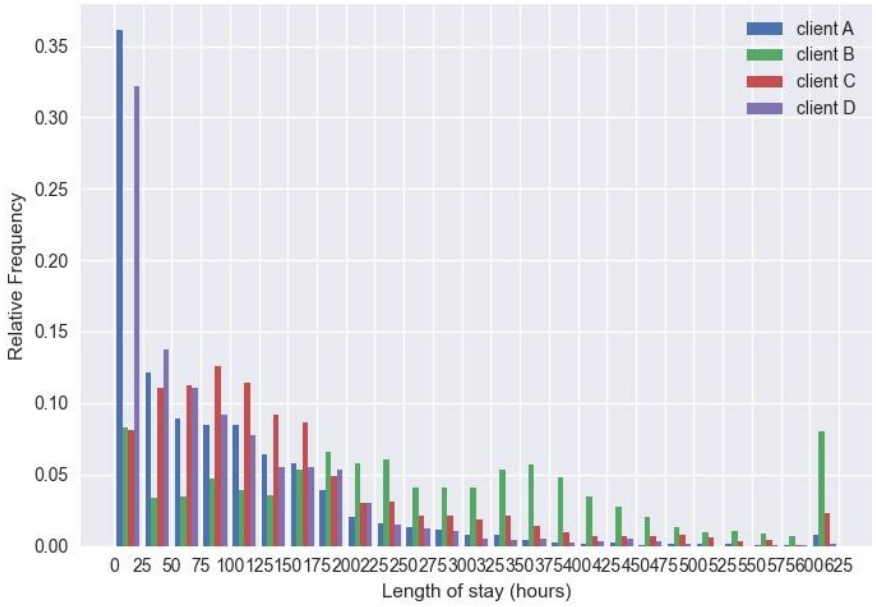


Figure 10 - Histogram of the length of stay of containers for each client. A cutoff is made for >625 hours.

2.5. The Information Process

Container terminals have limited information about the arrival times of vessels that bring in / take out containers. While Cofano did not have quantitative data available about the delays vehicles experience, an employee was able to provide a description about the nature of delays. In general, a schedule about arriving and departing containers is made days to weeks in advance. The actual time of arrival and departure however depends entirely on when a truck, train or ship arrives at the terminal. The type and amount of delay experienced by containers is dependent on its mode of transport. Ships have planned and realized delay; the majority of ships is delayed by a day or more, which is communicated to port authorities. This type of delay is planned delay and can be accounted for in planning of container handling. The exact time of arrival can still vary several hours. Trains are relatively reliable. Trucks can get stuck in traffic and experience delays of an hour to several hours. Delays and changes in arrival are also dependent on whether a container is retrieved or delivered to the terminal. As ships can have delays of several days, trucks retrieving containers that arrive by ship are only dispatched from the party retrieving the container when the ship has arrived at the terminal. There is no adaptation however when a truck delivers a container to be retrieved by a ship, so a delay of arrival of a ship can result in a pile-up of containers that are retrieved by a ship.

2.6. Conclusions

This chapter looked at the context where container handling within a terminal takes place. We discovered that container terminals are an interface between different modes of transport. We determined the relevant types of containers that attend terminals of clients of Cofano, which are 20HV, 40HV, 20RF and 40RF. The constraints that these types of containers impose on container stacking at a terminal is that in general, containers of different lengths are supposed to be stacked separately from each other, and reefer containers impose the restriction that these can only be stored in stacks that have access to electricity, to allow for cooling of the container. In addition, we looked at equipment of different types, and what restrictions these have when it comes to stacking containers. The available stackers and cranes available depend on the size of the terminal and the modes of transport available to the terminal, with large terminals that are attended by ships generally having quay cranes and gantry cranes available, which can transport a container horizontally and vertically, while land-based terminals generally operate using reach stackers, which can only stack diagonally.

The planning of container handling is done at three levels: strategic (equipment acquisition and terminal design), tactical (personnel scheduling, berth allocation, etc.) and operational (container storage logistics). The problem handled in this research concerns planning on the operational level. Cofano offers a rudimentary container allocation tool that provides suggestions for container placement. While helpful, this still results in situations where some containers are reshuffled >3 times, and up to 35 times.

Container logistics are made more difficult due to delays in vehicle arrivals and thus in container arrivals and departures. These delays can range from hours in the case of trains to days in the case of ships. Trucks will wait to retrieve containers until they have arrived, but trucks delivering containers to be taken by ship will not wait if a ship is delayed, which can result in pileups of containers. Information about delay is partial; ships experiencing delay of one or multiple days will notify terminals, but delay on the day of arrival is unknown until the ship arrives at port.

3. Literature Review

This chapter discusses the second research question: *"What can we learn from literature on methods to solve CRPs?"*. In doing so, we discuss the mathematical model behind different CRPs, popular methods to solve CRPs, and introduce ADP and relevant aspects of ADP.

3.1. Container Relocation Problems

As discussed in Section 1.2, the problem of preventing reshuffles when handling containers is called the Container Relocation Problem (CRP). This section introduces the concept of the CRP as well as variants that exist within this class of problems. Additionally, a classification scheme is discussed to concisely distinguish which variant of the CRP is being discussed, and the problem that this thesis concerns is given a classification.

3.1.1. Basics

Consider a storage area that is arranged in stacks with fixed positions in a two-dimensional area. Let $\mathcal{M} = \{1 \dots m\}$ be a set of stacks. We introduce a set of items $\mathcal{N} = \{1 \dots n\}$ that need to be stored in this area, where typically $m < n$. From a theoretical standpoint these items can be any stackable identical item, such as coil springs, boxes or containers. In this case the problem is referred to as the Blocks Relocation Problem (BRP), which is the most general form of the problem. The problem context of this paper has containers as the item to be stacked, so that term will be used from this point on. The CRP is a subsection of the BRP, where basic assumptions are used from the environment of a container terminal.

In practice, containers might not be the same size, but containers of different sizes are often stored in different places, so we assume that all items are the same size. Items are moved by cranes above the stacks, so stack size is limited to a number of items p. An item is blocked if one or more items with later retrieval times are stacked on top of it.

Typically, the storage area is represented as a rectangular bay which is divided into C columns, each having R rows that are filled from the bottom up to r, with $r \leq p$. Container terminals are usually divided into blocks, as mentioned in Section 2.1. The distance between stacks of the same block is much smaller than the distance between stacks of different blocks, which is relevant if one is not only concerned with the number of reshuffles but also other metric such as total distance per container or travel time per container (Lehnfeld & Knust, 2014).

3.1.2. Loading, Unloading and Premarshalling

CRPs can be divided into several categories of problems based on whether containers loaded into the terminal, containers are unloaded from the terminal, or a mix of these processes happens. Additionally, a variation of the loading problem called the premarshalling problem exists.

Loading problems are concerned with storage of incoming items. Pure loading problems assume no items are retrieved during the loading process. The solution to such a problem is a schedule with the picking order and location for each incoming item. The objective is often to minimize the expected number of reshufflings, as individual retrieval times are often unknown (Lehnfeld & Knust, 2014). Unloading problems deal with retrieval of items. One must define the order to retrieve items in, and if reshuffles are needed, where they should go. Dependent on the problem, the order of items to be retrieved can be (partially) fixed.

A variation on the unloading is premarshalling, where items are sorted in storage such that when the time comes that they have to be retrieved, this can be done in the shortest time or with the lowest number of required reshuffles.

Combined problems occur as a combination of loading and unloading, and possibly also premarshalling problems. These can be formulated as a sequence of individual problems of the loading, unloading and premarshalling kind.

3.1.1. CRP Problem Variants

Aside from loading, unloading, premarshalling and a combination of those, CRPs are also defined by the sequence in which containers are handled and the amount of freedom in the order of handling containers. When discussing the order of container handling, we distinguish sequences (ordered groups of containers) sets (unordered groups of containers).

In loading problems, it is common to store a set of incoming items since a train or ship provides several containers to be unloaded at once. Alternatively, a sequence of incoming items can be given, when the order of handling is preset. The order in which containers from a ship must be handled is a stowage plan. Deviations from the stowage plan are sometimes possible and can improve efficiency, but are commonly not considered, see (Jovanovic et al., 2019).

In unloading problems, items often must be retrieved in a predefined sequence given by the order in which trucks arrive. Another possibility is that a sequence of outgoing item sets is given. In this case it is known in advance which container has to be loaded on which ship, but not the order in which containers for any given ship have to be handled. Other applications are based on items being grouped by weight or other attributes. Sets of sequences or sequences of sets can also be given

Aside from the order in which containers are handles, additional constraints or possibilities allow for different variants and applications of the CRP. Examples of these are: additional temporary storage, grouping of empty containers, stacking restrictions (weight, type, etc.), location constrictions, and different objective functions. Conventionally, the objective is to minimize the number of reshuffles, as it is generally applicable to all CRP contexts and not dependent on equipment present at a terminal. Purely considering the number of reshuffles keeps inconsistencies of the real world out of the optimization model. Another option is minimization of costs, given by travel costs or penalties for (soft) violations.

3.1.2. Problem Classification

A classification scheme of loading, unloading, premarshalling etc. problems is given in (Lehnfeld & Knust, 2014). It is denoted by $\alpha|\beta|\gamma$, and denotes the type of problem, specifications around the containers, and the objective. The α field stands for the type of problem:

$$\alpha_1 = \begin{cases} L & if\ a\ loading\ problem\ is\ considered \\ U & if\ an\ unloading\ problem\ is\ considered \\ P & if\ a\ premarshalling\ problem\ is\ considered \\ LU & if\ a\ combined\ loading\ unloading\ problem\ is\ considered \\ LP & if\ a\ combined\ loading\ premarshalling\ problem\ is\ considered \end{cases}$$

With additional fields α_2, α_3 and α_4 for determining the number of available stacks, the height of available stacks, and the presence of temporary storage area respectively. There is no α_1 notation for a combined loading, unloading and premarshalling problem, as a combined unloading and premarshalling problem decomposes into several independent premarshalling problems (Lehnfeld & Knust, 2014).The β field specifies restrictions around the containers:

$$\beta_1 = \begin{cases} \pi^{in} & if\ a\ sequence\ of\ incoming\ sets\ is\ given \\ \mathcal{L}^{in} & if\ a\ set\ of\ incoming\ items\ is\ given \\ (\mathcal{L}^{in})_K & if\ a\ sequence\ K\ containing\ sets\ of\ incoming\ items\ is\ given \\ \{\pi^{in}\}_K & if\ a\ set\ of\ K\ sequences\ is\ given \end{cases}$$

Fields β_2 through β_7 give information on the presence of stacking restrictions, weight limits, height limits, location restrictions, the possibility of voluntary moves and if items have to be pushed back to their original location if not retrieved. The γ field involves de objective function. The most popular objective functions are:

$$\gamma = \begin{cases} \#RS & min\ the\ number\ of\ reshuffles \\ E(\#RS) & min\ the\ expected\ number\ of\ reshuffles, in\ case\ of\ incomplete\ information \\ TC & min\ transporation\ costs, mostly\ given\ as\ time\ needed/distance\ travelled \\ f & minimize\ given\ (multi\ objective)\ function\ f \end{cases}$$

Using this classification scheme, we can classify the problem under consideration in this research. In Section 1.4 the scope of the problem was formulated. The problem that is discussed is a combined loading/unloading problem. The layout of the terminal under consideration is given as part of the input, so that determines the number of stacks and the limit of number of items within each stack. In container terminals, there is no temporary storage available, so the α field of this problem is given by *LU.*

The input of the problem is given as a sequence of sets of containers, where the first set in the sequence has its sequence revealed, following restriction 5 in the scope. This is different from any option the classification scheme provides, so a new option is defined $\left(\mathcal{L}_{k\geq 2}^{in\ \&\ out}; \pi_{k=1}^{in\ \&\ out}\right)_K$. This research does respect stacking and location restrictions surrounding reefers, dangerous goods etc. as defined in Section 1.4, assumption 6. We do not allow proactive moves as explained in Section 1.4, restriction 4, and items do not have to be pushed back. The β field is thus defined as $\left(\mathcal{L}_{k\geq 2}^{in}; \pi_{k=1}^{in}\right)_K, s_{ij}, L_i$.

The γ field defines the objective function, which as per restriction 3 in the scope is defined as minimizing the transportation cost, expressed as a combination of number of reshuffles and distance travelled per container. (Li et al., 2019) define a flexible objective function x_1*#RS + x_2*D, where D stands for the average distance travelled per container. As this problem contains stochastic input, the objective is adapted to be x_1*E(#RS) + x_2*E(D). The complete α|β|γ classification of this CRP is then $LU|\left(\mathcal{L}_{k\geq 2}^{in}; \pi_{k=1}^{in}\right)_K, s_{ij}, L_i|x_1\mathbb{E}(\#RS) + x_2\mathbb{E}(D)$.

3.2. CRP Solution Methods

There have been numerous papers into the CRP and related problems, with a variety of different approaches. This section addresses these previous works, first by looking at research that addresses individual loading/unloading problems and then by looking at combined loading/unloading problems. Special attention is paid to research that has used ADP and ways in which the value of states of a CRP can be approximated, as that is relevant for our approach.

3.2.1. Individual Loading / Unloading Problems

The $U|\pi_{in}|\#RS$ CRP is the most common form of the CRP and has a significant amount of research done. These often assume a static environment, in which all information is available from the start. Multiple papers have formulated mathematical models / LPs of the unloading CRP, with varying approaches to solve this mathematical model, such as Branch and Bound, heuristics or LP solvers (Forster & Bortfeldt, 2012; Jin, 2020; Kim & Hong, 2006; Wan et al., 2009).

Loading CRPs are less straightforward, as there is not a direct measure available to evaluate the quality of a certain layout; the appearance of reshuffles only occurs during retrieval of containers, so the quality of a layout only becomes apparent upon unloading. Nevertheless there is research done into these problems, often called the Container Allocation/Storage Problem (Jovanovic et al., 2019; Li et al., 2019; Sikorra et al., 2021; Zhang et al., 2014).

3.2.2. Combined Loading / Unloading Problems

Combined *LU* problems are more diverse in the problem environment they try to solve and in their approaches. Almost all of the research done on these types of problem use a sequence of sets of alternating incoming and outgoing containers. (Güven & Eliiyi, 2014; Ries et al., 2014) use a 'Value of Goodness' to first allocate a container to a bay based on characteristics such as bay utilization and quay-bay distance, and then to a place within that bay based on characteristics such as expected time until departure for containers in that bay. These algorithms provide rudimentary but effective methods to reduce the expected number of reshuffles and total transportation costs respectively.

There are also multiple papers that tackle the Dynamic CRP; a variant of the unloading CRP where a sequence of batches needs to be loaded and unloaded, which is formulated as a series of *U/L* CRPs that are solved sequentially. These problems keep information on future batches of containers out of the problem until they are handled, in order to account for the limited amount of information available in advance in real-world terminal operations (Tang, 2015).

3.2.3. ADP Approaches

There has been little research on using ADP to solve the CRP. (Zhang et al., 2014) construct a Stochastic Dynamic Program (SDP) to solve the container loading problem, by constructing a punishment metric $r(X, D, k)$ based on assignment of a new container k to a heaver stack D, given input state X. Overall objective of this method is to efficiently stack containers by incurring cost based on stacking of containers using weight categories. To solve this for large problem instances, the SDP approach is replaced with an ADP approach. Additionally, (Boschma, 2020) has done research also at Cofano into the combined LU CRP using ADP, with minimizing the expected number of reshuffles as the objective. A linear value function approximation is used to estimate the value of a certain layout, which is a linear combination of features that approximate the value of a state. This concept is elaborated on in Section 3.2.4. This is to our knowledge currently the only research that tackles the combined LU CRP using ADP, so the model formulated in this paper will provide a novel approach to this problem.

3.2.4. Value Approximations

Most methods that solve non-trivially sized versions of a CRP make use of approximations of the value of a certain layout in order to approximate the long-term value of a decision, as it is established that evaluating all states is not feasible due to the NP-hard property of the CRP. In ADP, and the remainder of this thesis, these approximations are referred to as feature functions. This section discusses some lower bounds, upper bounds and other characteristics that are important.

Lower Bounds

Using a lower bound on the number of reshuffles to estimate the quality of a certain layout without evaluating the exact number of reshuffles necessary is a popular method in research on CRPs. The simplest lower bound that is commonly used is known under many names such as Optimistic Expected Number of Additional Reshuffles (Optimistic ENAR) (Kim & Hong, 2006), LB1 (Zhu et al., 2012) or Blocking Lower Bound (BLB) (Galle, 2018), and simply assumes that for any container blocking any below it, it is relocated once (as if to a temporary infinite storage area), and never again.

More sophisticated LBs try to account for where reshuffled containers can be placed when they are reshuffled. (Zhu et al., 2012) expand on the simple lower bound by checking if the reshuffled containers from LB1 can be put on any stack where all containers are retrieved later than it, where an additional reshuffle is added to get LB2. LB3 expands on that by looking at what container causes the reshuffle. Let's say container 16 in the retrieval order is blocking container 4 in a given layout, then the layout is considered where containers 1-3 are removed, as well as their blocking containers. Container 16 then causes another reshuffle if it cannot be placed in that new layout without blocking.

Another lower bound, called ELB4, is used in (Zhang et al., 2020). This lower bound preserves the relocated container if it becomes the highest priority container in that stack, and then looks ahead to the next container and determines the lower bound using a similar lower bound to LB3. At this point however, the lower bound looks more like a small tree search that returns the best lower bound and is thus not computationally efficient according to the paper.

For stochastic variants of the CRP an expected BLB is formulated, where for containers that might block a container with the same batch number, the chance of blocking is calculated based on a uniform distribution of order within that batch. For any two containers, this is 1/2, but this becomes more complicated as more containers of the same batch end up in the same stack (Galle, 2018).

Upper Bounds

Upper bounds are usually generated from partial solutions that are derived using a fast heuristic. One such heuristic is the Reshuffle Index (RI) heuristic: for each column in a yard-bay, the number of relocations necessary to retrieve the earliest departing container from that column is determined. Any container is then assigned to the stack with the lowest RI (Hakan Akyuz & Lee, 2014; Wan et al., 2009). For each container that needs to be reshuffled, the destination is determined by finding the RI for each non-empty stack. Another relatively fast heuristic is the Min-Max heuristic (Caserta et al., 2012), where the first outgoing container of each stack is found, disregarding empty stacks. If any stacks exist with an earliest retrieval order higher than the container under consideration (an incoming or reshuffled container), the stack among that group with the earliest retrieval order is chosen. Otherwise, an empty stack is chosen. If that is not available, the stack with the highest retrieval order is chosen.

A more computationally heavy heuristic is the $MRIP_k$ upper bound, where the 'minimize reshuffle integer program' of the CRP is solved exactly for the next K containers, where K is any number lower than the number of containers in the terminal. The first container is then relocated according to the optimal solution of that subproblem, and then the $MRIP_k$ problem is solved for containers 2 through K+1, until no more containers are left (Wan et al., 2009). Another upper bound is the Virtual Relocation Heuristic (VRH), consisting of two stages. First, all blocking containers above the target container are sorted by descending priority number. Then the well-relocated stacks (stacks that do not have any blocking containers) for the blocking containers are determined, and the stack with the top container with the highest priority number is chosen. If no such stack is found, the container is relocated in the second stage. There, the remaining containers are ranked by ascending priority number and relocated, where possibly they must be inserted into a stack that had a container relocated to it in stage one. The stack with the lowest number of blocking containers is then selected.

Other Indicators

There are other characteristics of a terminal layout that do not give a lower or upper bound, but instead give some form of estimate for the number of reshuffles. One of those is the pessimistic ENAR (Kim & Hong, 2006). This ENAR differs from the Optimistic ENAR lower bound, in that the assumption

is that containers are not located to a separate location but relocated to a random location within the available area. This is expressed using E(k,n), which is the expected additional relocations resulting from future blocks relocated to the empty slots of a stack with blocks whose highest priority is n and empty slots which is number k. E(k,n) can be calculated recursively, starting at E(1,n) using the formulas:

$$E(k,n) = \frac{\sum_{i=1}^{n-1} E(k-1,i)}{N-T+k} + \frac{N-n-T+k+1}{N-T+k} * \{1 + E(k-1,n)\} \quad (1)$$

$$E(1,n) = \frac{N-T-n+2}{N-T+1} \quad (2)$$

Where T is the maximum height of a stack in the future, n is the highest priority number in a stack, k the number of empty slots in a stack under the assumption that blocks are filled up to tier T (Kim & Hong, 2006).

Another measure is the blocking degree proposed in (Jiang et al., 2021). This does not determine the number of reshuffles but sums the amount of priority difference for every blocking container. It does this by determining the container with the highest priority and dividing the stack in an upper and lower part. For the upper part, the difference in priority is summed up. For the lower part, this procedure starts again by identifying the highest priority container in that stack. Lastly, some papers combine a lower and upper bound method, and combine them in a weighted sum to get a composite measure (Scholl et al., 2017). The paper uses the minimax rule as upper bound, and $x \in$ {blocking number, confirmed relocations, confirmed relocations with lookahead} as possible lower bounds, giving the formula:

$$CM(x,\alpha) = \alpha * MM - (1-\alpha) * x \quad (3)$$

3.2.5. Overview of CRP Solution Methods

This section discussed multiple theses that solve (a variant of) the CRP. Table 4 shows an overview of all discussed theses, with classification of the CRP type as mentioned in Section 3.1.2. Many different ways of formulating CRPs are possible, depending on the goal of the research. Research is mostly either a more sophisticated method of solving a more abstract version of the CRP, or a rudimentary method of solving realistically sized CRPs. Studies on the Unloading CRP are most common, as this problem has a well understood objective. The performance of Loading CRPs is more difficult to evaluate, as the costs (reshuffles) incurred during loading only become apparent when unloading, so this would have to be included in the problem definition. Search tree related methods are popular for deterministic versions of the CRP (A*, B&B, BSFT, LP), but for larger problem instances these methods are assisted by heuristics to truncate the decision space. Realistic problem instances are often solved entirely using heuristics.

Table 4 - Overview of theses on the CRP. Solution Method abbreviations are: RL – Reinforcement Learning, B&B – Branch & Bound, LP – Linear Programming, GA – Genetic Algorithm, NS – Neighbor Search, BFST – Best-First-Search Tree, FL – Fuzzy Logic.

Work	CRP type	Goal	Variations	Solution Method	Approximations
(Sikorra et al., 2021)	$L \mid \pi$	$E(\#RS)$		RL, Deep Q-learning	-
(Boschma, 2020)	$LU \mid \{\mathcal{L}^{in}, \mathcal{L}^{out}\}$	$E(\#RS)$	State abstraction	ADP	Multiple
(Jiang et al., 2021)	$U \mid \pi$	$E(\#RS)$		RL + heuristic	Blocking Degree
(Zhang et al., 2020)	$U \mid \pi$	$\#RS$		B&B	Multiple UB/LBs; ML improved LB
(Jin, 2020)	$U \mid \pi$	$\#RS$		LP	-
(Li et al., 2019)	$L \mid \pi$	$\#RS$ & TC	Real-life terminals	GA	-
(Jovanovic et al., 2019)	$L \mid \mathcal{L}$ & π	$\#RS$	Semi-restricted	NS	-
(Galle, 2018)	$U \mid \{\pi\}$	$E(\#RS)$		BFST	-
(Scholl et al., 2017)	$U\|\pi, UP\|\pi$	$\#RS$		Heuristic	UB/LB Composite
(Tang, 2015)	$U\|\pi, LU\|\{\pi^{in}, \pi^{out}\}$	$\#RS$		LP	-
(Hakan Akyuz & Lee, 2014)	$LU\|\{\pi\}$	$E(\#RS)$		LP with heuristics	Column index, UB heuristic, beam search heuristic
(Ries et al., 2014)	$LU\|\{\pi\}$	$\#RS + TC$		FL	Time-until-departure
(Güven & Eliiyi, 2014)	$LU\|\{\pi\}$	$E(\#RS)$		2-Stage Heuristic	Stack utilization & Time-until-departure
(Forster & Bortfeldt, 2012)	$U\|\pi$	$\#RS$		Min-Max Heuristic	-
(Zhu et al., 2012)	$U\|\pi$	$\#RS$		A*-algorithm	Multiple LBs
(Wan et al., 2009)	$U\|\pi$	$\#RS$		Iterative LP	-
(Kim & Hong, 2006)	$U\|\pi$	$\#RS$		B&B	Optimistic & Pessimistic ENAR

3.3. Approximate Dynamic Programming

This section introduces the concept of ADP by first introducing the Markov Decision Process, and the changes to this problem that then form the basis for ADP. Afterwards, design choices that are important about ADP are discussed.

3.3.1. Markov Decision Processes

Markov Decision Processes (MDP) are a class of optimization problems. Assume there is a state space $\mathcal{S}$, and at each point in time t you are in state $S_t \in \mathcal{S}$. At that point one can take a decision $x_t \in \mathcal{X}$, which results in a reward or cost given by $C_t(S_t, x_t)$, and a transition to a new state S_{t+1} with probability $\mathbb{P}(S_{t+1}|S_t, x_t)$. So, the decision x_t affects the direct rewards and the transition probability towards the next state. The solution to this problem comes in the form of a policy $\pi \in \Pi$, a function $X^\pi(S_t)$ that returns a decision $x_t \in \mathcal{X}_t$ for all states. The best policy is found by solving the objective function of Equation (4), which minimizes the total costs over all timesteps, possibly discounted by a discount factor γ (Powell, 2011):

$$\min_{\pi \in \Pi} \mathbb{E}^\pi \left\{ \sum_{t=0}^{T} \gamma C_t(S_t, {X_t}^\pi(S_t)) \right\} \tag{4}$$

Where the minimization objective can be replaced by maximization depending on the context of the problem at hand. Solving this problem for every given path of states from t=0 to t=T is inefficient, and backwards recursion, a classic method used for MDPs, can be applied. In this method we start at t=T and introduce a function $V_T(S_T)$, that gives the value of being in state S_T. We then evaluate each action x_T to find $V_T(S_T)$, and then step back one timestep t=T-1 to find the optimal action x_t, where x_t has the largest one-period contribution (in terms of cost or profit) plus the value $V_{t+1}(S_{t+1})$ of landing in state S_{t+1}, possibly discounted by a factor γ. Thus, we must solve equation 2, where equation 3 gives the value of being in state S_t using the optimal action in that state. These equations are also known as the optimality equations.

$$x_t^*(S_t) = \arg\max_{x_t \in X_t} \big(C_t(S_t, x_t) + \gamma V_{t+1}(S_{t+1}) \big) \tag{5}$$

$$V_t(S_t) = C_t(S_t, x_t^*(S_t)) + \gamma V_{t+1}\left(S_{t+1}\big(S_t, x_t^*(S_t)\big) \right) \tag{6}$$

These equations assume that the transition from state S_t to S_{t+1} is deterministic, however in real-life problems (and with the problem this research concerns), this transition is stochastic. Equation 4 can be modified to account for this stochasticity:

$$V_t(S_t) = \max_{x_t \in X_t} \big(C_t(S_t, x_t) + \gamma \mathbb{E}\{ V_{t+1}\big(S_{t+1}(S_t, x_t, W_{t+1})\big) | S_t \} \big) \tag{7}$$

Where W_{t+1} is the random incoming information between t and t+1. This gives the expectation form of the Bellman equation. The standard form is slightly different in that it does not use the expectation, but $\sum \mathbb{P}(S_{t+1}|S_t, x_t)$, the sum of probabilities of transitioning from S_t to S_{t+1}, given x_t. The expectation form is more common however in ADP, so that will be used instead.

3.3.2. From Exact to Approximate Dynamic Programming

As discussed, stochastic MDPs can be solved by recursively computing Equation (7). However, practical problem instances quickly become too large to solve exactly, due to the three curses of dimensionality:

1. The state space $\mathcal{S}_t$ becomes too large to find $V_t(S_t)$ for all states $S \in \mathcal{S}_t$
2. The decision space $\mathcal{X}_t$ becomes too large to evaluate the optimal decisions for all states
3. The expectation of future costs becomes too large to evaluate due to a large information space $\mathcal{W}_t$

ADP tries to solve this problem by stepping forward in time. This first introduces two challenges: we need a way to sample random incoming information, and a way to make decisions. For the second problem we introduce $\bar{V}_t(S_t)$, which approximates the value of being in state S at time t. This starts

with an initial estimate $\bar{V}_t^0$. This initial estimate is improved by iterating over the problem multiple times, improving the estimate $\bar{V}_t(S_t)$ every iteration based on previous observations.

The Post-Decision State

A way to avoid approximating the expectation in Equation (7) and reduce the computational burden is introducing the Post-Decision State variable (Mes & Rivera, 2017). Consider the standard transition function $S_{t+1} = S^M(S_t, x_t, W_{t+1})$. This function can be broken down into 1) the effect of the decision, and 2) the effect of the external information. This allows breaking the transition function into two parts:

$$S_t^x = S^{M,x}(S_t, x_t) \tag{5}$$

$$S_{t+1} = S^{M,W}(S_t^x, W_{t+1}) \tag{6}$$

Where S_t^x is the post-decision state: the point immediately after making a decision, but before new information W_{t+1} has become available. This distinction has as a benefit that computing the expectation in equation 4 is not necessary anymore, as S_t^x is the point before any random information is available, and thus the transition from S_t to S_t^x is deterministic. An illustration of the transitions including the pre-decision state and post-decision state is given in Figure 11.

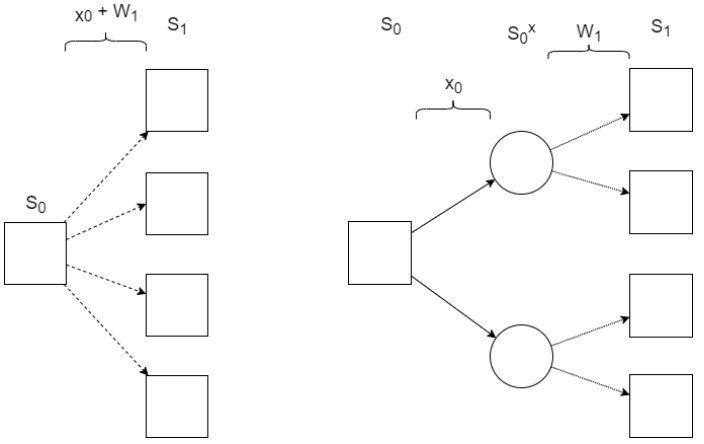


Figure 11 - Generic decision tree showing difference between MDP and ADP. Decision nodes (squares) and outcome nodes (circles). Solid lines are decisions, and dotted lines are random

With the introduction of the post-decision state, the optimality equations change. Just as $V_t(S_t)$ represents the value of state S_t, we introduce $V_t^x(S_t^x)$ as the value of the post-decision state S_t^x. The relation between these two form the new optimality equations:

$$V_t(S_t) = \max_{x_t \in X_t} \left(C_t(S_t, x_t) + \gamma V_t^x(S_t^x) \right) \tag{8}$$

$$V_t^x(S_t^x) = \mathbb{E}\{V_{t+1}(S_{t+1}) | S_t^x\} \tag{9}$$

The advantage in computation lies in the fact that Equation (8) is a deterministic optimization problem, which while still challenging, opens up a vast base of research for solving these problems (Powell, 2011). Equation (9) can be replaced by a Value Function Approximation (VFA) using knowledge learnt through previous iterations.

Iterations and Sampling

As we are using an approximation of the value of a state $\bar{V}_t(S_t^x)$ that has little to no knowledge of the actual value at the start, we have to run our algorithm iteratively and let the algorithm approach the value function over iterations. Assuming we have a suitable approximation $\bar{V}(S_t^x)$ for post-decision state S_t^x, and we are in iteration n at time t in state S_t^n, our optimization problem becomes:

$$\hat{v}_t^n = \max_{x_t \in X_t} \left(C_t(S_t^n, x_t^n) + \bar{V}_t^{x,n-1}\left(S^{M,x}(S_t^n, x_t^n)\right)\right) \quad (10)$$

$$\tilde{x}_t^n = \arg\max_{x_t^n \in X_t} \left(C(S_t^n, x_t^n) + \gamma \bar{V}_t^{x,n-1}\left(S^{M,x}(S_t^n, x_t^n)\right)\right) \quad (11)$$

After determining the optimal decision, the VFA of the post-decision state $S_{t-1}^{x,n}$ needs to be updated. This is possible, as the decision x_{t-1}^n puts you state $S_{t-1}^{x,n}$, after which random information $W_t(\omega^n)$ puts you in state S_t^n. This means that while $\hat{v}_t^n$ is a sample of the value of being in state S_t^n, it is also a sample of the value of the decision that put us in state $S_{t-1}^{x,n}$, so we can update our post-decision VFA using the formula:

$$\bar{V}_{t-1}^{x,n}(S_{t-1}^{x,n}) = (1 - \alpha_{n-1})\bar{V}_{t-1}^{x,n-1}(S_{t-1}^{x,n}) + \alpha_{n-1}\hat{v}_t^n \quad (12)$$

Where α_{n-1} is the updating factor at iteration n-1. The basic structure of the ADP algorithm then becomes:

0. $Initialization$
    a. $Initialize\ \bar{V}_t^{x,0}, t \in \mathcal{T}$
    b. $Set\ n = 1$
    c. $Initialize\ S_0^1$
1. $Choose\ a\ sample\ path\ \omega^n$
2. $For\ t = 0, 1, ,2, \dots, T$:
    a. $Solve\ equation\ (10)\ and\ (11)$
    b. $If\ t > 0, update\ \bar{V}_{t-1}^{x,n-1}\ using\ (12)$
    c. $Find\ the\ postdecision\ state$
3. $Increment\ n. if\ n \leq N, go\ to\ step\ 1$
4. $Return\ the\ value\ functions\ \bar{V}_t^N$

3.4. Aspects of ADP

In the previous section we discussed the basic structure of the ADP algorithm. There are however still many options available within the ADP algorithm with regards to the structure of the VFA, finite problems vs infinite problems, choice policies, and action enumeration vs linear programming. These aspects are discussed in this section.

3.4.1. VFA Options

In the introduction to ADP, the structure of the VFA has been handwaved. There are however many structures that can be chosen from. In problems with a large number of states (even with a post-decision state), the learning rate is still prohibitively low if the chosen VFA cannot visit every state often enough. A lookup table is an example of a VFA that suffers from this, while others can generalize across states to alleviate this problem, at the cost of accuracy for specific states. Common forms of VFA's are:

1. Lookup tables
2. Hierarchical state aggregation
3. Basis functions
4. Neural networks

Hierarchical state aggregation

A common strategy used in ADP that is similar to using simple lookup tables is hierarchical state aggregation. Using knowledge of characteristics of the state, several layers of aggregation are made, where each state belongs to one aggregated state in each layer (Mes & Rivera, 2017). When new knowledge about a state comes in, the value of each aggregated state it belongs to is also updated,

giving general knowledge about similar kinds of states. Estimates about a new state are then made using a weighted sum of the aggregated states it belongs to. This does introduces two new challenges: knowledge about the problem is required to make meaningful aggregation levels, and a tradeoff needs to be made between aggregation error and sampling error (Mes & Rivera, 2017).

Basis Function

This VFA uses the assumption that particular features or quantitative characteristics of a (post-decision) state can be used to approximate the value of that state. For each identified feature a basis function a ∈ A is created, of which the value is obtained from a post-decision state $S_t^{x,n}$ using a basis function $\varphi_a(S_t^{x,n})$. We assume the approximated next-stage value of a post-decision state can be expressed by a weighted linear combination of features (a linear approximation):

$$\bar{V}_t^{x,n}(S_t^{x,n}) = \sum_{a \in \mathcal{A}} \theta_a^n \phi_a(S_t^{x,n}) \tag{13}$$

This weight θ is updated recursively in each iteration n. While the basis function is linear, the values for features do not have to be. Selecting the right set of features for the basis function can be done in small instances by testing all combinations and using the set with the highest R^2 based on the goodness of fit between the values of the basis function and the optimal value functions. For larger instances, the right combination can be found by first running the ADP algorithm for a subset of states using different feature sets, then simulating the resulting policies for a couple of iterations, and repeating that for a number of replications (Powell, 2011).

Neural Networks

Neural networks are a relatively new method to use as a VFA. The concept of a neural network is inspired by that of a biological neural network: a layer of input, consisting of nodes containing a single datapoint, feeds values into one or multiple hidden intermediate layers via matrix multiplication and ends into the activation of one or multiple nodes in the output layer.

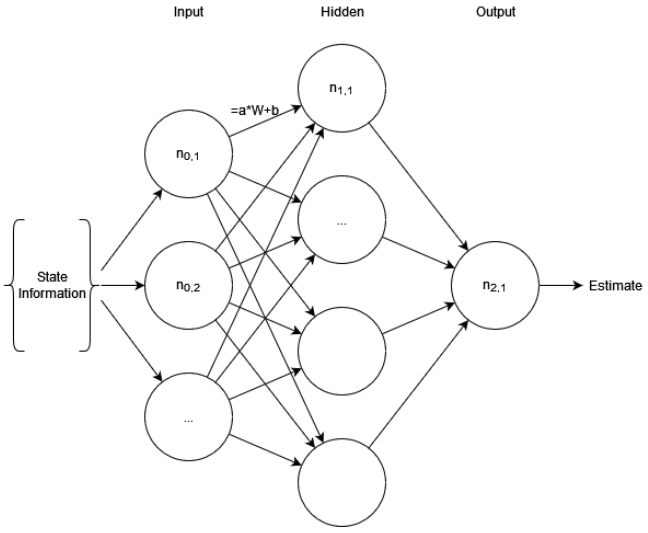


Figure 12 - Basic structure of a Neural Network

In the context of ADP, a NN is given feature functions and/or parts of the state $S_t \in \mathcal{S}$, and as output gives an approximation of the value of a state, similar to a basis feature function. Neural networks have as an advantage that they can approximate complex higher-order interactions of features without using non-linear functions (van Heeswijk & La Poutré, 2019), meaning they can still be applied in a Linear Programming setting (this concept is explored in Section 3.4.3). Neural Networks in ADP

have been tested on classic problems such as the Linear Sum Assignment Problem, Travelling Salesman Problem and Talent Scheduling Problem, providing considerable computation time reduction at the cost of reasonable performance loss when compared to popular heuristics for these problems (Xu et al., 2020). Of note is that while for NNs in Reinforcement Learning settings often the entire state is given as input without feature functions, in ADP this can lead to prohibitively large input layers due to the state space increasing exponentially. To solve this, domain knowledge about the ADP problem at hand is required to select state features that might be relevant, and supplement those with feature functions (Huré et al., 2021).

3.4.2. Finite problems vs Infinite problems

In Markov Decision Processes, problems can be either finite or infinite, meaning that they end at some point or are stationary. As a finite problem terminates at some point, transient input can be provided. This means that a model can be trained for one specific scenario instead of all scenarios that can be encountered in a problem setting. To illustrate using the Dynamic CRP; A finite problem setting can train for 3 ships arriving today, a general strike tomorrow, and 5 ships the day after. Meanwhile an infinite problem has to train for all reasonable combinations of ships within that 3-day period and may not be able to account for special events. An advantage infinite problems have over finite problems is however that they do not need to be re-trained; once trained, the model would be able to give solutions immediately, regardless of the situation. This can be an advantage if there is a small period between when information becomes available and when a new solution is required.

The finite vs infinite problem property also has an effect on the VFA to be used, as in a finite problem setting the VFA has to reflect that over time the optimal behavior shifts from long-term strategy to greedy behavior. In (hierarchical) lookup tables and basis functions, this shift can be accounted for by creating separate values for each timestep in a lookup table, or separate weights in case of a basis function. Neural networks have not been applied extensively in finite horizon optimization problems, so it is not possible to definitively say if a neural network can account for shifting priorities in finite optimization problems. They have however been applied in (Huré et al., 2021), and intuitively it should be possible to give the indicators of the finite planning horizon as input for a neural network, such as the remaining number of containers or time left in the planning horizon.

Single pass vs double pass

In case of a finite problem with separate weight sets for each timestep, costs incurred late in the problem horizon take a large number of iterations to be reflected in the VFA at early timesteps, in case of a simple forward pass when updating weights. To overcome this, a double pass approach can be used, consisting of a forward pass and a backwards pass. In the forward pass the decisions are simulated moving forward. Then in the backward pass the value functions moving backward are updated using the trajectory information.

3.5. Single-Stage Optimization Methods

In ADP, equation (11) needs to be solved at every stage, in every iteration of the training process. This means that it is important to find the optimal decision quickly in order to prevent a prohibitively long training time. In the case of the Dynamic CRP this presents a problem, as the decision space for any timestep is very large, even for the restricted CRP; every incoming container requires one 1 container to be placed, and every departing container requires $E[blockers]$ containers to be reshuffled. All batches consist on average for one half out of incoming containers, and for one half of outgoing containers. This means that the size of the decision space $|\mathcal{X}_t|$ becomes equal to Equation Error! Reference source not found..

$$|\mathcal{X}_t| = \left(E[as] * \frac{1}{2}(1 + E[blockers]) \right)^{|B_t|} \quad (14)$$

Where $E[as]$ is the expected number of available stacks, $E[blockers]$ the number of expected blockers of any particular container, and $|B_t|$ the size of batches. This means that the decision space grows exponentially with the size of batches, making it infeasible to enumerate over all available decisions $x_t \in \mathcal{X}_t$ for realistically sized terminals. If we assume that $E[as] = C$ with C being the number of stacks in the terminal, and $E[blockers] = \rho \frac{1}{2}(P-1)$ (i.e., the number amount of containers above any particular container for a terminal with P stacks and an occupation of ρ), the decision space for different sizes of terminals becomes:

C	P	ρ	|B|	E[as]	E[blockers]	|X|
8	4	0.6	2	8	0.9	5.78E+01
20	4	0.6	3	20	0.9	6.86E+03
50	6	0.6	5	50	1.5	9.54E+08
300	8	0.6	8	300	2.1	2.19E+21
300	8	0.6	15	300	2.1	1.03E+40

These decision space estimations are only rough indications but demonstrate that evaluating all possible actions in a single timestep is not possible for all but the smallest problem sizes. We thus need algorithms to efficiently find the optimal decision that minimizes not only the direct costs $C(S_t^n, x_t^n)$ but also the estimated future costs $\bar{V}_t^{x,n-1}\left(S^{M,x}(S_t^n, x_t^n)\right)$.

3.5.1. Pruning-Best-First Search

As there is little research done on using ADP on the CRP, not many methods have been tried for single-stage action optimization in the context of ADP. One thesis that has is from (Boschma, 2020), which uses an adaptation of the Pruning-Best-First-Search (PBFS) to find the best set of actions sequentially by allocating each container that needs to be allocated in in this timestep in order. In this way, the size of the decision space of a batch of containers doesn't scale exponentially with the size of a batch, but multiplicatively. The indicator that is used to determine where to allocate the containers is the Value Function Approximation, which is trained during the training of the ADP algorithm.

3.5.2. Linear Programming

An exact method for finding the optimal decision is Linear Programming. Linear programming allows large optimization problems to be solved effectively by formulating all aspects of the problem as a linear expression of decision variables. As equation (11) is an optimization problem, this too can be solved using linear programming. This does require all parts of the equation to be expressed in a linear program:

5. The direct costs
6. The VFA
7. The decision space
8. Transition of the current state to the post-decision state as a result of the actions taken
9. Basis functions for a VFA

This matter is problem-dependent for the direct costs, decision space and transition to the post-decision state. For the CRP, LPs have been formulated (Jin, 2020; Tang, 2015; Wan et al., 2009). Several options exist for the VFA: Lookup tables can be made compatible with LP by multiplying the presence of a certain state with the future cost associated with that state. Basis feature functions are linear combinations of basis functions, so if the basis functions are expressed linearly, a basis feature

function can also be used. Linear Neural Networks also exist, and have as a benefit that they can express non-linear interactions without using non-linear activation functions (van Heeswijk & La Poutré, 2019).

3.5.3. A*- algorithm

The A*-algorithm is a procedure generally used to find the shortest path between two points in a graph. In the CRP the method has been used to solve the unloading problem (Zhu et al., 2012), where the number of reshuffles is used as the distance to be minimized. Additionally, a variant on the A*-algorithm called the Adapted Pruning-Best-First Search (PBFS) has been used to solve large problem instances of the dynamic CRP. In this variant, the set of decisions in x_t is divided into a sequence of decisions to relocate incoming or reshuffled containers. The VFA learnt using ADP is used to estimate the best location to put each container. While this reduces the size of the decision space to $|\mathcal{X}_t| \leq (i,j)*|B_t|$, the performance of the algorithm is reduced significantly when compared to exact methods for small problem instances. An improvement on this tradeoff can be to reintroduce a priority queue on states to visit, and to find more promising decisions to make.

3.5.4. Branch & Bound

Another solution method for solving the optimization problem of finding $x_t^* \in \mathcal{X}_t$ is to use Branch & Bound. Multiple papers have used this method for solving the unloading CRP (Kim & Hong, 2006; Zhang et al., 2020). In this approach, the problem is divided into allocating individual containers, similarly to the search tree used by A*. For each point in the search tree, a LB and UB is produced, and points in the tree with a LB ≥ the global UB are discarded. As the worst-case complexity of this method is still equal to $|\mathcal{X}_t| = (i * j)^{|B_{t,c}|}$, a Beam Search method can be applied to the procedure. In this method, a maximum amount of viable nodes is allowed to be explored at any stage in the search tree. Unpromising nodes are pruned, even if their LB < the global UB, reducing the decision space to $|\mathcal{X}_t| \leq (i,j)*z*|B_t|$, where z is the max width of the beam search. In (Zhang et al., 2020), the decision to prune nodes is made using a random forest using a range of LBs/UBs as input. While training a random forest takes time, the VFA in our ADP approach can provide a metric to prune unpromising nodes on.

3.6. Conclusions

In conclusion, CRPs are categorized by a couple of characteristics. The first is the type of movement containers make; Unloading problems only have containers moving out of the terminal, loading problems have containers moving into the terminal, Premarshalling problems shuffle containers to minimize future unloading costs, and some combine Loading and Unloading or Loading and Premarshalling. Another characteristic is restrictions in the order in which containers are handled; containers can have a given sequence of handling or be an unordered set of containers. Lastly, different objectives can be distinguished. Minimizing the number of reshuffles or expected reshuffles in the presence of randomness is the most common objectives. Others are the minimization of transportation cost, usually given as the amount of moves and total distance travelled.

A variety of solution methods exist to solve CRPs. Exact methods are done on small terminals (<30 containers) and are mostly in the form of LPs. Larger Unloading problems are solved using B&B, Lookahead policies and various heuristics. Combined Loading/Unloading problems concern real-life problem instances and use general heuristic methods such as 2-stage heuristics or tree search with pruning. An overview of the discussed theses is given in Table 4.

The Dynamic CRP can also be formulated as a Markov Decision Problem (MDP). Formulating the problem this way makes it possible to incorporate randomness into a CRP, by having information become available between stages. Large instances of MDPs can be solved using Approximate Dynamic

Programming (ADP). This method makes it possible to account for the exploding state-space that happens due to randomness and larger problem instances. It does this by separating the transition from state S_t to S_{t+1} by introducing a post-decision state S_t^a; the state after which a decision is made, but before random information comes in. This distinction allows effective computation of the optimal decision by avoiding the need to calculate expectations.

4. Model Design

In this chapter, we formally introduce the problem and propose an Approximate Dynamic Programming (ADP) approach to solve it. We first introduce the problem that will be modelled. Afterwards, it is formalized into a Markov Decision Process (MDP) using a design framework for MDP's. Afterwards we discuss the design of the Value Function Approximation (VFA). Finally, some methods to prune the decision space of the ADP are discussed.

4.1. Introduction of problem

The setting of the problem that is solved in this research is a terminal with c stacks and p tiers, where each place (c, p) can hold one container. At the start of the problem a number of places are occupied with containers, and over a given planning horizon a number of containers arrive and depart in batches. The sequence of batches is known, the order of containers arriving/departing within these batches is not. Containers can have one of four types: 20HV, 40HV, 20RF, 40RF. This is further explained in Section 4.2, assumption 2. For each stack it is pre-determined which types of containers can be placed there, so that containers of different sizes are not stacked on top of each other.

When a batch of containers arrives, the order of containers within that batch is revealed, and these containers must be handled in the given order. Containers can be moved out of the terminal towards an entrance, into the terminal from an entrance, and internally, with any internal move being a reshuffle. Any container that blocks containers moving out of the terminal need to be reshuffled, but this is only allowed for containers blocking the container that currently needs to leave the terminal. The goal is to minimize costs made during the handling of all batches in the planning horizon, which consist of the number of reshuffles, the distance travelled by all containers, and the number of times a container is put on a stack it was not allowed, see Section 4.2, assumption 11. After all containers in a batch are either moved in or out of the terminal we arrive at the post-decision state. To transition from the post-decision state to the pre-decision state of the next stage we process random information, which in this context arrives in the form of revealing the order in which the next batch of containers needs to be handled.

4.2. Assumptions

With the problem introduced, we formalize all assumptions made in the model.

1. The objective of this model is to minimize transportation costs, defined as a weighted sum of the amount of reshuffles, distance travelled by all containers and violation of soft constraints

The objective of this model is to minimize transportation costs. In CRPs transportation costs are commonly defined as the weighted sum of the number of reshuffles and distance travelled by all containers. As this model also contains a soft constraint, we add this to the cost function. How the cost for reshuffles, distance and soft constraint violations are defined are explained in assumptions 9, 10 and 11 respectively.

2. The stacks, entrances and their locations are fixed

As described in Section 2.4, this research is focused on optimizing the logistics of container handling at an operation level. This means that long-term planning decisions such as the layout of the terminal, allocation of ships to berths and equipment are assumed not in the scope of this problem.

3. There are 4 types of containers present at the terminal: 20HV, 40HV, 20RF and 40RF

During the analysis of the types of containers that enter the terminals of clients of Cofano 73% of containers were 40HV/45HV or minor variants of those, 25% are 20HV or minor variants, 1% are 20RF containers, and 1% are other types, as mentioned in Section 2.2. In order to take multiple lengths and types of containers into account while not adding exceptions for container types that make up <0.1% of containers, we assume that these 4 types make up the containers arriving at the terminal.

4. Containers of different lengths are kept separated, and terminal stacks are designated to one length

While in theory 20ft and 40ft containers can be stacked on top of each other, with two 20ft containers supporting one 40ft container and vice versa, in practice the stacking of different lengths of containers is avoided as much as possible, as it complicates stacking. For this reason, containers are only stacked on containers with the same length. Terminal stacks are designated to a specific container length in advance, as stack allocation to container types is left out of the scope to reduce complexity.

5. Containers are handled one-by-one

In a terminal multiple containers from different areas of a terminal can be moved at the same time, but this research assumes containers are handled one by one. As long as it is accounted for that moves are done in parallel, this should not reduce the practical applicability of the algorithm, and this assumption reduces complexity.

6. Containers arrive and depart in batches

Containers can arrive in a terminal via ship, train or truck. Ships and trains contain a large number of containers that are released at the same time, which allows them to be grouped. While trucks do not necessarily follow this pattern, this assumption allows the incorporation of partial knowledge about the arrival and departure sequence of containers.

7. The sequence of batches is known beforehand, but the order of containers within a batch is known when that batch is next in line

At a container terminal the containers that are scheduled to arrive and depart is known in advance but delays as discussed in Section 2.5 mean that the exact sequence of arrivals and departures is not completely known. In addition, containers arriving by ship are loaded and unloaded with a stowage plan, which is not released beforehand. This process of partial information is modelled by assuming a fixed sequence of batches of containers arriving and departing from the terminal, with the order within a batch only being released when that batch is the next one to be handled.

8. A container may only be reshuffled when it is blocking the current target container

As mentioned in Section 1.4, restriction 4, the CRP has a restricted and unrestricted variant. The unrestricted variant is in small cases shown to result in lower amounts of reshuffles, but with a decision space that is multiple factors larger than the restricted variant. As this problem already has a large decision space, we use the restricted CRP, meaning that only containers blocking the currently leaving container are allowed to be reshuffled.

9. The cost of a reshuffle is assumed to be a flat amount

The goal of this research is to reduce the total time spent moving containers, which can be expressed as the sum of reshuffles plus the distance of all reshuffles, inward and outward moves. As equipment is not accounted for, we assume a fixed cost for each reshuffle. We use a value of 2 as the cost for

reshuffling a container, derived from an assumed 1 minute to lift a container, and 1 minute to lower a container, for 1 minutes of work total. We base these values on (Li et al., 2019).

10. The distance cost for a movement from point a to point b is assumed to be fixed

For the same reason described in assumption 9, we calculate the distance cost from point a to point b in advance and assume it to be fixed. We assume the cost of distance travelled to be equal to 0.006 per travelled meter, which is the amount of minutes per travelled meter, assuming a speed of 10km/h (Li et al., 2019).

11. The restriction of container types to a pre-determined stacks is used as a soft constraint

For each stack it is pre-determined which types of containers can be placed there. We formulate this restriction as a soft constraint however, as it would otherwise be possible to get stuck in a deadlock position, where due to the allocation of containers no spots are left for an incoming container of type y. We define the cost of violating this soft constraint as equal to the cost of 4 reshuffles, so a cost of 8.

4.3. Markov Decision Process model

In this Section the problem that needs to be solved is formalized using a framework a MDP, which can be solved using ADP. It requires the following parts, mentioned in chapter 5 of (Powell, 2011).

- Sets and constants
- State Variables
- Decision Variables
- Exogenous information processes
- Transition function
- Objective Function

This section describes the set up of these parts according to the framework provided. First however, we define the stage of the Markov Decision Process (MDP), as this is a standard part of any MDP and will not be covered by the framework. In our model, each stage is a timestep where a batch of containers is handled. This batch is a combination of outgoing and incoming containers that are scheduled to arrive/depart in a certain timeframe (e.g., all the containers scheduled to arrive or depart in the next 30 mins). The sequence of batches is known, but the order of containers arriving/departing within each batch is not. This information is only given when the batch itself is due to be handled. These timesteps are denoted by t.

4.3.1. Sets and constants

The MDP contains a number of sets and parameters.

Sets

- Timestep $t \in \{0 \dots T\}$
  - The stage of the MDP. In each stage one batch of containers needs to be handled.
- Location index $c \in \{1 \dots C, C+1 \dots C+R\}$
  - The locations consist of c stacks in the terminal and r entrances of a terminal. If a terminal has 10 stacks and 2 entrances, locations 0-9 are the 10 stacks, and locations 10 and 11 are the two entrances.
- Tier index $p \in \{0 \dots P\}$
  - Tier p indicates the vertical location of a container within a stack c. The combination of (c, p) forms a location where one container can be placed.

- Container index $i \in \{0 \dots I\}$

Parameters

- Values of the transportation-cost function $\alpha_1, \alpha_2, \alpha_3$
  - α_1 refers to the weight of reshuffling costs, with $\alpha_1 = 2$
  - α_2 refers to the weight of distance travelled, with $\alpha_2 = 0.06$
  - α_3 refers to the weight of violating the soft constraint mentioned in Section 4.1, with $\alpha_3 = 8$
- Information about container i:
  - Arrival batch $a_i \in \{0 \dots T\ if\ container\ arrives\ during\ the\ planning\ horizon, or\ undefined\ if\ container\ is\ already\ present\ at\ start\ of\ the\ problem\}$
    - Some containers are already present at the initial state of the terminal, meaning they do not have an arrival batch
  - Departure batch: $d_i \in \{0 \dots T\}$
  - Length/type $y_i \in \{0\colon 20HV, 1\colon 40HV, 2\colon 20RF, 3\colon 40RF\}$
  - Entrance $n_i \in \{C+1, \dots, C+R\}$
  - Exit $e_i \in \{C+1, C+R\}$
- Distance between location c and e: $d_{ce} \geq 0$
- Boolean indicating that container type y is allowed on stack c. $w_{cy} \in \{0, 1\}$
  - There are 4 types of containers in this problem: 20HV, 40HV, 20RF, 40RF. In this problem the assumption is that 20/40 containers do not mix, and both lengths have their allocated stacks. Additionally, a small fraction of stacks for both lengths can contain reefer containers. On these stacks it is also allowed to place HV containers. If containers are allocated on a stack they are not allowed on this results in a soft-constraint penalty.
- Initial position of container in the terminal $(c_i, p_i) \in \{(0 \dots C, 0 \dots P)$ if container is present at the start of the problem, or undefined if container arrives later$\}$

4.3.2. State Variables

The state variables consist of all the changeable information needed at time t to model the system from time t onward. For our problem we define the following variables of the state:

Variables

- Information about container i:
  - Order within batch when arriving: $b_i \in \{\mathbb{R}\ if\ a_i \leq t, -otherwise\}$
  - Order within batch when departing: $f_i \in \{\mathbb{R}\ if\ a_i \leq t, -otherwise\}$
  - Location of container: $c_i \in \{0 \dots C+R\}$
  - Tier of container: $p_i \in \{0 \dots P\ if\ container\ is\ in\ terminal, -otherwise\}$

To illustrate: Let's take the example of Table 5, illustrated in Figure 13. This is a state at t=1, so the order-within-batch of all incoming and outgoing containers up until t=1 is known. From columns a and d we can see that in $t = 1$, first container 1 needs to exit, then container 4 enters, then container 5 enters, and lastly container 3 leaves. This is shown in Table 4.

Table 5 - Example of a possible state at t=1

ID	a	b	d	f	y	(c, p)	n	e
0	-	-	3	-	1	(3, 1)	4	4
1	-	-	1	0	0	(2, 0)	4	5
2	0	1	2	-	1	(0, 0)	5	4
3	0	0	1	3	0	(3, 0)	4	5
4	1	1	3	-	2	(5, -)	5	5
5	1	2	5	-	1	(5, -)	5	4
6	2	-	4	-	0	(4, -)	4	4

Table 6 - The arrivals and departures from Table 3 given as a sequence of batches.

Timestep t	Required actions (a & d)	Order within batch (b & f)
0	3 IN	0
	2 IN	1
1	1 OUT	0
	4 IN	1
	5 IN	2
	3 OUT	3
2	6 IN	?
	2 OUT	?
…	…	…

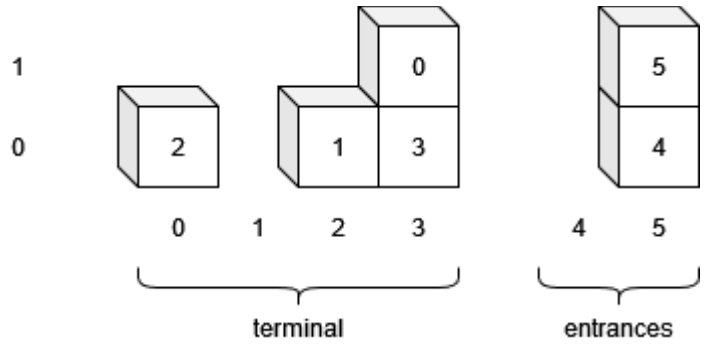


Figure 13 - Illustration of what the state from Table 3 would look like

4.3.3. Decision Variables

The decision space $\mathcal{X}_t$ contains all possible combinations of every decision that can be made at time t. The decisions that make up this decision space are highly problem context dependent. In the problem setup, at every timestep t a batch of incoming/outgoing containers needs to be handled in a given order. We use the restricted version of the CRP, which means that only containers that are blocking a target (outgoing) container may be reshuffled. The decisions that thus need to be made are:

- For every incoming container:
  - Which location (c, p) to put the container
- For every outgoing container:
  - For every blocking container:
    - Which location (c, p) to reshuffle the container to

Of course, containers cannot be placed on any (c, p) position. Containers can only be stacked on top of an existing stack, so for each stack c only tier p is available for which p = stack height + 1. If the stack

height is at the height limit, no container can be placed there at all. Additionally, containers should only be placed on stacks where their container type is allowed. It is chosen to leave this as a soft constraint with penalty α_3, in order to prevent deadlocks in the case of a nearly full terminal. In that case, the container can be placed on a non-allowed stack anyway, and a large cost is incurred.

To come back to the example given in Table 3 and Figure 13: First container 1 needs to exit, then container 4 enters, then container 5 enters, and lastly container 3 leaves. The decision space is thus:

- Nothing for container 1 (no reshuffles required)
- Where to put container 4
- Where to put container 5
- Where to reshuffle all containers that block container 3 (so container 0, and container 4 and 5 also possibly, if they were put on stack 3).

The size of the decision space for a single timestep can grow very fast; every incoming container requires one 1 container to be placed, and every departing container requires $E[blockers]$ containers to be reshuffled. All batches consist on average for one half out of incoming containers, and for one half out of outgoing containers. This means that the expected size of the decision space $|\mathcal{X}_t|$ becomes equal to Equation Error! Reference source not found., with an upper bound equal to Equation (16.

$$|\mathcal{X}_t| = \left(E[as] * \frac{1}{2}(1 + E[blockers])\right)^{|B_t|} \tag{15}$$

$$UB[|\mathcal{X}_t|] = \big((C-1) * (P-1)\big)^{|B_t|} \tag{16}$$

Where $E[as]$ is the expected amount of available stacks, $E[blockers]$ the amount of expected blockers of any particular container, $|B_t|$ the size of batches, and C and P the number of stacks and tiers. This means that the decision space grows exponentially with the size of batches, making it infeasible to enumerate over all available decisions $x_t \in \mathcal{X}_t$ for realistically sized terminals. A different method for finding x_t needs to be found, which is discussed in Section 4.5.

4.3.4. Transition function

The transition function takes in decision x_t to get from pre-decision state S_t to post-decision state S_t^x, see Equation (5). In this problem context, processing the decision x_t means assigning incoming containers to their location, reshuffling blocking containers and removing outgoing containers. New information about incoming batches and updates about the retrieval order is not incorporated, as that is part of the exogenous information W_t.

In our example, if we send container 1 to entrance 4, container 4 to stack 2, container 5 to stack 0, and then reshuffle container 0 to stack 1 and lastly send container 3 to entrance 5, we get the post-decision state in Table 7 and Figure 14:

Table 7 - The post-decision state S_t^x

ID	a	b	d	f	y	(c, p)	n	e
0	0	3	3	-	1	(1, 0)	4	4
1	0	1	1	0	0	(4, -)	4	5
2	0	2	2	-	1	(0, 0)	5	4
3	0	0	1	3	0	(5, -)	4	5
4	1	1	3	-	2	(2, 0)	5	5
5	1	2	5	-	1	(0, 1)	5	4
6	2	-	4	-	0	(4, -)	4	4

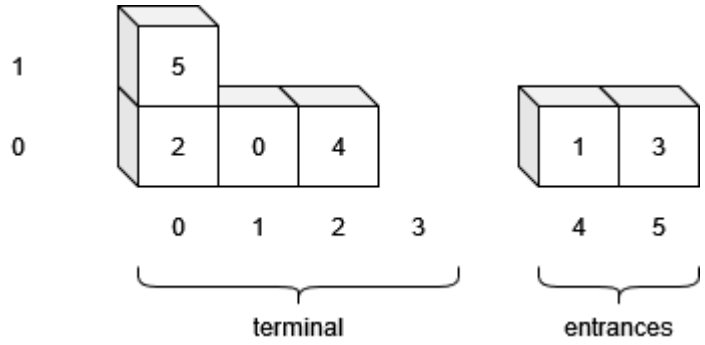


Figure 14 - Illustration of what post-decision state S_1^x looks like

4.3.5. Exogenous information processes

The flow of exogenous information is an important dimension in ADP, as it brings the model from state S_t^x to S_{t+1}. While the possibilities of random incoming information is often (assumed to be) in the form of random distributions, ADP works using sampling with realizations of this random information. We use W_t to denote the general exogenous information that becomes available at time t, and $\omega \in \Omega$ for realizations of this exogenous information. In our problem the exogenous information is the random information that becomes available after handling the container batch at timestep $t-1$:

- The definitive order of containers arriving/departing in, realized by generating the sequence of incoming/outgoing containers in that batch according to assumption 7.

In our example, the exogenous information in this case is that container 6 arrives first, then container 2 leaves afterwards, shown in Table 8.

Table 8 - State S_2 after the exogenous information

ID	a	b	d	f	y	(c, p)	n	e
0	0	3	3	-	1	(1, 0)	4	4
1	0	1	1	0	0	(4, -)	4	5
2	0	2	2	1	1	(0, 0)	5	4
3	0	0	1	3	0	(5, -)	4	5
4	1	1	3	-	2	(2, 0)	5	5
5	1	2	5	-	1	(0, 1)	5	4
6	2	0	4	-	0	(4, -)	4	4

4.3.6. Objective Function

The objective function is the combination of the direct contributions to the objective function plus the estimated value of the post-decision state one arrives in due to the state S_t and action x_t.

$$z = \min_{x_t \in X_t} \left(C_t(S_t, x_t) + \gamma V_t^x(S_t^x) \right)$$

The contribution function is the first part of this equation, and as mentioned in Section 3.3.2, is a weighted sum of the amount of reshuffles (#RS) and distance (D) travelled by containers in timestep t. Additionally, a penalty (#P) is given for putting containers where they aren't allowed:

$$C_t(S_t, x_t) = \alpha_1(\#RS) + \alpha_2(D) + \alpha_3(\#P)$$

To calculate the distance travelled by containers, a distance matrix is used. The distance this matrix represents can be the Euclidian distance, shortest direct path between locations, or some other arbitrary measure. The only limitation is that this distance is fixed.

The weights for this composite contribution function are to be decided. One approach is to use weights such that x_1 is equal to the average picking up + putting down time of a container (Li et al., 2019), and x_2 is equal to the average speed of horizontal movement within the terminal, as this would cause a minimization of total time spent handling containers.

The remaining part of the objective function is the VFA, which considers the value of future states. Section 4.4 will discuss the design of this VFA.

4.4. Value Function Approximation Design

Section 4.1 described the problem that needs to be solved, and Section 4.3.5 mentions the objective function that is optimized for. This objective function contains a 'Value Function Approximation', that approximates the expected future costs. This section discusses the methods this research uses to approximate this value function approximation, which are a Basis Feature Function and a Neural Network, which both use features as input.

4.4.1. Basis Feature Function

A basis feature function consists of a set of features that indicate the quality of a state, combined into a weighted sum to indicate the value of a state, see Equation (13). This method of approximation is powerful if it works, as the linear relation between the value of features and value of the state allows the weights of the basis function to be fitted with a relatively low amount of observations (Si et al., n.d.). This method however is only applicable if features can be found that have a somewhat linear relation between the value of a feature and the value of a state, and thus extensive knowledge about the problem domain is required. In Section 4.4.3 the list of features used in this research are listed.

As mentioned in Section 3.4.2, the application of a basis feature function is different dependent on whether the problem is finite or infinite. The problem in this research is finite, meaning that for each timestep in this problem a separate set of weights for each of the features is used. The updating of the weights is done using 'recursive least squares' updating, which has the advantage of converging faster than other methods such as stochastic gradient descent (Boschma, 2020). Recursive least squares works using the following set of formulas:

$$\alpha^n = 1 - \frac{\delta}{n+1}$$

$$\gamma_n = \alpha^n + \left(\phi^n(S_t)\right)^T B^{n-1} \phi^n(S_t)$$

$$B^n = \frac{1}{a^n}\left(B^{n-1} - \frac{1}{\gamma^n}\left(B^{n-1}\phi^n(S_t)\left(\phi^n(S_t)\right)^T B^{n-1} \right) \right)$$

$$H_n = \frac{1}{\gamma_n} * B^{n-1}$$

$$u^n = H^n \phi^n(S_t)(\bar{V}_t^{n-1}(S_t) - \hat{v}_t^n)$$

$$\theta_t^n = \theta_t^{n-1} - u^n$$

Where δ is a variable between [0, 1), and B^n a matrix of $a * a$, where a is the number of features. B^0 is initialized with $\rho\mathbb{I}$, where ρ is a small constant (0.1 is used in this research).

4.4.2. Neural Network

Neural networks are a powerful and general method for approximation, as they combine the quality of being able to approximate any non-linear function while being differentiable at all points, allowing for directed optimization. The drawback of being able to approximate non-linear functions is that this method is not able to fit a function with the low amount of observations a basis function can. As neural networks allow features to interact with each other through the multiple layers a network has, the interactions between features can be captured (e.g., the importance of the number of unordered stacks decreases as the amount of non-empty stacks increases). This means that features used as input for the neural network do not need to directly correlate with the value of a state. In Reinforcement Learning it is common to include the entire information on the state, or large parts of it, to fit the value function. However, in order to keep the Neural Network comparable to the Basis Feature Function both use features as input, with as difference that the Neural Network will use 1 set of weights for all timesteps of the problem, with 'time since start' and 'time until end' as extra features.

Updating is done using the Adam algorithm, which is a variation on the Stochastic Gradient Descent algorithm that adapts the parameter learning rates of weights based on the average first moment and second moment of gradients, speeding up convergence to the optimal values (Ruder, 2016). This research uses the Huber loss function, which is a loss function that is equal to the Mean Square Error (MSE) for any particular error, up until a threshold value γ, at which point the gradient for loss does not increase anymore. This loss function strikes a balance between the MSE, which tends to put a lot of emphasis on outliers in data (or states in the case of this research), and Mean Absolute Error, which has a constant gradient, which can diminish convergence speed.

4.4.3. Features

In both the Basis Feature Function and Neural Network, features form the input with which the value (or in this problem, the future expected costs) of the post-decision state is estimated. In this section we list all the features that this research uses. As a selection criterium for this list, we require that a feature 1) plausibly gives an indication of future costs, and 2) that it is actionable; it must indicate a value that can be changed by choosing different actions. For instance, 'terminal occupation' or 'batch size of timestep $t + y$' is not featured in this list, as these features are a given in the problem and cannot be changed by changing the decision policy.

Expected Blocking Lower Bound (EBLB)

This feature is a simplification of the Expected Blocking Lower Bound (Galle, 2018), also used in the ADP method in (Boschma, 2020), which takes the possibility of containers departing in the same stack into account. Instead of calculating the exact probability that a container blocks one below it if it departs in the same stack, we assume the possibility is always 0.5. This makes this feature computationally simple. An example of this feature is given in Figure 15.

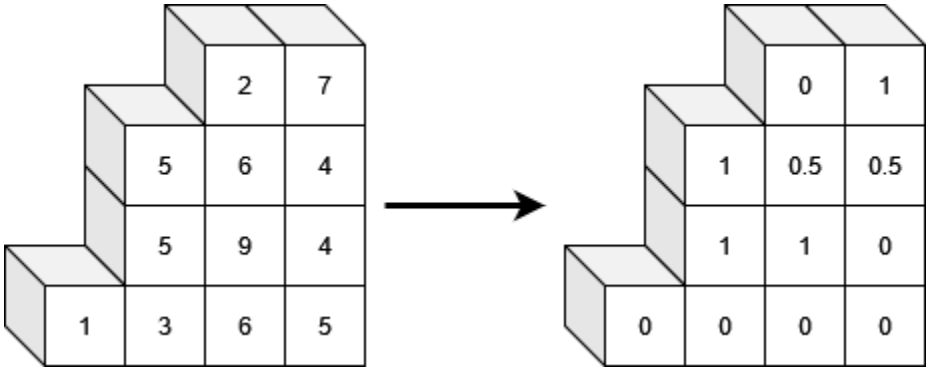


Figure 15 - Example of the EBLB. Container batch labels are shown on the left, and the blocking value it results in is given on the right

Expanded EBLB (E-EBLB)

This feature is an adaptation of the LB2 from (Zhu et al., 2012), and adapts it to the Dynamic CRP. The principle of the lower bound revolves around adding more reshuffles if no stacks are readily available where the blocking container can be located. It works using the following algorithm:

1. Note for every stack in the terminal the earliest departing container
2. For all containers, determine if a (possible) block happens for container x, resulting in an initial blocking value $b_x^{init} = 0.5 \; or \; 1$
3. If so, compare the earliest departing container in all other stacks to the departure batch of container x:
    a. If there is any stack where the earliest departing container departs later, the additional blocking value $b_x^{add} = 0$
    b. Else, if there is any stack where the earliest departing container departs at the same time, $b_x^{add} = 0.5$
    c. Else, the $b_x^{add} = 1$
4. The final blocking value $b_x^{tot} = b_x^{init} * (1 + b_x^{add})$
5. The value for $E\text{-}EBLB = \sum_x b_x^{tot}$

Figure 16 shows an example of this procedure.

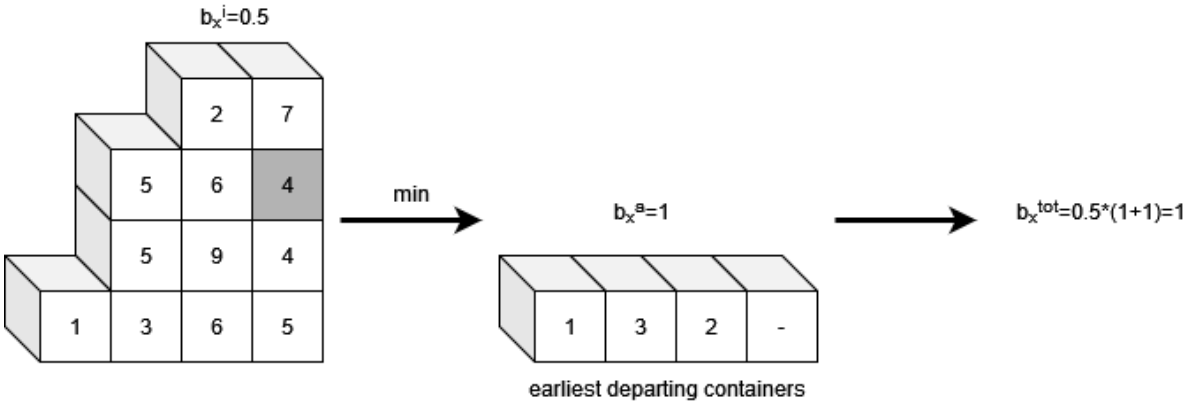


Figure 16 - Example of the expanded blocking lower bound for the container in grey

Dynamic Lookahead Lower Bound (DLLB)

The LB3 from (Zhu et al., 2012) is written for the unloading problem, but can be adjusted to apply to the dynamic CRP:

1. For all containers, denote their initial blocking value $b_x^{init} = 0.5 \; or \; 1$
2. Then, remove all containers from the terminal whose departure is before the time this container is reshuffled. This time is determined by the earliest departing container below container x in the stack.
3. If so, compare the earliest departing container in all other stacks to the departure batch of container x:
    a. If there is any stack where the earliest departing container departs later, the additional blocking value $b_x^{add} = 0$
    b. Else, if there is any stack where the earliest departing container departs at the same time, $b_x^{add} = 0.5$
    c. Else, the $b_x^{add} = 1$
4. The final blocking value $b_x^{tot} = b_x^{init} * (1 + b_x^{add})$
5. The value for $LA\text{-}EBLB = \sum_x b_x^{tot}$

Figure 17 shows an example of this procedure.

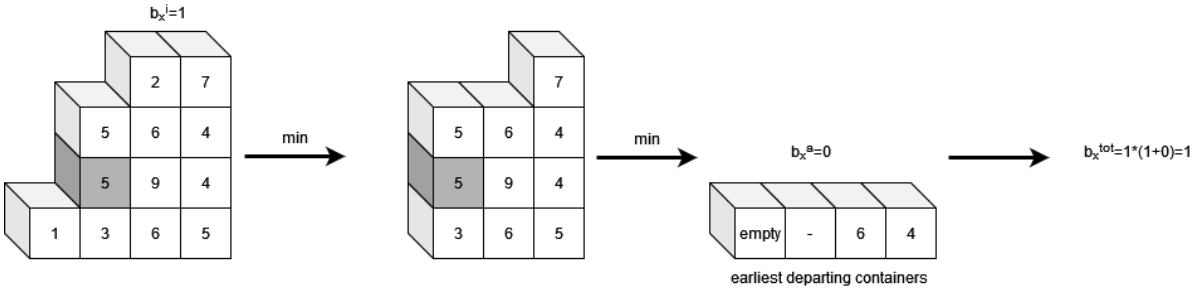


Figure 17 - Example of the lookahead blocking lower bound for the container in grey

Travel Distance Lower Bound (TDLB)

As the objective function of this CRP not only concerns the amount of reshuffles but also the distance travelled per container, a lower bound for this objective also needs to be found. A simple lower bound can be found by assuming no additional reshuffles take place, and all containers are moved straight from their origin to destination. This is done for all containers in the terminal.

Blocking Degree (BD)

This feature calculates for each container x the difference between its departure time and the earliest departure time of containers below it, if any container below it departs earlier than container x. This indicates not only the amount of blocking containers, but also the amount by which containers are blocking. As generally it is assumed that if a container has to be blocking, it should preferably be close in retrieval order to the container it is blocking (Caserta et al., 2012), this indicator can give additional information on the quality of a state compared to the EBLB.

Unordered Stacks (US)

This measure counts the number of stacks that (may) require a reshuffle. This measure is similar to the BLB, but only counts stacks instead of containers. It is also used in (Boschma, 2020).

Semi-Ordered Stacks (SOS)

This feature is a variation on Unordered Stacks that counts the amount of stacks that do not contain containers that block earlier departing containers but does contain containers that depart in the same timestep.

Batch Label Difference (BLD)

The BLD calculates for each container in the terminal the absolute difference in batches between the container below it and the container itself. A BLD near zero indicates a stack of which batch labels lay close to each other, which is assumed to be beneficial. The sum of BLDs of all containers gives the BLD feature value for the entire terminal. It is also used in (Boschma, 2020).

Average Stack Height (ASH)

The ASH calculates the average stack height of all non-empty stacks. It is also used in (Boschma, 2020).

Squared Stack Height (SSH)

This feature as an adjustment of the ASH feature, by instead calculating the sum of the squares of each stack height. Terminals with a more evenly distributed stack height will have a lower squared sum.

Non-Empty Stacks (NES)

This feature gives the sum of the nr of stacks that contain at least one container.

Used Space Percentage (USP)

The USP takes a container type as argument, and for that type returns the amount of used spaces where container type y is allowed divided by the total amount of spaces for container type y available. Having a high percentage of used spaces for a certain type of container can lead to a situation where a container has to be placed on a spot where it is not allowed, violating the soft constraint mentioned in Section 4.1. This results in high costs by having to incur the penalty associated with it.

Highest Used Space Percentage (HUSP)

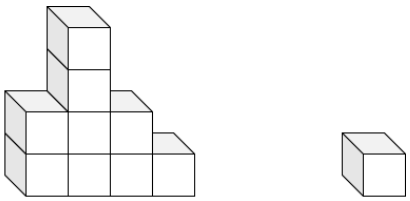
This feature calculates the USP for each container type in the terminal and returns the highest value.

Non-Ideal Stacks (NIS)

This feature takes as input an integer y, and for all incoming containers x in batch $t + y$, and returns the sum of stacks that meet the following criteria:

4. The container is allowed on the stack
5. The stack is not full
6. The stack doesn't contain containers that depart earlier than container x

These criteria are calculated over the terminal in the current position, so containers that might arrive before container y are not accounted for. The value of the feature is the sum of this count over all incoming containers in batch $t + y$. Figure 18 shows an example; stacks 0, 1 and 2 are not ideal, stack 3 is.



stack ID	0	1	2	3
allowed types	(0, 1)	(0, 2)	(2, 3)	(2)
full	n	y	n	n
earliest departing container	5	7	2	6
ideal	n	n	n	y

container type	2
departure batch	4

Figure 18 - Illustration of the calculation of the Non-Ideal Stacks feature

Non-Ideal Containers (NIC)

This feature does the same as the NIS feature, but instead of summing the amount of non-ideal stacks, it tallies all the containers for which there are 0 stacks that meet the criteria.

Future Incoming Costs (FIC)

This feature takes in an argument y. For each container x that arrives at the terminal in timestep $t + y$, the move costs are calculated for each viable destination stack, consisting of:

7. The distance multiplied by objective weight α_1
8. The reshuffle costs multiplied by objective weight α_2
    - The reshuffle costs are 1 if the earliest departing container departs earlier than container x, and 0 otherwise
9. Wrong stack penalty costs multiplied by objective weight α_3
    - A wrong stack penalty is applied if container x is not allowed on that stack

The destination with the lowest cost is chosen. The value of the feature is the lowest cost available for each incoming container in batch $t + y$. Of note is that containers are not actually placed in the terminal. Instead, the FIC value for each container is calculated based on the terminal as-is. This is because the order of arrivals of containers not known when batch $t + y$ is handled, and thus not possible to account for without testing all permutations of batch $t + y$, whose order is not yet known.

Future Outgoing Costs (FOC)

This feature does the same as FIC, except using outgoing costs. The costs consist of 1) the distance between the current location of the outgoing container and its exit multiplied by α_1 and 2) the amount of containers on top of the outgoing container multiplied by α_2.

Minimum Wrong-Stack Penalty (MWSP)

This feature takes in an argument y. It then calculates the following:

0. Check the amount of spots available for each container type
1. Set $MWSP$ to 0
2. For $t + 1$ until $t + y$:
    a. For each incoming container:
        i. If there is an available spot for that container type, use it. Decrease available spots for that type by 1.
        ii. Else, $MWSP \mathrel{+}= 1$, and add container to list of wrongly stacked containers
    b. For each outgoing container:
        i. If container is wrongly stacked, remove from list.
        ii. Else, increase available spots for that container type by 1.

The value for this feature is the value for MWSP after this procedure.

Min-Max Value (MMV)

The MMV is designed such that if it was the feature by which containers were allocated, it would always allocate according to the Min-Max Heuristic. It does this by using the following procedure:

1. $MMV = 0$
   Calculate b_{max} = latest departing container in terminal
   Calculate b_{min} = earliest departing container in terminal
2. For all containers in terminal:
    a. $b_{container}$ = Departure batch for container
    b. If container is on the ground floor:
        i. $MMV \mathrel{+}= b_{max} - b_{container}$
    c. Else, if it blocks any containers below it:
        i. b_{lowest} = earliest departure batch for the containers below
        ii. $MMV \mathrel{+}= b_{max} - b_{min} + b_{container} - b_{lowest}$
    d. Else, if it does not block any containers below it:
        i. b_{lowest} = earliest departure batch for the containers below
        ii. $MMV \mathrel{+}= b_{container} - b_{lowest}$

The value for this feature is the value of MMV after this procedure.

Squared (SQ)

Features for a Basis Feature Function are most useful if there is a linear positive relation between future costs and the value of the feature [source]. For some features however, this may not be the case; the USP feature likely does not correlate linearly with increased costs, but is largely irrelevant until the value approaches 1, which is when the costs start to increase rapidly. For this reason, features can be squared in order to make their value increase more than linearly with future costs.

Square-Root (SQRT)

Just as some feature values only correlate with increased costs when their values get large, some features can only indicate a lack of increased future costs when their value approaches zero, while at higher values the correlation disappears. This feature takes the square root of features, so that such a feature correlates more linearly with increased future costs.

Reshuffle Index Heuristic (RIH)

This heuristic, discussed in Section 3.2.4, produces an approximation of future costs. For the Dynamic CRP, this feature is modified by assuming no more containers enter the terminal, and all containers are removed from the terminal in order. In the case of ties, the container with the lowest ID is assumed to leave first. The value of this heuristic is the sum of 1) distance costs, 2) reshuffle costs and 3) wrong-stack penalty costs incurred during the unloading of all containers in the terminal. The way containers are relocated is using the following:

1. For all possible destinations for container x:
    a. Get the reshuffle-index, which is the amount of reshuffles necessary to access the earliest departing container *if* container x is placed there.
2. Place the container in the location that has the lowest reshuffle index
    a. In case of a tie, place the container in the closest destination

Min-Max-Heuristic (MMH)

This heuristic also produces an approximation of future costs similar to the RIH but uses the Min-Max heuristic instead. This heuristic works using the following:

1. Get all possible destinations for container x, and find the b_c^{min} = batch of the earliest departing container of stack c
2. $b^{container}$ = the departure batch of container x
3. If there are any stacks c for which $b_c^{min} > b^{container}$:
    a. Place the container in the stack with the highest b_c^{min} for which $b_c^{min} > b^{container}$
    b. In the case of a tie, place the container in the closest destination
4. Else, if there is any empty stack:
    a. Place the container in an empty stack
    b. In the case of a tie, place the container in the closest destination
5. Else:
    a. Place the container in the destination with the lowest b_c^{min}
    b. In the case of a tie, place the container in the closest destination

Constant (C)

As part of basis function design, one of the features needs to be a constant, which returns the same value for all states. This is done so that states that return zero on all features can still have a value, and so that for the linear regression to determine the weights of each feature, the approximation is not forced to go through the origin, which may introduce bias.

4.5. Single-Stage Optimization Methods

As mentioned in Section 4.3.3, the decision space for this problem explodes when using realistically sized terminals. Therefore, a method is needed to generate promising decisions. This section will discuss the methods that are used for this optimization problem.

4.5.1. Linear Programming

Linear programming is an efficient exact solution method to determine the optimal solution for decision problems with a large decision space, as the linearity of the problem formulation can be used to discard entire regions within $\mathcal{X}_t$ that are not promising. This makes Linear Programming much faster than enumerating over all options $x_t \in \mathcal{X}_t$. This does require the problem to be formulated in a linear program. There exist (integer) LPs of the individual unloading problem (Jin, 2020; Tang, 2015), but this problem needs to be adapted to be suitable for use in our problem setting.

The CRP as used in this research has minimization of the amount of reshuffles as well as the distance travelled by containers as the objective function, which needs to be included into the LP. Additionally, the value of S_t^x as estimated by a VFA needs to be incorporated as well. For a basis function this means that all features that make up the basis function need to be included in the linear program, and for a neural network this means that all input (features, metadata) as well as the structure of the neural network need to be written into the linear program.

As Linear Programming is an exact solution method, it still has the potential to have a prohibitively large computation time if the decision space is large enough, but with the possibility of lower costs. The LP formulated for the single-stage decision problem and VFA as used in this research can be found in Appendix A – Linear Program .

4.5.2. Partial Search Tree

The Partial Search Tree (PST) is a method adapted from (Boschma, 2020) in their research on the Dynamic CRP using ADP. This method uses the fact that within a timestep, the ADP problem is deterministic. It divides the decision needed for the MDP problem (a set of decisions to move outgoing containers, locate incoming containers and reshuffle blocking containers) into a sequence of individual actions to locate a container somewhere. It then uses a search tree with individual containers as nodes, and locations as branches, to quickly find a decision approximating the optimal decision. Which branch to explore next is determined using the value function approximation of the ADP algorithm; the node with the lowest sum of immediate costs and estimated future costs is chosen. If the end node of tree is reached, where no more containers need to be handled, the path from root to the end node forms one action x_t. As the decision space is prohibitively large, this method does not explore the search tree completely. Instead, it generates y number of decisions, and chooses the best option from those. The difference in method between this research and (Boschma, 2020) is that their algorithm explores one path through the tree and uses that option, while this research then goes back to explore other unexplored nodes, starting with the option that has the lowest combined direct contribution plus estimated future costs $C(S_t, X_t) + V(S_t)$. This can have added value, as the indication $V(S_t)$ is an approximation of future costs, meaning that it can under- or overestimate future costs.

To illustrate, we use the example in Figure 19. The order of containers that need to be handled is {1 out, 7 in, 4 out}. We start in the root node, where container 3 needs to be reshuffled (as it is the topmost blocking container of container 1). The options to reshuffle container 3 to are stack 0 and stack 2. We choose stack 0, as the sum of immediate + estimated future costs is the lowest for stack 0. Container 1 can then leave. The next action is to put container 7 in, which can go to stack 0, 1, or 2. Once every container in this batch is handled, we have a complete set of decisions, see Figure 20. The decision set made using this tree is then {3 to stack 0, 1 out. 7 to stack 1. 3 to stack 1, 4 out.}, marked in grey in Figure 20. The next option that is explored is the node with the lowest total estimated cost, which would be to place container 7 in stack 2.

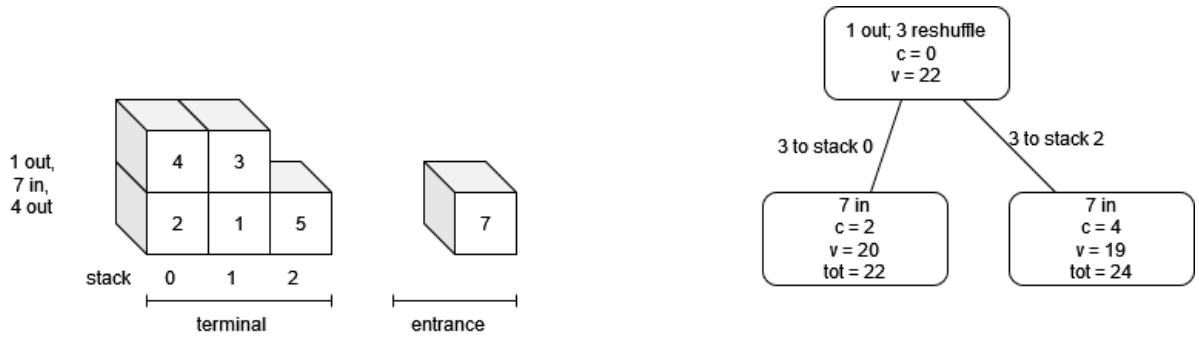


Figure 19 - example of PST, pre-decision

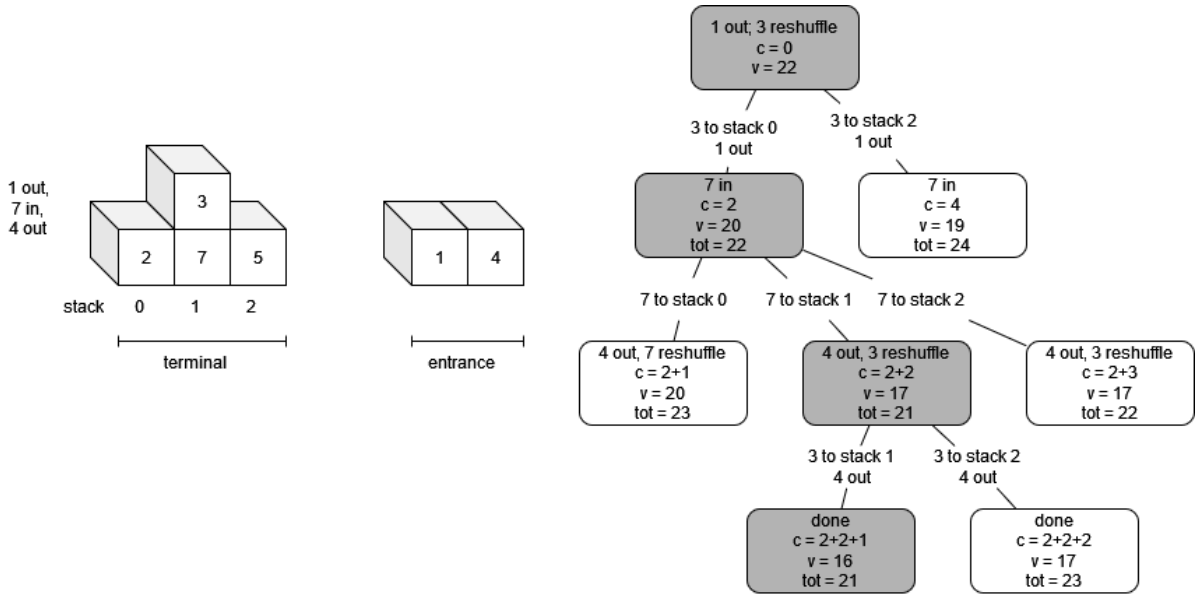


Figure 20 - example of PST, post-decision

The path of nodes in grey form one complete action for the MDP, but the PST does not stop at that point. It continues exploring unexplored nodes, starting with the lowest $C(S_t, X_t) + V(S_t)$, which in this case is to place container 7 in stack 2 instead of stack 1. It works out the rest of the tree until all containers are handled from that point, creating a new set of container allocations that form a complete action for the MDP. It repeats this process until the stopping criteria is met, which in this research is when y complete actions are made.

4.6. Pruning methods

Regardless of employing more effective methods to find the optimal decision, the speed at which this happens can be improved if actions that we know beforehand (probably) are not productive are not considered at all. As the vast amount of computation time is spent calculating the values of features, and that this is done for each potential post-decision state, reducing the amount of states that need to be evaluated reduces the computation time greatly.

Considering that the size of the decision space in our problem is given by $(nr\ of\ containers\ in\ batch) * (amount\ of\ locations\ available\ per\ container)$ for the PST, the decision space can be shrunk by eliminating a portion of the available spots per container. The pruning of unpromising moves is already done by using the restricted form of the CRP instead of the unrestricted form (see restriction 4, Section 1.4).

Another method that has previously been used in the Dynamic CRP is the use of a Corridor (Boschma, 2020; Caserta et al., 2009): When considering the possible stacks to reshuffle a container to, only the stacks within a certain range δ are considered, limiting the amount of options. In both papers the limit δ refers to the δ^{th} nearest neighbor stack, as both papers consider a terminal consisting of a rectangular grid, not considering distance between stacks. This research does consider distance, and intuitively it makes sense to reshuffle to a location that is nearby rather than far away, all other things being the same, if part of the objective is to minimize the amount of distance travelled per container.

In order to make sure the amount of destinations for any container is not reduced too much, the corridor that is implemented in this research only goes into effect when the stacks that are full and stacks where a container is not allowed are pruned beforehand. The list of possible destinations is then pruned to the δ^{th} closest stacks.

4.7. Conclusions

In this chapter we formulated the problem and the model that provides the solution to the problem. We first introduced the problem using a description and assumptions. The problem can be formulated as a Markov Decision Problem, consisting of states, stages, a transition function, an exogenous information process and an objective function. We use Approximate Dynamic Programming as the solution method for the MDP, which consist of a Value Function Approximation (VFA) to estimate future costs, and a Single-Stage Optimization Method to find the optimal decision in a given stage. For the VFA we discuss a Basis Feature Function and a Neural Network approach and discuss features that might be appropriate for our problem, both from literature and original contributions. For the Single-Stage Optimization Method we discuss Linear Programming and a Partial Search Tree as options. Lastly, we discuss methods to prune the decision space in a single stage by introducing a Corridor method, which restricts movements to those with a low distance cost.

5. Experiments and Analysis of Results

In this chapter, several experiments are performed to evaluate the performance of the ADP algorithm when used with particular settings described in Chapter 4 and compare its performance against heuristics.

5.1. Experiment Design

The structure of experiments is as follows. We first generate a set of problem instances that are going to be used for all the experiments. This allows for consistent comparison of results between experiments. We then perform several experiments regarding different options and settings for the ADP algorithm, and then test its performance in different problem settings. The experiments are done in sequence and concern one aspect each, namely:

1. Features
2. Feature sets
3. Algorithm for updating weights of the Basis Feature Function
4. Choice policy
5. Pre-training
6. Corridor method
7. Single-stage optimization methods

These experiments are done in sequence, as testing all combinations of configurations is prohibitively time-consuming. The experiments are done in the listed order, as we estimate that in this order the most impactful experiments are done first. As these experiments are done sequentially, we initially assume some commonsense default values for the algorithm to begin with and adjust settings as the optimal parameters for each setting is found. These default settings are given in Appendix E. Afterwards, we also run experiments on using a Neural Network as an alternative to a Basis Feature Function. After running the mentioned experiments, we evaluate the performance and variance in performance of the ADP algorithm in different problem settings.

5.2. Problem Instance Generation

To evaluate the performance of different variants of the algorithm that this chapter investigates, a set of problem instances need to be generated. This set of problem instances can be used to compare the results of different algorithms more fairly. Problem instances are generated using a script that takes in a set of desired characteristics of the problem. These characteristics are:

- *Number of stacks; c*

The number of stacks present in a terminal. This variable influences the capacity of a terminal and how many options are available to locate a container to.

- *Number of tiers; p*

The number of tiers per stack influences the capacity of a terminal, and how many containers each container is expected to have below it, assuming the same fraction of occupation. With a larger amount of tiers, stacking properly becomes more important.

- *Expected Length of stay; $E[LoS]$*

The expected length of stay of a container in the terminal in hours. Containers stay in the terminal with an exponential distribution, with an average stay of $E[LoS]$. In a terminal with a lower length of stay, a larger fraction of the containers in a terminal leaves each batch.

- *Expected number of cycles;* $E[cycles]$

The planning horizon of the problem, expressed as the expected amount of container cycles a terminal encounters. Each container stays in the terminal for $E[LoS]$ hours, and the problem instance deals with a schedule of incoming/outgoing containers of t hours. The timespan t that a problem instance deals with is then given by $t = E[cycles] * E[LoS]$

- *Average occupation;* $E[occ]$

Average occupation of the terminal as a fraction of the total capacity of a terminal. This variable determines how busy the terminal is expected to be at any given moment. The initial terminal layout as well as the schedule of arrivals and departures is generated semi-randomly such that occupation can fluctuate, but hovers around the given $E[occ]$.

- *Batch timespan;* h

The timespan that each batch covers in hours. Larger batch timespans mean a lower amount of batches in a problem instance, but with a larger amount of containers per batch. This increases the uncertainty of a problem instance, as the order of arrivals/departures of containers between batches is known, but not within a batch.

To generate the problem set, for each of these settings a standard value is used, as well as a lower setting and a higher setting. One problem instance will have standard values for all of these settings, and twelve other problem instances have a higher or lower value for one of these settings. This means that the problem instances are generated with the following settings:

Table 9 - settings of generated problem instances

instance	c	p	E[LoS]	E[cycles]	E[occ]	h
0	20	4	40	4	0.6	1
1	10	4	40	4	0.6	1
2	40	4	40	4	0.6	1
3	20	3	40	4	0.6	1
4	20	6	40	4	0.6	1
5	20	4	20	4	0.6	1
6	20	4	80	4	0.6	1
7	20	4	40	2	0.6	1
8	20	4	40	8	0.6	1
9	20	4	40	4	0.4	1
10	20	4	40	4	0.8	1
11	20	4	40	4	0.6	0.5
12	20	4	40	4	0.6	2

These problem instances are used in the following sections. As can be noted from the table, the generated terminals are not realistically sized, containing 10, 20 to up to 40 stacks. Real terminals contain upwards of 300 stacks. However, several factors limit the problem size that this thesis will be able to handle. ADP has high computation costs due to the iterative nature of the algorithm. Additionally, the algorithm is written in Python, a flexible but slow programming language, and some aspects of the algorithm were not efficiently written, such as the aspect that in each state, all features are calculated from scratch, instead of calculating the changes in values from the changes since the previous state. Additionally, this thesis trains multiple hundreds of algorithms over the course of all the experiments, so the available time to train each algorithm is limited. We additionally do not train

the algorithm on problems available in literature, as no literature with the exact same problem formulation could be found.

The pseudo-code that was used to generate the problem instances is provided in Appendix B, and an example of a complete problem instance is given in Appendix C. In order to make experiments better comparable to each other and over iterations, random-number generation is controlled in two ways. For the evaluation simulations that are used to track the performance of the algorithm over iterations, the same set of 5 pre-generated samples is used, and at the start of training a random-number generator is initialized so that for each algorithm instance the same random samples are drawn during training.

5.3.Feature Generation

A crucial step in using ADP with a Basis Feature Function successfully is to have features that give useful information about future costs. We thus investigate the Pearson's correlation between available features and future costs. To do this, we generated 1000 problem instances with the characteristics of problem instance 0 and solved the problem instances using the min-max heuristic. We noted the total costs, and costs in each category (distance, reshuffling, wrong-stack penalty). Of these states, we use all states in timesteps 0-20, leaving us with 20.000 states. For each feature mentioned in Section 4.3.2 the correlation between it and the future costs is calculated. Additionally, we generate composite features using the following methods:

- Generate a composite feature by taking the squared value of a feature
- Generate a composite feature by taking the square-root of a feature
- Generate a composite feature by calculating the product of two features
- Generate a composite feature by calculating the product of a feature with a terminal characteristic.
    - A terminal characteristic is an indication of the terminal that cannot be influenced by the actions taken. Used characteristics are:
        - Batch size of timestep t + 1 and t+2
        - Occupation of the terminal
        - Amount of incoming containers at t+1, t+2 of type 0, 1, 2, 3
        - Amount of outgoing containers at t+1, t+2 of type 0, 1, 2, 3
        - Average, st.dev., min, max of leaving timestep of containers at t+1, t+2

With 23 original features, this results in $23 + 23 + 253 + 621 = 920$ additional composite features. These are too many too list, so in Table x we give the correlation for all original features as well as the top 10 highest correlating composite features for the total future costs, distance future costs, reshuffle future costs and penalty future costs. To calculate future costs, we use a discounting factor of 0.9, as this resulted in the highest average correlation between future costs and feature values. Additionally, we use the absolute value of the correlation, as both a large negative or positive Pearson's correlation indicate that a feature contains useful information about future costs.

Table 10 – Selection of Absolute Pearson's correlation between future costs of different types and (composite) feature values

Feature name	Total	Distance	Reshuffle	Penalty
√RIH	0.512	0.263	0.562	0.483
RIH	0.510	0.256	0.558	0.488
√MMH	0.509	0.265	0.560	0.476
MMH	0.507	0.258	0.556	0.481
RIH*MMH	0.495	0.238	0.540	0.486
RIH^2	0.494	0.236	0.538	0.487
MMH^2	0.492	0.239	0.537	0.480
RIH*EBLB	0.452	0.236	0.582	0.362
RIH*SOS	0.451	0.235	0.580	0.361
MMH*EBLB	0.449	0.236	0.579	0.356
MMH*SOS	0.447	0.235	0.577	0.355
RIH*ASH	0.439	0.191	0.465	0.462
MMH*ASH	0.437	0.193	0.463	0.458
RIH*OCC	0.423	0.183	0.443	0.450
√LAEBLB	0.414	0.241	0.579	0.273
LAEBLB	0.410	0.231	0.577	0.275

Table 10 shows the highest correlating features found. The complete table with all features is given in Appendix D. The selection consists mostly of the Reshuffle-index heuristic, Min-Max heuristic and variations on it. Another notable thing is that even the features that correlate most with the cost categories have a relatively weak correlation value, as in general features with a correlation value of >0.75 are seen as highly correlating, 0.5-0.75 are moderately correlating, and anything <0.5 is seen as weakly correlating. This suggests that no single feature in this set is capable in itself of approximating the value of future costs.

5.4. Feature Set Selection

This section evaluates the selection of a feature set for the basis feature function, using the discussion of individual features in Section 5.2. We first test the performance of the ADP algorithm using all features, and then generate a variety of feature sets using a combination of sequential feature selection, nearest-neighbor search, and several handmade feature sets. For the ADP algorithm we use common sense settings wherever possible. We use the settings given in Appendix E.

One measure to evaluate how useful each feature is, is to look at the contribution of each feature to the basis feature function. The contribution of a feature to the future cost estimation is given by its weight multiplied by its activation, and thus the contribution of each feature in simulation n at timestep t can be expressed as:

$$contribution\ of\ feature\ i = \frac{\theta_i^n \phi_i(S_t^{x,n})}{\sum_{a \in \mathcal{A}} \theta_a^n \phi_a(S_t^{x,n})}$$

At the end of the training period, the trained agent is run through a set of 5 evaluation simulations, which have pre-defined random events, which allows them to be compared more easily over iterations and settings. This still leaves $(nr\ scenarios) * (nr\ simulations) * (nr\ timesteps/scenario)$ of observations per feature. From these observations we calculate the average and the maximum over the nr of simulations and timesteps to find which features have a large contribution, and which do

not. The top features in both of these indicators are shown in Table 7 and 8, with the full tables given in Appendix F.

Table 11 – average contribution of each feature from all timesteps per problem

Feature	0	1	2	3	4	5	6	7	8	9	10	11	12	avg
TDLB	0.049	0.046	0.050	0.064	0.041	0.039	0.057	0.049	0.045	0.052	0.052	0.049	0.055	0.050
ASH	0.058	0.051	0.051	0.068	0.049	0.046	0.070	0.054	0.046	0.049	0.068	0.053	0.055	0.055
SSH	0.045	0.034	0.040	0.055	0.037	0.037	0.051	0.042	0.033	0.037	0.050	0.037	0.038	0.041
NIS1	0.051	0.052	0.044	0.068	0.047	0.056	0.061	0.049	0.048	0.058	0.057	0.055	0.053	0.054
NIS2	0.056	0.063	0.044	0.064	0.047	0.056	0.062	0.049	0.071	0.063	0.058	0.066	0.053	0.058
MMV	0.028	0.057	0.025	0.028	0.036	0.032	0.025	0.036	0.049	0.081	0.031	0.041	0.034	0.039
MMH	0.046	0.040	0.039	0.050	0.045	0.050	0.046	0.050	0.041	0.051	0.049	0.045	0.050	0.046
RIH	0.044	0.040	0.039	0.050	0.043	0.048	0.042	0.048	0.042	0.051	0.047	0.045	0.049	0.045
EBLB^0.5	0.063	0.052	0.049	0.056	0.052	0.049	0.053	0.051	0.051	0.028	0.057	0.046	0.055	0.051

Table 12 –max contribution of each feature from all timesteps per problem

Feature	0	1	2	3	4	5	6	7	8	9	10	11	12	avg	max
NIS1	0.283	0.674	0.262	0.473	0.481	0.281	0.480	0.254	0.632	0.509	0.226	0.690	0.413	0.435	0.690
NIS2	0.288	0.797	0.206	0.503	0.435	0.390	0.576	0.225	1.101	0.409	0.271	0.705	0.379	0.483	1.101
NIC1	0.279	0.208	0.203	0.260	0.498	0.194	0.279	0.198	0.234	0.202	0.304	0.193	0.349	0.262	0.498
FIC1	0.698	0.756	0.707	0.841	0.700	0.779	0.948	0.377	0.986	0.347	0.656	0.480	0.305	0.660	0.986
FIC2	0.587	0.791	0.176	0.885	0.621	0.557	0.715	0.506	0.374	0.261	0.535	0.694	0.347	0.542	0.885
FOC1	0.703	0.647	0.718	0.719	1.807	0.845	0.699	0.698	0.711	0.724	0.937	0.644	0.563	0.801	1.807
FOC2	0.652	0.333	0.683	0.572	0.820	0.665	0.683	0.412	0.645	0.469	0.835	0.627	0.313	0.593	0.835
EBLB^2	0.088	0.301	0.120	0.233	0.101	0.136	0.135	0.101	0.675	0.309	0.091	0.255	0.109	0.204	0.675
NES^2	0.136	0.251	0.090	0.102	0.079	0.047	0.116	0.102	0.171	0.058	0.258	0.123	1.157	0.207	1.157

Values that have a low standard deviation and a low absolute average consistently contribute little to the value estimation. Features that jump out this way are MWSP1, MWSP2, NES^2 and HUSP^2. Others occasionally or consistently contribute a significant amount to the value estimation. Features that (occasionally) contribute a relatively large amount to the value estimation are TDLB, ASH, NIS1, NIS2, FOC1, FOC2, MMV, EBLB^0.5. These contributions can give some idea as to which features are useful to have in a feature set. A number of other feature sets are composed and trained with, namely:

1. 'no bad': the full feature set without bad performers (MWSP, NES^2, HUSP^2)
2. 'only good': a feature set containing only the top contributing features (TDLB, ASH, NIS1, NIS2, FOC1, FOC2, MMV and EBLB^0.5)
3. 'composite': a feature set containing 1 feature; a composite feature containing the sum of
    a. α_1 * Expanded-EBLB
    b. α_2 * TDLB
    c. α_3 * $MWSP_1$
4. 'heuristics': a feature set containing two heuristics; Min-Max and Reshuffle-Index
5. 'future features': a feature set containing all the features whose evaluation depends on future incoming/outgoing containers
6. 'half': a feature set containing the 18 features that had an above average contribution to the value estimation as mentioned in Table 7
7. 'top 14': a feature set containing the top 14 features in contribution to the value estimation:
    a. C, ASH, √EBLB, TDLB, NIS2, MMH, RIH, NIS1, MMV, SSH, √BD, BLD, US, EBLB
8. 'new 1': a handpicked feature set containing:

   a. √EBLB, TDLB, MMH, RIH, SSH, √BD, BLD, US, E-EBLB, $EBLB^2$, C
9. 'new 2': a handpicked feature set containing:
   a. ASH, NIS2, MMH, RIH, NIS1, MMV, LA-EBLB, BD, HUSP, C
10. 'new 3': a handpicked feature set containing:
   a. ASH, √EBLB, TDLB, NIS2, MMH, RIH, NIS1, NIC1, MMV, SSH, √BD, BLD, US, E-EBLB, FOC1, FOC2, BD, $EBLB^2$, USP1, HUSP, MWSP1, C

Additionally, we generate potential feature sets using sequential feature selection and neighbor-search: With the states and future costs discussed in Section 5.3 we first create a feature set using backwards sequential feature selection. This procedure evaluates the performance of an estimator (in this case the Lasso approximation, a variant on Ordinary Least Squares that tries to minimize the amount of non-zero weights) on a given feature set, initially the full set of features. It then removes features one at a time, keeping the feature set that loses the least amount of accuracy, until the desired feature set size is achieved. Using this initial feature set, we create new feature sets using neighbor search. By removing or adding a feature and testing its performance using the same estimator, we generate 100 feature sets that can estimate the future costs using a linear model estimator. We then select 10 from these that ensure a variety of set sizes and set compositions, which are called 'sfs 1 – 10'. An overview of which set contains which feature is given in Appendix G. Testing all these feature sets gives the following results:

Table 13 - The realized Cost, MAE, R-squared and R-squared of the lasso linear approximation for each feature set. For some feature sets the lasso R^2 was not available as the set wasn't tested using a lasso approximation.

KPI	Cost	MAE	R^2	lasso R^2
composite	366.4	10.520	0.9761	-
full	338.8	9.319	0.9816	0.3639
future_features	433.7	23.437	0.9241	-
half	317.3	8.083	0.9828	-
heuristic	324.2	9.897	0.9759	0.2641
heuristic_indicators	345.6	10.383	0.9756	-
new_1	325.6	9.344	0.9753	0.1323
new_2	315.6	7.938	0.9823	0.2961
new_3	325.9	8.755	0.9809	0.3123
no_bad	340.5	9.599	0.9790	-
only_good	340.2	10.352	0.9748	-
reduced full	337.3	9.778	0.9774	0.3639
sfs 1	317.2	8.708	0.9797	0.3583
sfs 2	345.4	11.528	0.9671	0.3484
sfs 3	344.1	11.115	0.9717	0.3353
sfs 4	355.6	11.966	0.9706	0.3378
sfs 6	336.8	9.659	0.9785	0.3571
sfs 7	354.5	12.613	0.9671	0.3614
sfs 8	352.1	10.734	0.9743	0.3407
sfs 9	384.7	13.837	0.9653	0.3518
sfs 10	331.5	10.311	0.9719	0.3548
top_14	316.7	8.054	0.9822	0.3306
Min-max*	333.0	-	-	-
Reshuffle-index*	373.4	-	-	-

A couple of things stand out from this data. The difference in performance between feature sets is stark, with ‘half, ‘top 14’, ‘new 2’ and ‘sfs 1’ standing out as positive outliers, while ‘future features’ performs the worst by a large margin. A significant portion of feature sets also fails to perform better than the min-max heuristic, the best performing heuristic that was tested.

Another interesting result is that prediction accuracy and lower costs are a not a one-to-one correlation, which is especially true for the prediction accuracy of the lasso approximation. An explanation for this is that the point of minimizing costs using a basis feature function lies not in accurate prediction of costs, but in a composition that results in choosing states that result in lower costs vs higher costs. To minimize costs, accurately predicting future costs may thus be sufficient, but not necessary. For future experiments, the feature set “top 14” is used, as it is the top performer together with ‘new-2’ and ‘sfs 1’.

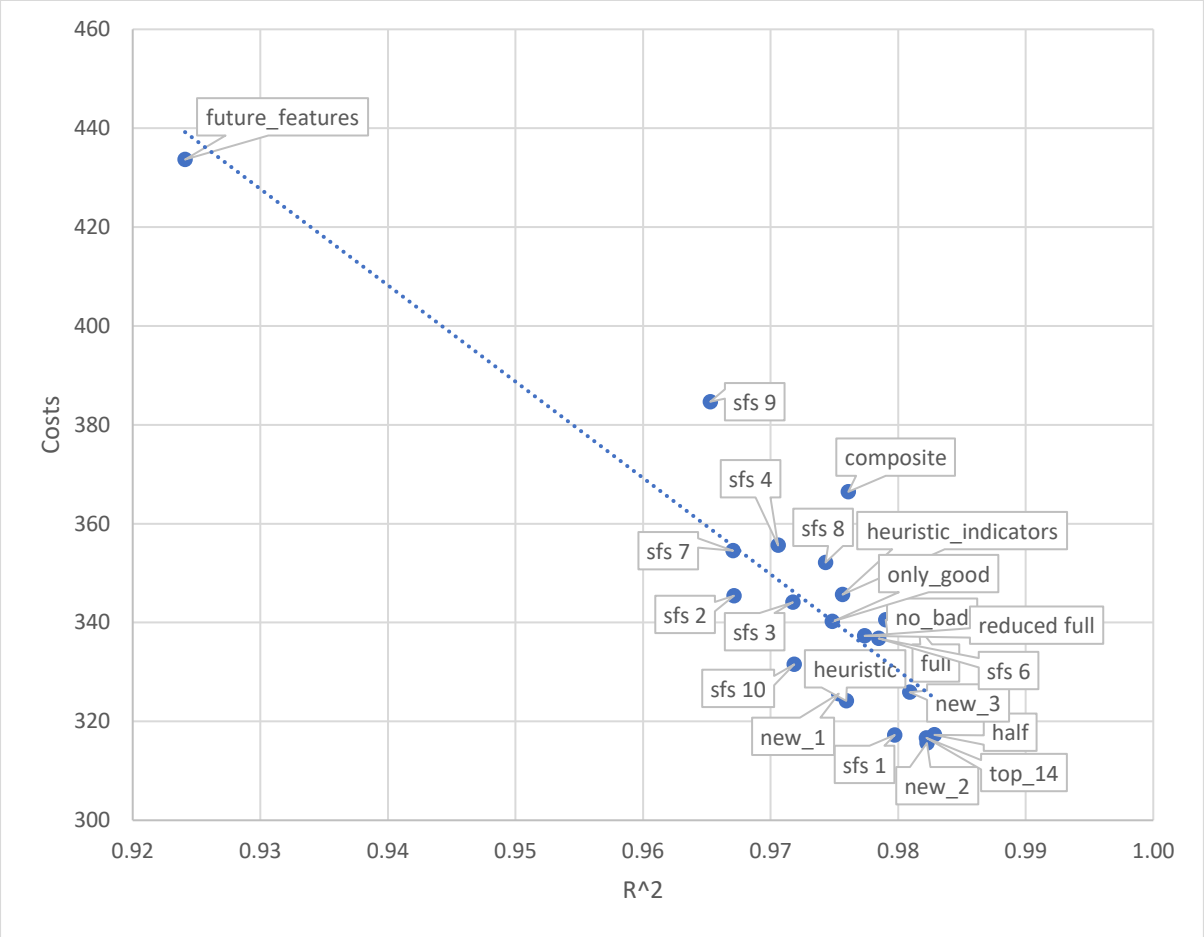


Figure 21 - Scatter plot of costs vs accuracy (in the form of r squared) of the final iteration of each feature set

5.5. Weight Updating Algorithm

The weight updating algorithm is responsible for adjusting the weights for each feature in the VFA. For this, we use the Recursive Least Squares (RLS) method. This method has the advantage over Stochastic Gradient Descent (SGD) in that the convergence towards the optimum is much faster, and in preliminary testing this method did not result in lower final performance than SGD. We use a variant of RLS with a harmonic step size. The algorithm requires two inputs, ρ and δ. The algorithm works using the following formula’s:

$$B_0 = \rho * \mathbb{I}$$

$$a_n = 1 - \delta/(n+1)$$

$$\gamma_n = \alpha_n + (\phi_i(S_t^{x,n}) * B_n * \phi_i(S_t^{x,n}))$$

$$H_n = \frac{1}{\gamma_n} * B_n$$

$$update = H_n * (\phi_i(S_t^{x,n})) * error$$

$$weights_{n+1} = weights_n - update$$

$$B_{n+1} = \frac{1}{\alpha_n} * \left(B_n - \frac{1}{\gamma_n} (B_n * \phi_i(S_t^{x,n}) * \phi_i(S_t^{x,n})^T * B_n) \right)$$

From the formula's we can deduce that ρ has an influence on the initial values of B_n, which results in a larger value of γ_n, which leads to a lower H_n, which leads to a smaller update of weights. A larger value of δ results in a lower initial α_n, which leads to a larger value for γ_n, and thus also a smaller update of weights. We first test a range of values for δ: 0, 0.25, 0.5, 0.75 and 0.999 (a value of 1 results in a division by 0). This leads to the following results:

Table 14 - settings and results of the different options for delta. (=average of last 50 iterations, **=heuristics for comparison)*

delta	a_0	a_100	a_500	R^2*	MAE*	Cost*
0	1	1	1	0.9825	12.28	310.4
0.25	0.75	0.9975	0.9995	0.9836	11.78	311.2
0.5	0.5	0.995	0.999	0.9832	12.28	312.0
0.75	0.25	0.993	0.999	0.9832	12.32	312.4
0.999	0.001	0.990	0.998	0.9865	10.99	312.1
Min-max**						333.018
Reshuffle-index**						387.0723

Overall, there does not seem to be a significant difference between any of the options (a difference of -2 between the best and worst performer for cost). As δ=0.5 was used before, this value will be kept in future experiments. For the value of ρ we test the following values: 0.025, 0.05, 0.1, 0.2 and 0.4. This leads to the following results:

Table 15 - Settings and results of the different options for rho

KPI	0.025	0.05	0.1	0.2	0.4
Cost	333.018	373.4279	315.6315	317.5825	315.38
MSE	176.0997	165.5349	168.581	170.6431	152.9873
Bias	-0.32787	0.064635	-0.80046	-0.62358	0.673754

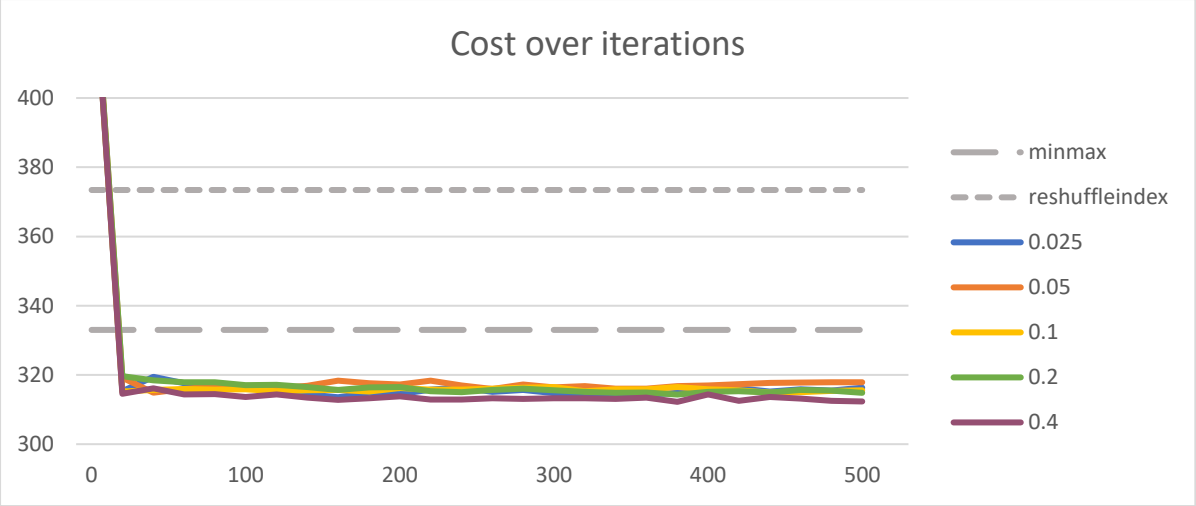


Figure 22 - Cost over iterations for different settings of rho, with min-max and reshuffle-index heuristics for comparison

While we see lower costs and MSE for a ρ of 0.4, the progression in costs over iterations is similar between all settings of ρ. For future experiments we use a value for ρ of 0.4.

5.6. Choice policy

The first parameter of the basis feature function that is tested is the choice policy. A decreasing-epsilon policy is used, where with a probability of ϵ a random action is taken. This value of ϵ is multiplied by a factor each iteration, so that it decreases over time. A variety of settings is tested with the following results:

Table 16 - Settings and results of different choice policy options (= average of the last 50 iterations, **=heuristics for comparison)*

Name	Initial e	multiplication factor	e at 100 iterations	e at 500 iterations	Cost*	MAE*	R^2*
pure exploitation	0	0	0	0	312.7	7.48	0.985
low non-decreasing	0.02	1	0.02	0.02	311.4	7.89	0.979
medium non-decreasing	0.05	1	0.05	0.05	310.4	7.90	0.983
high non-decreasing	0.2	1	0.2	0.2	309.6	7.62	0.985
medium decreasing	0.2	0.99	0.073	0.0013	311.5	8.36	0.976
highly decreasing	1	0.95	0.0059	7.27E-12	309.6	8.51	0.978
Minmax**					333.0		
Reshuffle-index**					387.1		

From the results, no significant differences in performance are distinguishable; the worst performing option has a score that is 1% worse than the best performing option. Just as in the experiments with feature sets, there is no strict correlation between prediction accuracy and performance.

One would expect at least a difference between 'high non-decreasing epsilon' and 'pure exploitation', so this lack of difference is in contrast with other reinforcement-learning problems. A possible explanation for this lack of difference could be the magnitude and compounding effect of randomness in this problem setting, akin to the 'butterfly effect'; two containers that arrive in a swapped order early in the problem can result in a slightly different container layout in the next timestep, which causes different actions in the next timestep, and the further along the problem is, the larger the differences become. In this way, 'exploration' is forced upon the algorithm due to the inherent randomness of the problem. Another explanation is that the algorithm is not able to learn the future costs to such a degree that exploration vs exploitation becomes a relevant matter.

As there was no significant difference between options, the option that is chosen is 'medium decreasing' epsilon, as that is a common choice for choice policies and was the default choice in the setup of these experiments. This choice policy is used for future experiments.

5.7. Pre-training

In ADP, feature weights for the BFF are commonly initialized at 0 to prevent unrealistically large estimated future costs. This does mean that initially the algorithm learns future values using a myopic policy, which typically results in high predicted future costs. To prevent this, it is possible to do pre-training, where the decision making in a simulation is done by a heuristic, while algorithm is updated to estimate future costs. To test this, the algorithm is pre-trained for 10, 100 and 1000 iterations using the min-max heuristic. Table 17 shows the resulting costs, MAE and R^2.

Table 17 - Costs, MAE and R^2 for different amounts of pre-training

KPI	0	10	100	1000	Min-max	Reshuffle-index
cost	315.4244	317.555	319.608	320.0199	333.018	373.4279
MAE	8.318282	6.946515	8.116679	8.272115	-	-
r_2	0.980588	0.986581	0.981394	0.980472	-	-

The results are paradoxical; while all amounts of pre-training improve the accuracy of the approximation, resulting in a higher R^2 and lower MAE, this improved accuracy results in higher realized costs.

5.8. Corridor

A measure that can be used to speed up the algorithm is a corridor; a pre-selection method that makes it so only the *x* closest options are considered. To test the effect of this measure on time-savings and performance, the problem instances are all tested without corridor, and a corridor of 30, 20, 15, 10, 8, 7, 6 and 5. The effect of this corridor can depend on the problem instance being tested, as a corridor of size 5 means that in problem instance 2 5/50=10% of all stacks are considered, while in problem instance 1 5/10=50% of all options are considered. We get the following results:

Table 18 – Costs per problem instance / corridor size, expressed as a fraction of the costs in 'No corridor'. Instance 1 has 10 stacks, Instance 2 has 50, all others have 20. (*=average of last 100 iterations)*

Problem Instance	5	6	7	8	10	15	20	30	None	Min-max	Reshuffle-index
0	1.04	1.02	1.00	0.98	1.02	1.00	-	-	1.00	1.04	1.26
1	1.04	1.08	1.03	1.07	-	-	-	-	1.00	1.13	1.15
2	1.11	1.07	1.04	1.02	1.00	1.00	0.99	1.01	1.00	1.01	1.42
3	1.04	1.03	1.02	1.01	1.00	1.01	-	-	1.00	1.09	1.17
4	0.99	0.98	0.97	0.96	0.96	0.99	-	-	1.00	1.07	1.34
5	1.11	1.02	1.04	1.04	1.00	0.99	-	-	1.00	1.13	1.33
6	1.02	1.02	1.02	1.03	1.02	1.00	-	-	1.00	1.04	1.32
7	1.03	1.01	0.97	1.01	0.99	1.00	-	-	1.00	1.04	1.27
8	1.06	0.98	0.95	0.96	0.98	0.97	-	-	1.00	0.94	1.11
9	1.00	0.99	0.98	1.04	1.02	0.98	-	-	1.00	1.16	1.05
10	1.06	1.03	1.02	1.00	1.05	1.02	-	-	1.00	1.04	1.22
11	1.00	1.00	1.04	1.01	1.01	1.00	-	-	1.00	1.11	1.37
12	1.06	1.05	1.04	1.02	1.01	1.00	-	-	1.00	1.09	1.18
avg	1.044	1.022	1.011	1.011	1.004	0.997	0.993	1.005	1.000	1.068	1.245

We expect to get a gradual and increasingly sharp increase in the costs as the corridor size decreases. While this is true in general, we see that for some problem instances this is not true; instance 4 has no increase in costs, and in some cases costs decrease going from a larger to a smaller corridor.

Table 19 - Time per problem instance / corridor size, expressed as a fraction of no corridor

Problem Instance	5	6	7	8	10	15	20	30	None
0	0.65	0.88	0.89	0.96	0.86	0.86	-	-	1.00
1	0.70	0.98	1.01	1.09	-	-	-	-	1.00
2	0.45	0.60	0.62	0.68	0.67	0.79	0.82	0.82	1.00
3	0.55	0.78	0.82	0.85	0.77	0.82	-	-	1.00
4	0.63	0.83	0.88	0.86	0.76	0.79	-	-	1.00
5	0.54	0.72	0.80	0.83	0.73	0.79	-	-	1.00
6	0.64	0.87	0.91	0.93	0.82	0.85	-	-	1.00
7	0.69	0.90	0.91	0.95	0.82	0.86	-	-	1.00
8	0.52	0.67	0.72	0.77	0.70	0.75	-	-	1.00
9	0.40	0.58	0.63	0.70	0.64	0.79	-	-	1.00
10	0.73	0.94	0.96	0.97	0.83	0.85	-	-	1.00
11	0.65	0.66	0.73	0.73	0.89	0.98	-	-	1.00
12	0.79	0.95	1.04	1.02	0.87	0.87	-	-	1.00
avg	0.612	0.797	0.840	0.872	0.781	0.834	0.815	0.822	1.000

Time savings across problem instances and corridors are inconsistent, but this is somewhat expected; the speed at which the experiments ran is dependent on a number of things such as other processes executed on the computer. In general however, they show a slight decrease in time which gets more drastic as the corridor gets narrower. As a corridor size of 15 is the smallest corridor that does not increase the average costs compared to no corridor, this setting is used for future experiments.

5.9. Single-Stage Optimization Method Experiments

This section discusses experiments regarding the short-term decision-making tools this problem explores; a LP and a PST (see Section 4.4).

5.9.1. Partial Search Tree

The PST can have a different amount of tries, which can increase performance at the cost of extra computation time. The algorithm is tested using a range of 1 to 10 attempts for the PST as explained in Section 4.5.2, and these are the results:

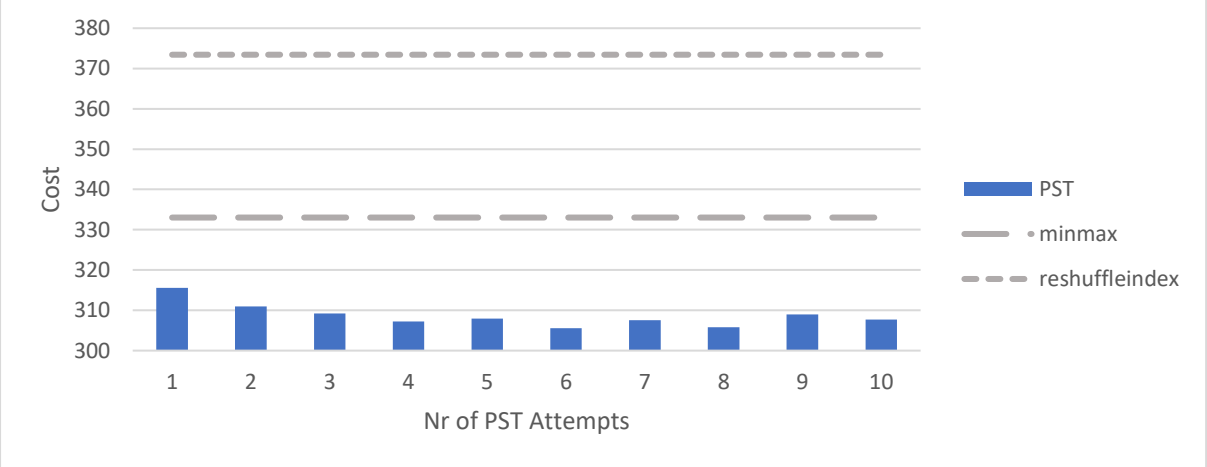


Figure 23 - Cost over iterations for the PST settings (minmax heuristic added for comparison)

This figure shows a decline up to 6 attempts, at which point there is no visible improvement. The MSE of each algorithm shown in Table 20 shows a similar pattern.

Table 20 - Cost, Bias and MSE for all PST settings

Attempts	1	2	3	4	5	6	7	8	9	10
Cost	315.6	311.0	309.2	307.2	308.0	305.5	307.5	305.8	309.0	307.7
Bias	-0.226	0.129	0.077	0.684	0.066	1.412	0.143	0.728	-0.610	0.134
MSE	210.7	170.0	166.5	153.3	153.9	144.2	153.2	156.2	133.9	142.2

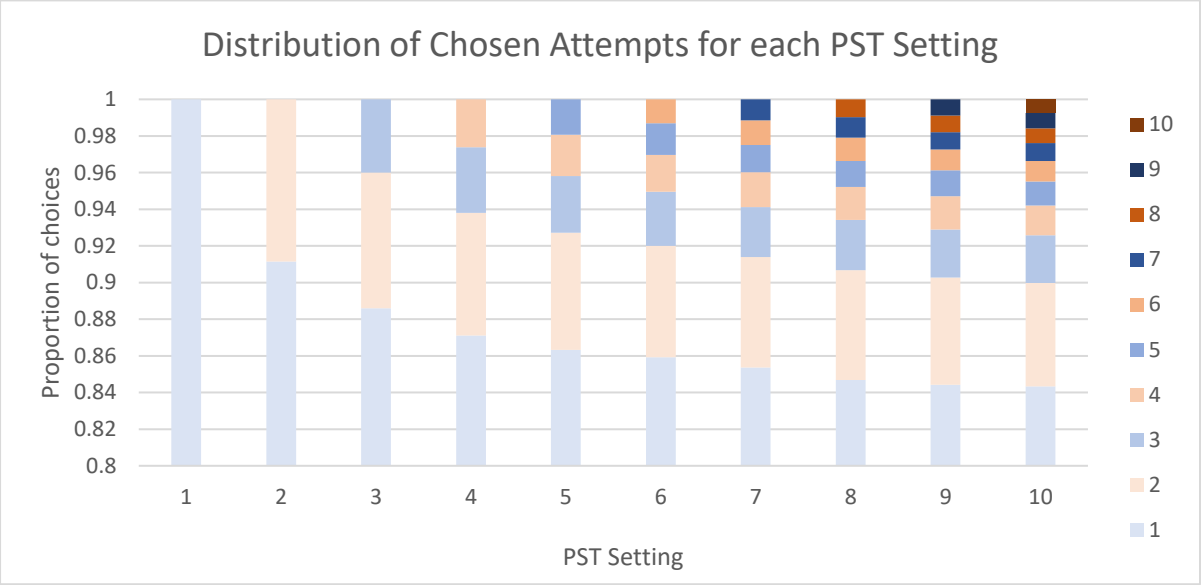


Figure 24 - stacked histogram of the proportion of times option x is chosen in the PST, with option x being the x-th finished solution from the PST

This histogram shows the proportion of times each option is chosen in the PST. The vast majority of times the first option from the PST is found to be the optimal one, with the first option being the best option even if the partial search tree provides 10 options. The PST with 6 attempts provided the lowest costs overall, so this setting is used in the Neural Network experiments.

5.9.2. Linear Programming

To compare the PST heuristic to an exact method we implement the LP as discussed in Section 4.5.1. It is however practically impossible to train the ADP algorithm using the LP as a single-stage solution method, as the method takes a prohibitively long time in python; compiling the LP at each timestep costs 30s to 3 mins, and solving it takes around 30s. The PST at 6 attempts, for comparison, requires 0.2 to 1 second per stage. To have some comparison between the performance of the two models, we first train an agent using the PST with 6 attempts. We then use the weights learned with those settings for the LP and run through the 5 evaluation simulations. The average cost per category is shown in Figure 25. From this figure it is clear that the LP outperforms the PST, as it resulted in a reduction of 5.3% costs overall, with a reduction of 10% in the amount of reshuffles.

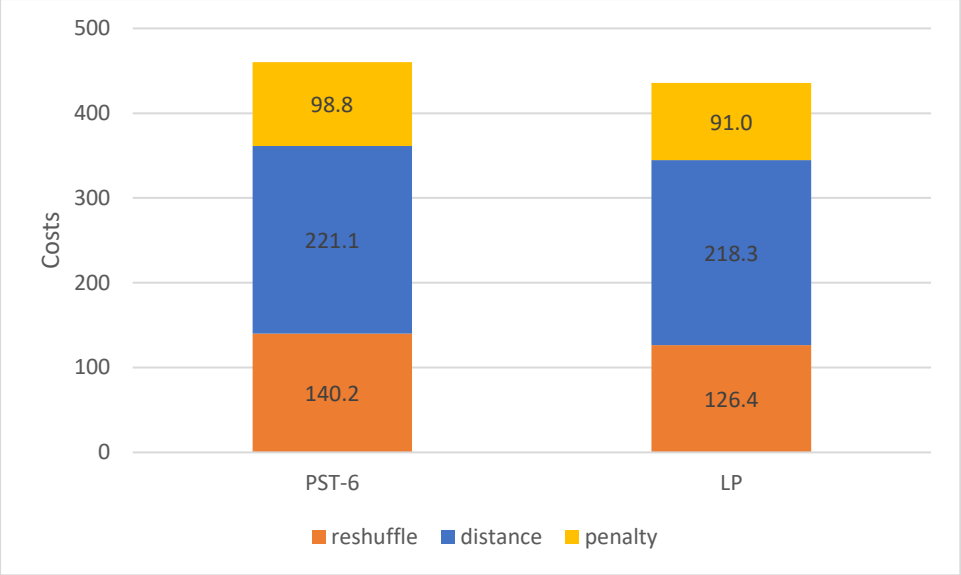


Figure 25 - Costs per category for the PST-3 and the LP

5.10. Neural Network Design Experiments

This section discusses experiments related to a Neural Network as a value function estimation alternative to the basis feature function. In order to keep the approach similar to the basis feature function, features are still used as input. However, in contrast to the basis feature function, one set of weights is used for all timesteps, and we use all features in the feature set that provided some amount of contribution in Section 5.4: E-EBLB, TDLB, BD, US, BLD, ASH, SSH, USP1, USP2, USP3, USP4, NIS1, NIS2, NIC1, FIC1, FOC1, MWSP1, MMV, MMH, RIH. In addition we provide some characteristics of the terminal as input, which provide an indication of future costs in the problem that cannot be influenced by the actions taken, those being: Timestep, batch size of t+1 and t+2, occupation of the terminal, amount of incoming containers at t+1, t+2 of each container type, amount of outgoing containers at t+1, t+2 of each container type, and the average/st.dev./min/max of departure timestep of containers at t+1 and t+2.

Additionally, the Neural Networks need more iterations to reach their optimal performance: 5000 instead of 500. To keep computation times reasonable, only problem instance 0 is tested on instead of all instances. Experiments are done on the learning rate schedule and the layers / layer size. To start with, one layer of 20 nodes with ReLu activation is used.

5.10.1. Learning rate schedule

The first setting used for the neural network approach is the learning rate schedule. An exponential decay learning rate schedule is used, where the learning rate is given by the following formula:

$$r_i = r_0 * d^i$$

Where the learning rate at iteration i is r_i, the initial learning rate is r_0 and the decay d. We test the following settings for r_i and d, shown in Table 21:

Table 21 - Initial learning rate, decay rate, and learning rate at 500, 2500 and 5000 iterations.

name	r_0	d	r_{500}	r_{2500}	r_{5000}
high decay	0.1	0.99	6.57E-04	1.22E-12	1.50E-23
high initial	0.3	0.995	2.45E-02	1.08E-06	3.91E-12
low decay	0.1	0.9975	2.86E-02	1.92E-04	3.67E-07
low initial	0.033	0.995	2.69E-03	1.19E-07	4.30E-13
medium	0.1	0.995	8.16E-03	3.61E-07	1.30E-12

This leads to the following results:

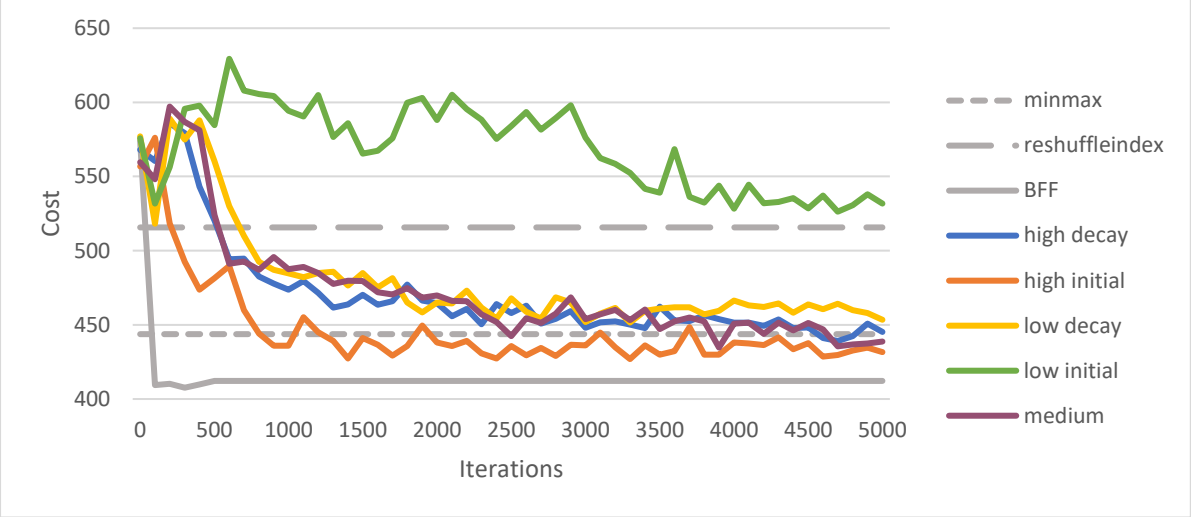


Figure 26 - Cost over iterations for the different learning rate settings. Heuristics and BFF added for comparison

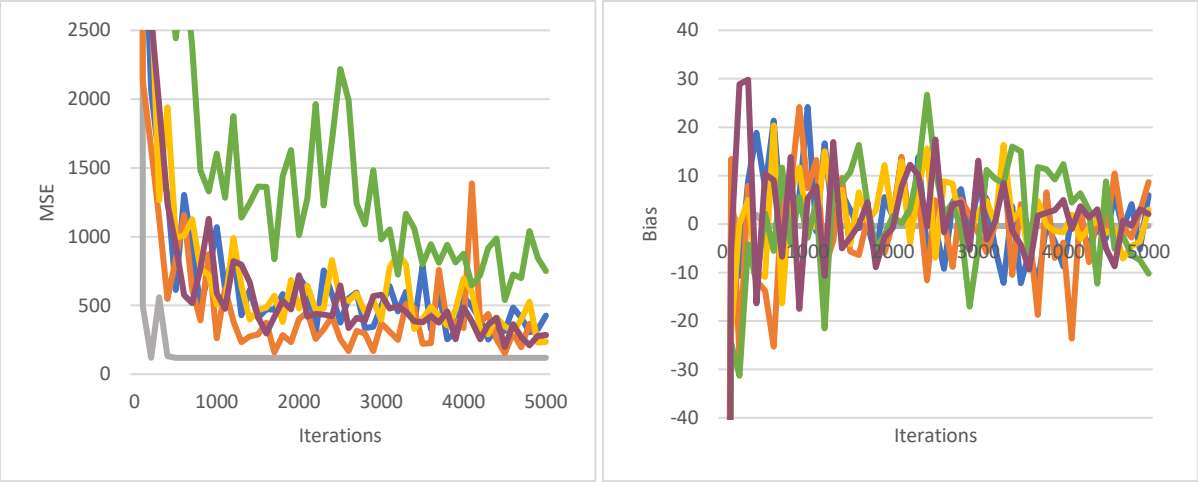


Figure 27 - MSE and Bias for different learning rate schedules for the Neural Network. BFF added for comparison

Figure 29 shows the cost over iterations for the different learning rate settings. All settings end up at a similar cost point except 'low initial', which did not manage to adjust its weights fast enough. This is also reflected in the MAE and R^2. All other settings end up around the same cost point, with 'high initial' scoring the best by a slight margin. However, all settings of the Neural Network perform worse than the Basis Feature Function, which additionally converges in $\frac{1}{10}^{th}$ of the iterations.

Of note is that occasionally the library that is used for updating the Neural Network fails to update, meaning that no learning takes place in that iteration. In testing, this happened 5%-10% of the time. This does not seem to have a significant impact on the performance of the algorithm.

5.10.2.Layers, and layer size

For the layer experiments, three options are explored: 1, 2 and 3 layers. These options have an increasing amount of nodes in their Neural Network. The option with a large amount of nodes still has an extremely low amount of nodes compared to modern reinforcement learning projects, which can have tenths to hundreds of layers with hundreds to thousands of nodes each, but this experiment is meant to explore the capabilities of small neural networks, not a deep learning approach. The following settings are used:

name	layer 1 size	layer 2 size	layer 3 size
1 layer	20	-	-
2 layers	40	20	-
3 layers	80	40	20

This leads to the following results:

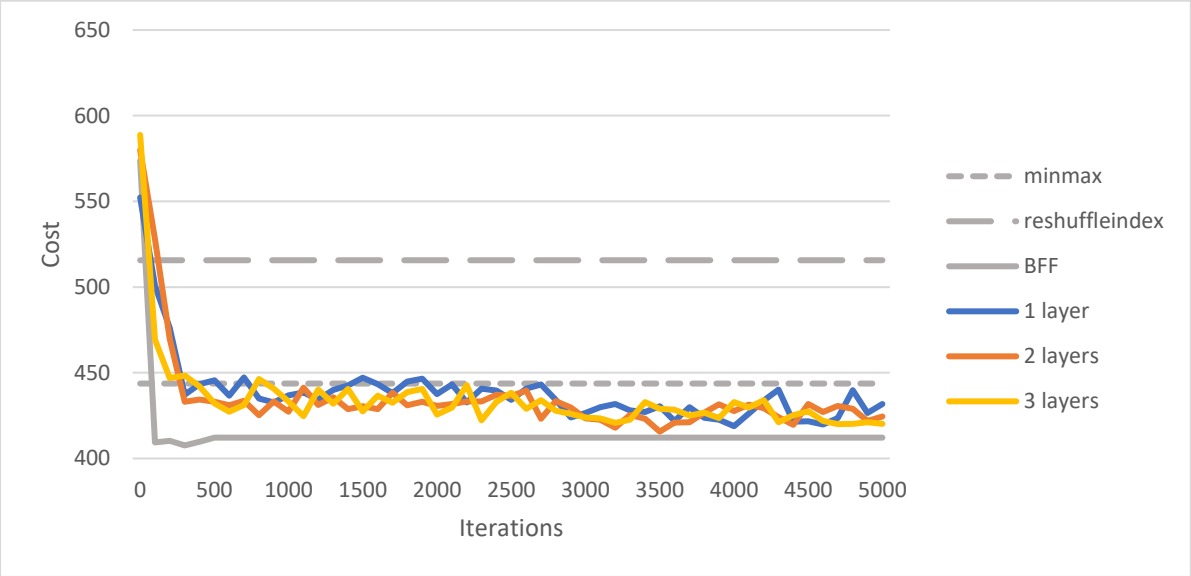


Figure 28 - Cost over iterations for different NN layer options.

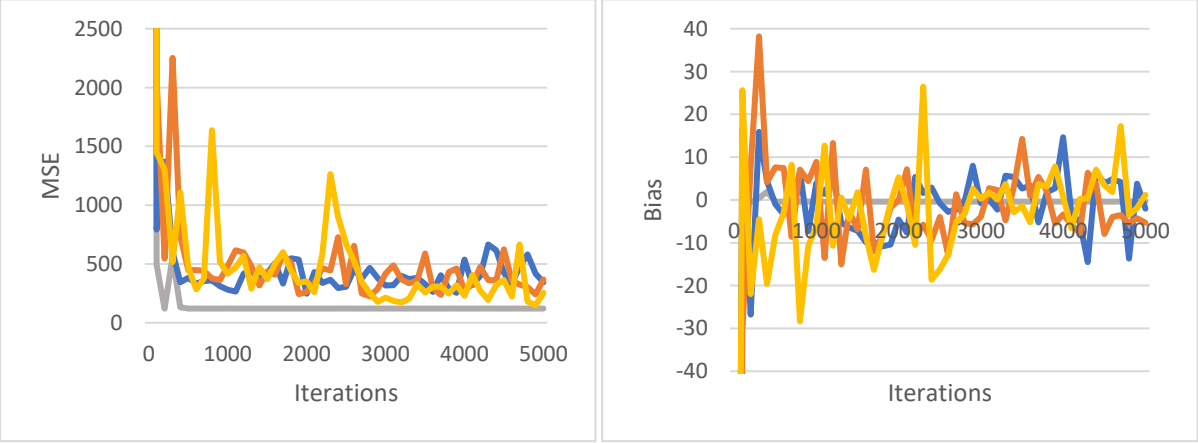


Figure 29 - R^2 and MAE over iterations for different NN layers

From Figure 28 we see that there is no significant difference in performance between the different layer options. Additionally, we see that neither option performs better than the BFF, even after 10 times the amount of training. The same goes for MSE and Bias, shown in Figure 29. While the Neural Networks is in essence not much different from the linear regression that the BFF uses, there are a number of key differences that can explain the difference in performance. The Neural Network has multiple layers and can thus have interaction between the activation of features. In exchange, the Neural Network has one regression for the entire problem instance, instead of one for each timestep, like the BFF. The methods for updating weights is also much different, with the BFF using Recursive Least-Squares (See Section 5.5), while the Neural Network uses Stochastic Gradient Descent, which has a much slower convergence rate.

5.11. Sensitivity Analysis

This section discusses the sensitivity of the produced algorithm to various aspects of the problem setting. We discuss sensitivity in performance to problem instances with differing characteristics, sensitivity to randomness, and sensitivity to different cost weights.

5.11.1. Sensitivity to characteristics of problem instances

As all problem instances were generated with distinctive characteristics, we can compare how the best version of the ADP algorithm in this research (the algorithm from the results in Section 5.4) compares to different heuristics in each problem instance. In Table 22, you can see the costs made in each problem instance by the ADP algorithm, the minmax heuristic and reshuffle-index heuristic. The last column shows the comparative costs of the ADP algorithm compared to the best performing heuristic (min-max or reshuffle-index, depending on the problem instance). As you can see, the ADP algorithm results in 3.2% higher costs to 13.6% lower costs, depending on the problem instance, with an average savings of 6.5%.

*Table 22 - Comparison of ADP algorithm performance to heuristics. *= savings are compared to best performing heuristic. **=total training time in minutes*

Problem instance	name	cost	minmax	reshuffle index	Savings*	training time**
0	normal	410.8	443.7	515.7	7.4%	381
1	low c	165.4	177.9	165.1	-0.2%	36
2	high c	811.6	839.0	1092.4	3.3%	3059
3	low p	224.7	248.1	254.5	9.4%	139
4	high p	905.9	1048.4	1247.5	13.6%	1707
5	short E[LoS]	258.0	295.7	361.8	12.7%	249
6	long E[LoS]	620.0	646.2	824.8	4.1%	622
7	Low E[cycles]	718.8	750.8	895.3	4.3%	569
8	High E[cycles]	194.6	188.5	217.1	-3.2%	203
9	low occupation	157.6	195.5	166.4	5.2%	84
10	high occupation	799.6	849.1	981.6	5.8%	496
11	short batch timespan	176.9	200.5	221.9	11.8%	226
12	long batch timespan	665.4	738.3	811.3	9.9%	423
average	-	469.9	509.4	596.6	6.5%	630

We test the variance in performance by generating another 9 problem instances with the settings of problem instance 0. Figure 30 shows the costs per category for these problem instances. The green line shows the costs made by the best performing heuristic, so we can see that the proposed solution consistently results in marginally lower costs. We can also see that especially the 'wrong-stack penalty'

soft-constraint cost is the most variable along problem instances, but that the total amount of costs can also vary a large amount. With such high variability in costs within the same settings for problem instances however, it is not possible to draw conclusions about changes in algorithm performance for different types of problems.

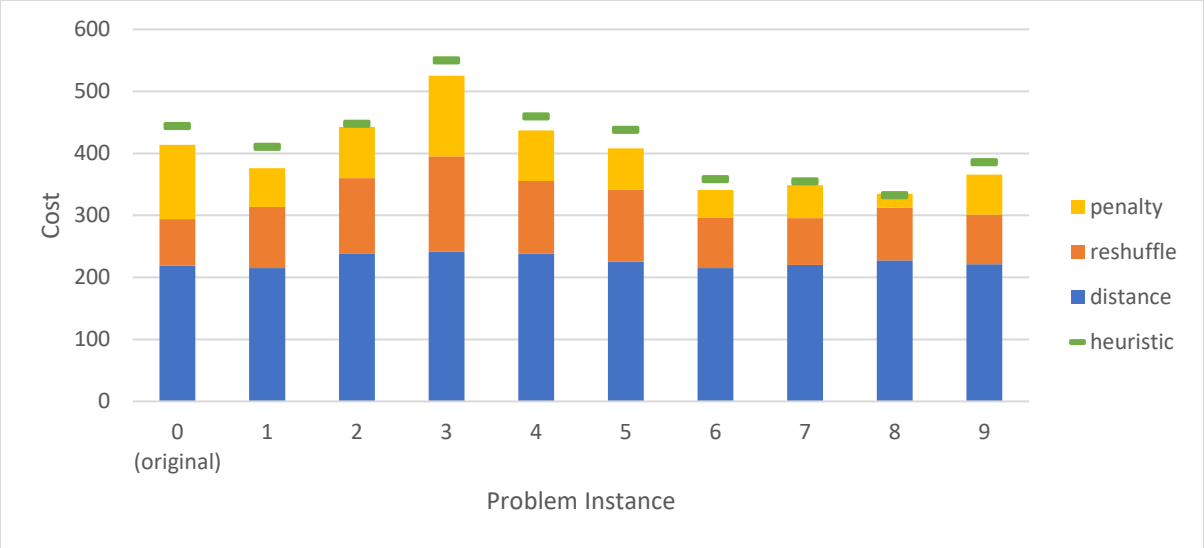


Figure 30 - Comparison of costs per category and costs of heuristic per problem instance

5.11.2. Sensitivity to randomness

We test the sensitivity of the algorithm to randomness by training it on a modified version of the problem instances. Instead of training the algorithm normally, the algorithm is trained using one of the five pre-drawn RNG samples used only for evaluation simulations. This means that every time, the algorithm will encounter the same revealed order within batches, turning this problem into a deterministic problem. These five trained algorithms from pre-drawn RNG samples are shown in Figure 31 in blue. Shown in red are the evaluation simulations using the same five pre-drawn RNG samples, but with an algorithm trained on the regular stochastic problem. The accuracy of the algorithms trained on the deterministic problem are averaged and shown in Figure 32, together with the accuracy of the normal algorithm.

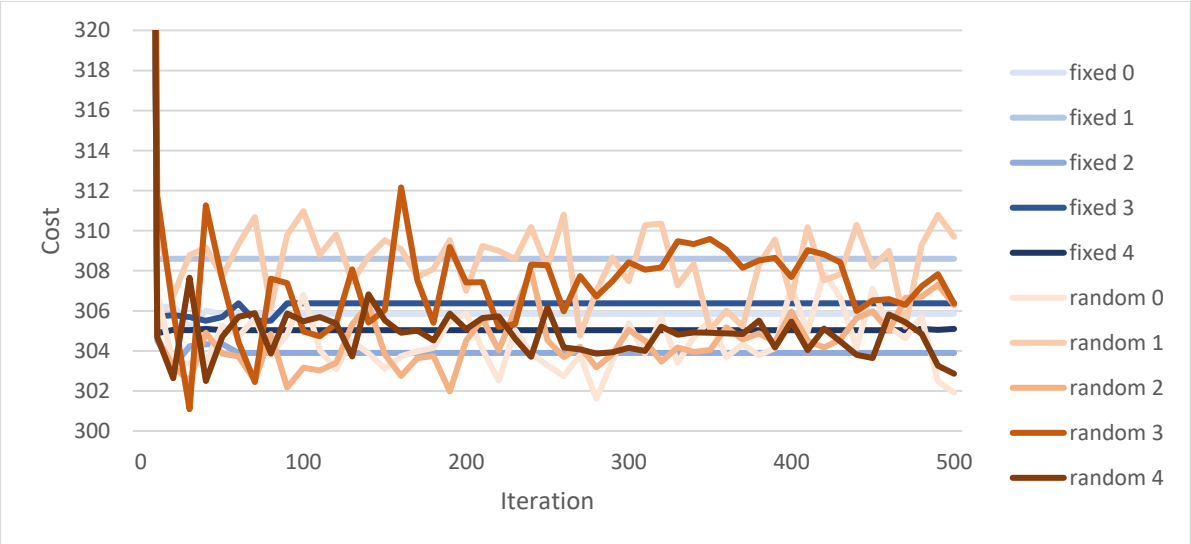


Figure 31 - Cost over iterations for the fixed RNG (in blue) and the normal RNG (in orange) on the pre-generated evaluation sample paths

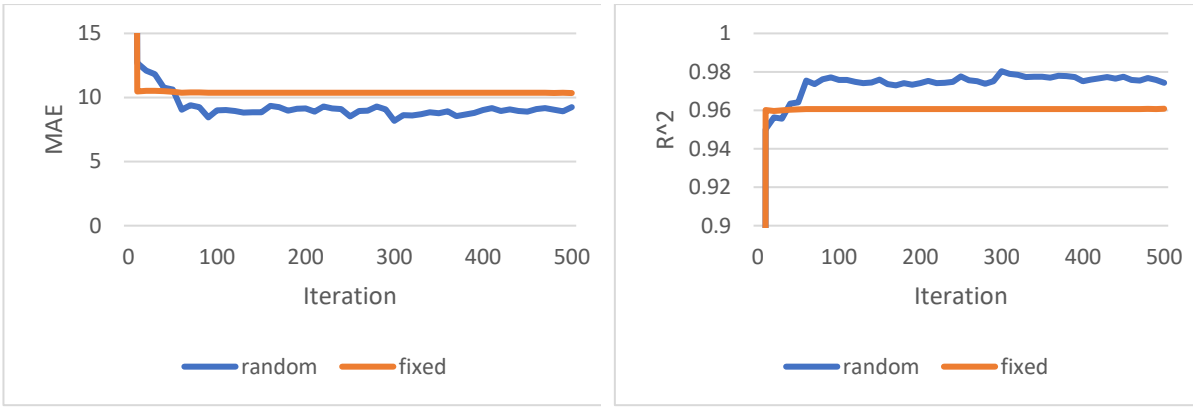


Figure 32 - MAE and R^2 over iterations for the fixed RNG and normal RNG

Surprisingly, the algorithm performs worse when the problem instances are converted to a deterministic problem. One would expect that the removal of randomness would make it possible to predict future costs more accurately, however the costs incurred by the algorithm trained on the regular stochastic problem performs as well as the deterministic variant and achieves equal accuracy.

5.11.3. Sensitivity to cost categories

Lastly, we compare the performance of the ADP algorithm when the distance, reshuffle and wrong-stack penalty costs are changed. In previous experiments the algorithm is tested with the following cost weights:

$$cost\ per\ m\ travelled = \alpha_1 = 0.06,$$

$$reshuffle\ cost = \alpha_2 = 2,$$

$$wrong\text{-}stack\ penalty = \alpha_3 = 8.$$

In this analysis the algorithm is tested with each cost either doubled or halved. This leads to the results shown in Table 16. From this we see that if any cost category weight increases or decreases, the costs made in that category simply increase by that amount, implying that the weight increase influences decisions made by the algorithm minimally. This effect is not as strong for the weight for the wrong-stacking penalty, with an increase of 83% when doubling weight α_3.

Table 23 - Changes in costs caused by different multi-objective cost weights

Cost type	Change	Category-specific cost change	Total change
distance	x2	1.95	1.53
	÷2	0.51	0.74
reshuffle	x2	2.01	1.28
	÷2	0.52	0.88
penalty	x2	1.83	1.17
	÷2	0.53	0.91

5.12. Conclusions

In this chapter, multiple experiments are done to determine the best settings for the ADP algorithm for the problem being solved in this research. These include feature sets, choice policy, updating schedule for feature weights, a corridor, and settings for the PST. General observations are that the algorithm settings that produce the most accurate estimation of future costs do not always correlate with the settings that produce the lowest costs. The best version of the algorithm, a BFF with a PST with 3 attempts, produces lower costs than the minmax heuristic on the problem instances on which it was tested (306 vs 334), but at the cost of a training time of several hours, while the heuristic can compute its solution in less than a minute. A simple Neural Network was also tested on the problem, but these required 10 times the training and performed similarly to the BFF.

A sensitivity analysis was also performed using different types of problem instances, different cost weights and the randomness in the problem. These analyses show that the performance of the ADP algorithm compared to heuristics can depend on the problem instance, but that the ADP algorithm performs relatively better in instances where containers have a shorter length-of-stay. The algorithm is insensitive to changes in the multi-objective weights of the cost function, choosing similar actions if the costs of a single cost weight are multiplied or divided by 2.

6. Conclusions, Discussion & Recommendations

This chapter provides the conclusions of this research, a discussion of the scientific contribution of this research, the limitations on this research and recommendations for the company Cofano and for future research.

6.1. Conclusions

In Section 1.5 we defined research questions to achieve the research objective:

> *"Develop a model that creates container handling schedules for a container terminal that minimizes the transportation costs and ship transshipment times, while taking long-term costs into account."*

In subsequent chapters these questions were answered. In the context analysis we identified the types of containers that are handled at a container terminal and which stacking restrictions this imposes. We identify that vessels entering the terminal experience delay, which results in incomplete information about the sequence in which containers arrive at and depart from the terminal. In the literature review we discussed several methods by which CRPs are solved, including Linear Programs for small problem instances and heuristic methods for larger instances. We discuss the concept of ADP and how it can be applied to the CRP. We then formalize a model of the problem in the form of a Markov Decision Problem and solve this problem using ADP. This solution method consists of two parts: an approximation of future costs $\bar{V}_t^x\left(S^{M,x}(S_t^n, x_t^n)\right)$ using a Basis Feature Function or Neural Network, and an optimization method for solving the Bellman equation:

$$\tilde{x}_t^n = \underset{x_t^n \in \mathcal{X}_t}{\arg\min} \left(C(S_t^n, x_t^n) + \gamma \bar{V}_t^x\left(S^{M,x}(S_t^n, x_t^n)\right)\right)$$

For the first part of the solution method, we develop a BFF that uses features that give an indication of future costs. We test different feature sets and different parameters for the ADP algorithm and find several sets that are best at the approximation. In addition, we test several parameters of the ADP algorithm, such as choice-policy, learning step size and pre-training using a heuristic. These tests do not provide conclusive results regarding which parameters result in the best performance. Lastly we also test several small Neural Networks, which perform at a similar level as the BFF, but which take several factors more computation time to train.

For the single-stage solution method we develop a Linear Program and a PST method. The PST resulted in an increase of 8% costs over the LP, which is an exact method. Overall, the ADP algorithm provides better performance than the min-max heuristic, which was the best performing heuristic we tested. We do think the method can be improved using adjustments in the problem formulation and solution method discussed in the following section.

6.2. Discussion

The contribution of this thesis builds on the research done by (Boschma, 2020), and expands on it in several ways. Firstly, the objective is expanded to a multi-objective function that can take into account distance as a cost, which more accurately reflects the costs made in a terminal. It is also flexible, as the algorithm performance is not sensitive to the weight values chosen (see Section 5.11.3). Additionally, the assumption that any batch of containers only contains incoming or outgoing containers is relaxed. This means that any timeframe where a number of containers arrive and depart can be represented in a batch, whereas previously these would need to be represented as two different batches, with all departing containers being handled before or after the arriving containers.

The model also accounts for the four different kinds of containers that constitute 99% of all containers arriving in the terminals of clients of Cofano.

The method for finding actions within a timestep is improved upon by expanding the partial search tree method. By creating a predetermined number of branches of the search tree, better action sets can be found, improving performance. Multiple new features are tested for the basis feature function of this model, of which multiple were included in the final feature set (TDLB, SSH, NIS1, NIS2, FOC1, MMV, √EBLB, √BD). Lastly, a simple Neural Network approach is tested, and found to perform subpar compared to a basis feature function. Performance might be improved by adding characteristics as input that are not valid as features for a BFF; terminal occupation, number of containers left in the schedule, amount of containers in the next batch are all examples of cost indicators that do not qualify as features, as these are not dependent on the actions at a single stage, as explained at the beginning of Section 4.4.3.

This research does have limitations. As discussed in Section 5.11.1, the algorithm generally performs better than the tested heuristics, but in some cases it does not, while requiring orders of magnitude more computation time than any heuristic. The mathematical model that is used as the problem on which the algorithm is trained also contains assumptions that may not be reflective of reality. One of these is that containers arrive in pre-determined batches, of which the order is unknown. This represents on one hand a strictness where containers can not arrive in a later or earlier timeslot, and on the other hand a complete lack of knowledge about the order of containers arriving within a timeslot. The information process in reality is more complicated; containers arriving far into the future are more uncertain, and some containers are guaranteed to arrive together as they arrive in the same boat or train.

Additionally, the model tries to keep 20ft and 40ft long containers in completely separate stacks by using a soft constraint penalty. While this is done in real life as well for practical purposes, two 20ft containers can stack upon one 40ft container, and vice versa. This allows for more flexible stacking, which this algorithm does not utilize at all. Containers other than 20HV, 40HV, 20RF and 40RF were not used at all. While containers other than these four make up <5% of all container traffic in the clients of Cofano, they still need to be allocated manually, as some of their restrictions (containers with chemicals have set locations, open top containers cannot have containers above them, etc.) are not included into the model. Thus, they would need to be scheduled manually.

Lastly, the algorithm made in this research is tested entirely on toy problem sizes. Due to a combination of poorly written code, a low amount of available computation power and the nature of some features, the algorithm could not be trained on realistically sized problem instances. This means that the added value of this algorithm is hard to quantify on real-life problems.

6.3. Recommendations

This section provides recommendations to Cofano and for future research. First, we give the recommendations to Cofano for providing optimization software for their clients. The algorithm developed in this research can provide better results than heuristics, but is severely limited by the computation time required to train the algorithm. This means that the algorithm cannot be used in practical scenarios, but several things can be done to improve this problem. The training time of this algorithm was set at 500 iterations for the BFF, but as Figure 22 and others show, almost all training happens in the first 50 iterations. If the algorithm is trained not for a set amount of iterations, but instead until it hasn't improved in the last i iterations (e.g., last 50 iterations), then the training can be stopped prematurely, saving a large amount of time. Additionally, the algorithm is written in Python, which is a flexible high-level programming language that is easy to program in, but loses in speed.

Lastly, the computation time consists for a large majority in computing features, which are calculated from scratch for each state. This can be done more efficiently by not calculating feature values from scratch, but wherever possible only calculating the change in value compared to the previous evaluated state using a change-list.

The second category of recommendations is for future research. These can be divided into improvements to the mathematical model that represents the real-life problem, and the algorithm that solves the mathematical model. First we discuss potential improvements to the mathematical model. While the model formulated in this research relaxes the previous assumption that containers in a batch may only be one type out of incoming or outgoing, containers are still divided into batches whose sequence is randomly generated. This means that on one hand the model is restrictive in that containers can not arrive a batch earlier or later, but naïve in the sense that the order of containers is assumed to be unknown until that batch is handled. The information process in real-life container terminals is more complex, as described in Section 2.5. A potential subject for future research is a change in the information process where the vessel a container has a planned arrival (including known delays), and where the vessel that containers arrive on is kept track of. Ships can have random planned and day-of-arrival delays, and trucks and trains can also have random amounts of small delays. The fact that trucks delivering containers do not adjust for delayed ships can be simulated this way, as well as that trucks that retrieve containers arriving by vessel will wait to be dispatched until the ship has arrived.

Another assumption of the model used in this research is that containers with 20ft length and 40ft length do not mix, and stacks are designated to either type at the start of the planning. Mixing stacks of 20ft and 40ft containers is avoided in real life, as it introduces complexities in stacking, but it does happen. More importantly, stacks aren't dedicated to 20ft or 40ft containers alone in advance, allowing for flexibility in storage. Both of these assumptions can be relaxed to more accurately describe reality, but the latter is most relevant. This can be done by making 40ft containers take up two adjacent stacks, which then are temporarily 'fused' for as long as there are 40ft containers lying on top of them.

Lastly, this research did not account for containers other than 20HV, 40HV, 20RF and 40RF containers, as mentioned in Section 6.2. Most relevantly, this research did not distinguish between full containers and empty containers. While filled containers have a particular destination, empty containers are mostly interchangeable between each other, as long as both belong to the same owner. Other than accounting for whether the container is filled, creating more types of containers and features that can account for them (i.e., an 'non-accessible stacks' feature to indicate which stacks have an open-top container on top of them, and thus can not receive more containers) is a way to work towards a mathematical model that represents real life more closely.

We also have recommendations with regards to potential future research regarding the algorithm. This research used a distance-based corridor to eliminate reshuffle locations that are not promising. There are other characteristics to eliminate destinations on other than distance however, such as the reshuffle-index heuristic or the min-max heuristic. Both of these can be used to rank destinations from best to worst, at which point the top x destinations can be used as potential destinations. Another option is to reduce the amount of times the VFA is used by changing the way the PST chooses potential action sets. By using heuristic evaluations as the decision metric at nodes in the decision tree, a large set of action sets can be created quickly. The post-decision states of the final nodes in the search tree can then be evaluated using a VFA.

Another subject for future research is to split out features into more detailed mini-features. A significant portion of the features used in this research are sums of characteristics (e.g., the sum of unordered stacks, the sum of blocking lower bounds over all stacks or containers, the sum of batch-label differences over all stacks, etc.). By splitting these sums out over individual features, a BFF can account for situations where the stacks height, blocking amount or any other characteristics is more important for stacks closer to the exit, or for stacks that can have more types of containers, etc. This will result in a BFF with orders of magnitude more features, which can reduce the convergence speed of the algorithm. But this would have to be found out.

The last and most impactful potential research subject is to train the VFA not on accuracy of estimated future costs, but on minimization of costs directly. This changes the purpose of the VFA from a function that estimates future costs, to a function that influences decision making in a single stage such that the overall costs are minimized. Hypothetically, it is preferable that a VFA wildly over- or underestimates future costs, if that results in a situation where low-quality actions are punished better. The motivation for this being a possibility is found in multiple experiments where the setting that minimized the costs the best did not predict future costs most accurately (this happens in Section 4, 5.5, 5.6, and 5.7.

This method can be achieved using a genetic algorithm; creating random mutations in feature weights, evaluating the costs made by the resulting algorithm, and keeping the best scoring mutations. This training method converges far slower than a directed optimization method, so this method can be used to refine a set of weights that has been pre-trained using a faster converging method such as Recursive Least Squares, used in this research.

7. References

Boschma, R. (2020). *Exploring the world of Container Stacking using Approximate Dynamic Programming*.

Caserta, M., Schwarze, S., & Voß, S. (2012). A mathematical formulation and complexity considerations for the blocks relocation problem. *European Journal of Operational Research*, *219*(1), 96–104. https://doi.org/10.1016/j.ejor.2011.12.039

Caserta, M., Voß, S., & Sniedovich, M. (2009). Applying the corridor method to a blocks relocation problem. *OR Spectrum 2009 33:4*, *33*(4), 915–929. https://doi.org/10.1007/S00291-009-0176-5

Committee, E. C. for E. I. T. (2021). *ADR 2021 Volume 1 | UNECE*. https://unece.org/transport/documents/2021/01/standards/adr-2021-volume-1

Forster, F., & Bortfeldt, A. (2012). A tree search procedure for the container relocation problem. *Computers & Operations Research*, *39*(2), 299–309. https://doi.org/10.1016/J.COR.2011.04.004

Galle, V. (2018). The Stochastic Container Relocation Problem. *Transportation Science*, *52*(5), 1035–1058. https://doi.org/10.1287/trsc.2018.0828

Güven, C., & Eliiyi, D. T. (2014). Trip Allocation and Stacking Policies at a Container Terminal. *Transportation Research Procedia*, *3*, 565–573. https://doi.org/10.1016/J.TRPRO.2014.10.035

Hakan Akyuz, M., & Lee, C. Y. (2014). A mathematical formulation and efficient heuristics for the dynamic container relocation problem. *Naval Research Logistics*, *61*(2), 101–118. https://doi.org/10.1002/NAV.21569

Huré, C., Pham, H., Bachouch, A., & Langrené, N. (2021). Deep Neural Networks Algorithms for Stochastic Control Problems on Finite Horizon: Convergence Analysis. *Https://Doi.Org/10.1137/20M1316640*, *59*(1), 525–557. https://doi.org/10.1137/20M1316640

Jiang, T., Zeng, B., Wang, Y., & Yan, W. (2021). A New Heuristic Reinforcement Learning for Container Relocation Problem. *Journal of Physics: Conference Series*, *1873*(1), 012050. https://doi.org/10.1088/1742-6596/1873/1/012050

Jin, B. (2020). On the integer programming formulation for the relaxed restricted container relocation problem. *European Journal of Operational Research*, *281*(2), 475–482. https://doi.org/10.1016/J.EJOR.2019.08.041

Jovanovic, R., Tanaka, S., Nishi, T., & Voß, S. (2019). A GRASP approach for solving the Blocks Relocation Problem with Stowage Plan. *Flexible Services and Manufacturing Journal*, *31*(3), 702–729. https://doi.org/10.1007/s10696-018-9320-3

Kim, K. H., & Hong, G.-P. (2006). A heuristic rule for relocating blocks. *Computers and Operations Research*, *33*(4), 940–954. https://doi.org/10.1016/j.cor.2004.08.005

Lehnfeld, J., & Knust, S. (2014). Loading, unloading and premarshalling of stacks in storage areas: Survey and classification. *European Journal of Operational Research*, *239*(2), 297–312. https://doi.org/10.1016/J.EJOR.2014.03.011

Li, W., Xiaoning, Z., & Zhengyu, X. (2019). Efficient container stacking approach to improve handling: efficiency in Chinese rail–truck transshipment terminals: *Https://Doi.Org/10.1177/0037549719843347*, *96*(1), 3–15. https://doi.org/10.1177/0037549719843347

Mes, M. R. K., & Rivera, A. P. (2017). Approximate Dynamic Programming by Practical Examples. *International Series in Operations Research and Management Science*, *248*, 63–101. https://doi.org/10.1007/978-3-319-47766-4_3

MFAME. (2019). *Port of Shanghai Rated World's Best Connected Port by UN*. https://mfame.guru/port-of-shanghai-rated-worlds-best-connected-port-by-un/

Powell, W. B. (2011). Approximate Dynamic Programming: Solving the Curses of Dimensionality: Second Edition. *Approximate Dynamic Programming: Solving the Curses of Dimensionality: Second Edition*, 1–638.

Ries, J., González-Ramírez, R. G., & Miranda, P. (2014). A Fuzzy Logic Model for the Container Stacking Problem at Container Terminals. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *8760*, 93–111. https://doi.org/10.1007/978-3-319-11421-7_7

RTVDrenthe. (2021). *Minder containers, maar toch wil haven Meppel investeren*. https://www.rtvdrenthe.nl/nieuws/168011/Minder-containers-maar-toch-wil-haven-Meppel-investeren

Ruder, S. (2016). *An overview of gradient descent optimization algorithms*. http://arxiv.org/abs/1609.04747

Scholl, J., Boywitz, D., & Boysen, N. (2017). On the quality of simple measures predicting block relocations in container yards. *Https://Doi.Org/10.1080/00207543.2017.1394595*, *56*(1–2), 60–71. https://doi.org/10.1080/00207543.2017.1394595

Si, J., Barto, A., Powell, W., & Wunsch, D. (n.d.). *Handbook of Learning and Approximate Dynamic Programming*. Retrieved June 28, 2021, from https://books.google.nl/books?hl=nl&lr=&id=JHvz1elubJQC&oi=fnd&pg=PA1&dq=advantages+approximate+dynamic+programming&ots=Ul0BH79mXY&sig=6wKqdDzpT4GqxwlGVp_LABxHETM&redir_esc=y#v=onepage&q=advantages approximate dynamic programming&f=false

Sikorra, J., Bohlken, W., & Goes, M. (2021). Allocation of container slots based on machine learning. *Hhla.De*, 11–15. https://hhla.de/fileadmin/module/its/ITS_Tech_Paper-HPC_machine_learning.pdf

Statista Research Department. (n.d.). *Projected size of the global shipping container market between 2019 and 2027*. Retrieved June 28, 2021, from https://www.statista.com/statistics/1097059/global-shipping-containers-market-size/

Statista Research Department. (2022). *No Title*. https://www.statista.com/statistics/913398/container-throughput-worldwide/

Steenken, D., Voß, S., & Stahlbock, R. (2004). Container terminal operation and operations research - a classification and literature review. *OR Spectrum 2004 26:1*, *26*(1), 3–49. https://doi.org/10.1007/S00291-003-0157-Z

Tang, L. (2015). Research into container reshuffling and stacking problems in container terminal yards. *IIE Transactions*, *47*(7), 751–766.

van Heeswijk, W., & La Poutré, H. (2019). *Approximate Dynamic Programming with Neural Networks in Linear Discrete Action Spaces*. https://arxiv.org/abs/1902.09855v1

Wan, Y., Liu, J., & Tsai, P.-C. (2009). The assignment of storage locations to containers for a container stack. *Naval Research Logistics (NRL)*, *56*(8), 699–713. https://doi.org/10.1002/nav.20373

Xu, S., Panwar, S. S., Kodialam, M., & Lakshman, T. V. (2020). Deep Neural Network Approximated

Dynamic Programming for Combinatorial Optimization. *Proceedings of the AAAI Conference on Artificial Intelligence*, *34*(02), 1684–1691. https://doi.org/10.1609/AAAI.V34I02.5531

Zhang, C., Guan, H., Yuan, Y., Chen, W., & Wu, T. (2020). Machine learning-driven algorithms for the container relocation problem. *Transportation Research Part B: Methodological*, *139*, 102–131. https://doi.org/10.1016/J.TRB.2020.05.017

Zhang, C., Wu, T., Kim, K. H., & Miao, L. (2014). Conservative allocation models for outbound containers in container terminals. *European Journal of Operational Research*, *238*(1), 155–165. https://doi.org/10.1016/J.EJOR.2014.03.040

Zhao, M., Zhang, Y., Ma, W., Fu, Q., Yang, X., Li, C., Zhou, B., Yu, Q., & Chen, L. (2013). Characteristics and ship traffic source identification of air pollutants in China's largest port. *Atmospheric Environment*, *64*, 277–286. https://doi.org/10.1016/J.ATMOSENV.2012.10.007

Zhu, W., Qin, H., Lim, A., & Zhang, H. (2012). Iterative deepening A* algorithms for the container relocation problem. *IEEE Transactions on Automation Science and Engineering*, *9*(4), 710–722. https://doi.org/10.1109/TASE.2012.2198642

8. Appendices

Appendix A – Linear Program Formulation

Situation is as described in Section 4.1 and 4.2. The LP is adapted from (Tang, 2015), but adapted to reduce the amount of constraints, fit the problem more to this problem setting, and the VFA including features has been added. The changes are as follows:

1. The original LP was a pure unloading CRP, where all containers were unloaded. This problem is a mixed loading and unloading problem where a large part of the containers is not unloaded. As such, each stage either a container i leaves or enters, and that is indicated using constraints 3 and 4.
2. The original LP contained variable W_{scij}, indicating that at stage s at column c, container j was above container i. Sets i, j and c are large, resulting in a large amount of variables and constraints related to that variable. It is rewritten to variable W_{sij}, which removes the index c. This requires more constraints per variable than W_{scij}, but if $|c| > 5$, our approach results in overall less constraints.
3. Distance is calculated by introducing variable T_{sice}, indicating that container i travelled from stack c to e at stage s. This variable is multiplied by the distance between c and e.
4. A dummy stage S^+ is introduced after all containers are handled, which indicates the post-decision state over which features can be calculated.

Sets:

- stage of the problem = s, with $s^- = \{0 \dots S\}$ and $s^+ = \{0 \dots S+1\}$, where S is the amount of containers that leave or enter in this batch. The last stage is a dummy stage which represents the post-decision state.
- index for each container $i \in \{0 \dots I\}$
    - Index starts at 0 for programming reasons. Containers are sorted in the order that they leave.
- index for columns in a bay = $c \in \{0 \dots C\}$
- tier of stack c = $p \in \{0 \dots P\}$
- number of features for the basis feature function = $n \in \{0 \dots N\}$

Constants:

- weights for the multi-objective function = $\alpha_1, \alpha_2, \alpha_3 = 2, 0.006, 8$
- distance between column c and e = $d_{ce} > 0$
- distance between column c and the designated exit for the container leaving at stage s = $do_c > 0$
- distance between column c and the entrance for the container entering at stage s = $de_c > 0$
- Initial layout of the terminal = $l_{icp} \in \{0,1\}$
- Penalty moves $pm_{ie} \in \{0\ if\ container\ i\ is\ allowed\ on\ stack\ e, 1\ otherwise\}$
- Incoming penalty move $ipm_{se} \in \{0\ if\ container\ arriving\ in\ stage\ s\ is\ allowed\ on\ stack\ e, 1\ otherwise\}$
- Id of container moving at stage s $id_s = \mathbb{Z}^+$
- Indicator whether a container moves out at stage s $os_s \in \{0,1\}$
- Big number $M = 99999$

Decision variables:

- $X_{sicp} = indicator\ whether\ container\ i\ is\ at\ stack\ c, tier\ p\ at\ beginning\ of\ stage\ s. X_{sicp} \in \{1\ if\ container\ is\ present, 0\ if\ not\}$
- $W_{sij} = indicator\ whether\ containers\ i\ and\ j\ are\ in\ the\ same\ column\ c, and\ j\ is\ positioned\ above\ i. W_{sij} \in \{0\ if\ not, 1\ if\ yes\}$
- $A_{sij} = integer\ representing\ the\ distance\ in\ stack\ id\ between\ container\ i\ and\ j\ at\ stage\ s.\ A_{sij} \geq 0$
- $AY_{sij} = auxillary\ variable\ to\ help\ determine\ A_{sij}. AY_{sij} \in \{0,1\}$
- $T_{sice} = indicator\ whether\ container\ i\ travelled\ from\ column\ c\ to\ column\ e\ at\ stage\ s.\ T_{sice} \in \{0, 1\}$
- $OT_{ic} = indicator\ that\ container\ i\ travelled\ to\ its\ exit\ from\ column\ c$
- $IT_{ic} = indicator\ that\ container\ i\ travelled\ from\ its\ entrance\ to\ column\ c$
- $U = distance\ travelled\ by\ all\ containers. U \geq 0$
- $C = variable\ for\ the\ direct\ contributions\ to\ the\ cost. C \geq 0$

Objective:

1. $\min z = C + \gamma * VFA$
   Minimize the direct costs plus the discounted estimated future costs
2. $C = \alpha_1 * \sum_{sice} T_{sice} + \alpha_2 * U + \alpha_3 * WSP$
   The direct costs are equal to the weighted sum of reshuffles, distance U and penalty WSP
3. $U = \sum_{s \in S^-, ice} T_{sice} * d_{ce} + \sum_{s \in S^-, e} OT_{se} * do_{se} + \sum_{s \in S^-, e} IT_{se} * de_{se}$
   Distance is equal to the amount of internal, outward, and inward distance travelled
4. $WSP = \sum_{sice} T_{sice} * pm_{ie} + \sum_{se} IT_{sce} * ipm_{se}$

   The 'wrong-stack penalty' is equal to the sum of times that container i was moved to stack *e* when container i is not allowed on stack e. na_{ie} is equal to 1 if it is not allowed, and 0 if not.
5. $VFA = \theta_1 * EBLB + \theta_2 * TDLB + \theta_3 * BD + \theta_4 * US + \theta_5 * BLD + \theta_6 * SSH + \cdots + \theta_7 * 1$

   The VFA is equal to the value of each feature multiplied by its weight

Constraints:

1. $\sum_{c=1}^{C} \sum_{p=1}^{P} X_{sicp} = 1\ if\ container\ needs\ to\ be\ in\ terminal\ at\ that\ time, 0\ otherwise, i \in I, s \in S^+$
   Every container needs to be somewhere at stage s IFF it should be in the terminal.
2. $X_{0icp} = l_{icp} \quad , \forall i, c, p$
   Fix the layout of containers at stage s=0 to the initial layout of the problem L_{icp}
3. $\forall i, s \in S^-, if\ i = id_s\ and\ s = 1$:
   - $X_{sicp} - X_{s+1,icp} \leq \sum_e T_{sice} + OT_{ic} \quad , c \in C, p \in P$

   $else$:
   - $X_{sicp} - X_{s+1,icp} \leq \sum_e T_{sice} \quad , c \in C, p \in P$

   Any container can only 'disappear' from one place if it gets moved from that place, or gets removed
4. $\forall i, s \in S^-, if\ i = id_s\ and\ s = 0$:
   - $X_{s+1,icp} - X_{sicp} \leq \sum_e T_{siec} + IT_{ic} \quad , c \in C, p \in P$

   $else$:
   - $X_{s+1,icp} - X_{sicp} \leq \sum_e T_{siec} \quad , c \in C, p \in P$

Any container can only 'appear' in one place if it is the destination of a movement T or if it is moved into the terminal

5. $\sum_{i=s}^{S} X_{sicp} \leq \sum_{i=s}^{S} X_{sicp-1} \quad , s \in S^{+}, c \in C, 1 \leq p \leq P$
   Any container not on p=0 may only be there if there is a container beneath it.
6. $T_{sicc} = 0 \quad , s \in S^{-}, i \in I, c \in C$
   Travel from/to the same container is not allowed
7. $\sum_{c} OT_{sc} = 1 \ if\ a\ container\ leaves\ during\ problem, 0\ otherwise \quad , s \in S^{-}$
   Exactly 1 container leaves during the appropriate stage
8. $\sum_{c} IT_{sc} = 1 \ if\ a\ container\ enters\ during\ problem, 0\ otherwise \quad , s \in S^{-}$
   Exactly 1 container leaves during the appropriate stage
9. $A_{sij} \geq \sum_{cp} c * X_{sicp} - \sum_{cp} c * X_{sjcp} \quad , s \in S^{+}, i \in I, j \in \{I \neq i\}$
10. $A_{sij} \geq \sum_{cp} c * X_{sjcp} - \sum_{cp} c * X_{sicp} \quad , s \in S^{+}, i \in I, j \in \{I \neq i\}$
11. $A_{sij} \leq \sum_{cp} c * X_{sicp} - \sum_{cp} c * X_{sjcp} + M * AY_{sij} \quad , s \in S^{+}, i \in I, j \in \{I \neq i\}$
12. $A_{sij} \leq \sum_{cp} c * X_{sicp} - \sum_{cp} c * X_{sjcp} + M * (1 - AY_{sij}) \quad , s \in S^{+}, i \in I, j \in \{I \neq i\}$
    Setting the value of A_{sij}, which gives the absolute difference in column id between container i and j.
13. $W_{sij} \geq \left(\sum_{cp} p * X_{sjcp} - \sum_{cp} p * X_{sicp}\right)/M - A_{sij} \quad , i \in I, j \in \{I \neq i\}, s \in S^{+}$
    W_{sij} is higher than 0 if tier of j is larger and i, and difference in columns is 0 (they are in the same column). Practically, this means that container j blocks container i.
14. $W_{sij} \leq 1 - \frac{A_{sij}}{M} \quad , s \in S^{+}, i \in I,\ j \in \{I \neq i\}$
    W_{sij} is 0 if the difference in columns is larger than 0.
15. $W_{sij} \leq 1 - \left(\sum_{cp} p * X_{sicp} - \sum_{cp} p * X_{sjcp}\right)/M \quad , s \in S^{+}, i \in I,\ j \in \{I \neq i\}$
    W_{sij} is 0 if the tier of container j is lower than the tier of container i.
16. $W_{s+1,ij} \leq 2 - \sum_{c} \sum_{e} T_{sice} - \sum_{c} W_{sij} \quad , i \in I, j \in \{I \neq i\}, s \in S^{-}$
    If W_{sij} is 1 (so container j is above i on the same stack) and container i moves, then $W_{s+1,ij}$ must be 0 (as that would mean that both are moved towards the same stack, and then j must be under i).
17. $\sum_{i=s}^{S} X_{sicp} \leq 1 \quad , s \in S^{+}, c \in C, p \in P$
    There is always at most 1 container per spot.
18. $\sum_{e} \sum_{c} T_{sice} = W_{sui} \quad , i \in I, S^{-},\ u = container\ leaving\ at\ stage\ s$
    Do internal travel IFF you are blocking the currently leaving container.

VFA-Related Constants:

- $\theta_n = weight\ of\ feature\ n\ in\ the\ basis\ feature\ function$
- $B_i = leaving\ batch\ of\ container\ i$
- $exitdistance_{ic} = distance\ to\ its\ designated\ exit\ for\ container\ i\ from\ stack\ c$
- $sqd_p = difference\ in\ squared\ stack\ height\ per\ additional\ p$
  - so sqd_p = {1, 3, 5, 7, …, 2p+1}, assuming p starts at 0
- $a = set\ of\ incoming\ containers\ in\ batch\ t+1$
- $it_a = set\ of\ all\ stacks\ where\ container\ a\ is\ allowed$
- $b = set\ of\ outgoing\ containers\ in\ batch\ t+1$
- $B_{max} = batch\ of\ latest\ departing\ container\ at\ post-decision\ stage$
- $B_{min} = batch\ of\ earliest\ departing\ container\ at\ post-decision\ stage$

VFA-Related variables:

- VFA = Value-Function Approximation; the estimation of the value of the post-decision state.
    - VFA is continuous, with a lower bound on 0
- EBLB = Expected Blocking Lower Bound; the sum of the blocking number of all containers
    - EBLB is integer, with a lower bound of 0
- TDLB = Travel Distance Lower Bound; the sum of the distance between the location of all containers to their designated exits
    - TDLB is continuous, with a lower bound of 0
- BD_j = Blocking Degree; the largest batch difference between container j and any container i it blocks
    - BD_j is integer, with a lower bound of 0
- TBD = Total Blocking Degree; the sum of blocking degree for all containers
    - TBD is integer, with a lower bound of 0
- US_c = indicator whether stack c is ordered or not, with US=1 if it is ordered, and 0 if it is not
    - $US_c \in \{0, 1\}$
- TUS = Total Unordered Stacks; the sum of unordered stacks
    - TUS is integer, with a lower bound of 0 and an upper bound of |C|
- BLD_{cp} = Batch Label Difference; absolute batch label difference between the container at (c,p) and (c,p+1)
- TBLD = Total Batch Label Difference; the sum of all BLDs
- NES = Non-Empty Stacks; the amount of non-empty stacks in the terminal
- SSH = Squared Stack Height; the sum of all squared stack heights in the terminal
- AS1 = available spots 1; the squared percentage of used spots for container type 1 (also available for other types of containers)
- LAS = Least Available Spots; the highest value of AS1, AS2, etc.; giving the most constrained percentage of container spots.

VFA-Related Constraints:

1. $LBL_{c0} = M$
2. $LBL_{cp} \leq \sum_j B_j * X_{S,jck} \quad , \forall c, 1 \leq p \leq P, 0 \leq k < p$
3. $LBL_{cp} \geq \sum_j B_j * X_{S,jck} - M * (1 - LBLb_{cpk})$
4. $\sum_k LBLb_{cpk} = 1$
   Lowest Batch Label LBL_{cp} is the lowest batch label of all containers underneath it.
5. $PB_{c0} = 0 \quad , \forall c$
6. $DB_{c0} = 0 \quad , \forall c$
7. $PB_{cp} \geq (\sum_i \left((B_i + 1) * X_{S,icp}\right) - LBL_{cp})/M \quad , \forall cp$
8. $PB_{cp} \leq 1 + \left(\sum_i \left(B_i * X_{S,icp}\right) - LBL_{cp}\right)/M \quad , \forall cp$
9. $DB_{cp} \geq (\sum_i \left(B_i * X_{S,icp}\right) - LBL_{cp})/M \quad , \forall cp$
10. $DB_{cp} \leq 1 + \left(\sum_i \left((B_i - 1) * X_{S,icp}\right) - LBL_{cp}\right)/M \quad , \forall cp$
    Introduce 'possibly blocking' PB_{cp} and 'definitively blocking' DB_{cp}, which indicate if the container on c, p is (possibly) blocking another container.
11. $EBLB = 0.5 * \sum_{cp} PB_{cp} + 0.5 * \sum_{cp} DB_{cp}$
    $EBLB$ is equal to half of the sum of PB's and DB's.
12. $TDLB = \sum_{i,c,p} X_{S^+,icp} * exitdistance_{ic}$
    TDLB is equal to the minimum total distance all containers must travel to their exits
13. $BD_{cp} \geq \sum_i B_i * X_{S^+,icp} - LBL_{cp} \quad , \forall cp$
14. $BD_{cp} \leq \sum_i B_i * X_{S^+,icp} - LBL_{cp} + \left(1 - PB_{cp}\right) * M \quad , \forall cp$

15. $BD_{cp} \leq PB_{cp} * M \quad , \forall cp$
    Blocking degree BD_{cp} is the amount of batches by which the container at (c, p) blocks any container below it.
16. $BDT = \sum_{cp} BD_{cp}$
    Total blocking degree is the sum of BD over all (c,p)
17. $US_c \geq \frac{\sum_p PB_{cp}}{M} \quad , \forall c$
18. $US_c \leq \sum_p PB_{cp} \quad , \forall c$
19. $UST = \sum_c US_c$
    Introduce US_c, which is a binary variable that is 1 if there is any possible blocking in that stack, and 0 otherwise. UST is the sum of all unordered stacks.
20. $BLD_{cp} \geq \sum_i B_i * X_{S^+,icp} - \sum_j B_j * X_{S^+,jc,p-1} \quad , \forall cp$
21. $BLD_{cp} \geq \sum_i B_i * X_{S^+,ic,p-1} - \sum_j B_j * X_{S^+,jcp} - M + M * \sum_j X_{S,jcp} \quad , \forall cp$
22. $BLD_{cp} \leq \sum_i B_i * X_{S^+,icp} - \sum_j B_j * X_{S^+,jc,p-1} + M * BLDY_{cp} \quad , \forall cp$
23. $BLD_{cp} \leq \sum_i B_i * X_{S^+,ic,p-1} - \sum_j B_j * X_{S^+,jc,p} + M * \left(1 - BLDY_{cp}\right) \quad , \forall cp$
24. $BLD_{cp} \leq M * \sum_i X_{S^+,icp} \quad , \forall cp$
25. $BLDT = \sum_{cp} BLD_{cp}$
    Introduce BLD_{cp}, indicating the absolute difference in batch between the container at (c,p) and (c,p-1). $BLDT$ is the sum of all batch label differences
26. $SSH = \sum_{icp} sqd_p * X_{S^+,icp}$
    Squared Stack height (SSH) is the sum of the square of each stack height. sqd_{cp} is the derivative in value of the squared stack height between (c,p-1) and (c,p).
27. $NIS_{a,it_a} \geq \sum_{it_a} X_{S,a,it_a,P} / M \quad , \forall a, it_a$
28. $NIS_{a,it_a} \geq \left(ib_a - LBL_{it_a}\right)/M \quad , \forall a, it_a$
29. $NIS_{a,it_a} \leq 1 - \sum_{it_a} X_{S,a,it_a,P} / M + NISb_{a,it_a} \quad , \forall a, it_a$
30. $NIS_{a,it_a} \leq 1 - \left(ib_a - LBL_{it_a}\right)/M + NISb_{a,it_a} \quad , \forall a, it_a$
31. $NIS = \sum_a \sum_{it_a} NIS_{a,it_a}$
    NIS1 is for each incoming container in batch t+1 the amount of stacks where container type x is allowed, that either 1) are full, or 2) have a LBL lower than the container in question. Introduce NIS_{a,it_a}, which indicates that for container a, which is allowed in stack it_a, stack it_a is 'not ideal'.
32. $FOC = \sum_c d_{ce} * X_{S^+,bcp} + \sum_j W_{S,bj}$
    Future outgoing costs is equal to the distance costs + reshuffle costs of containers b. Any container on top of container b is going to be reshuffled next timestep, so using W is sufficient.
33. $MMV_{c0} = \sum_i X_{S,ic0} * B_{max} + 1$

    Min-max value is equal to $B_{max} + 1$ for any container on p=0

34. $MMVY_{cp} \geq LBL_{cp} - \sum_i B_i * X_{S,icp} - M * \left(1 - \sum_i X_{S,icp}\right) \quad , \forall c, p > 0$
35. $MMVY_{cp} \leq M * \left(1 - PB_{cp}\right) , \forall c, p > 0$
36. $MMVY_{cp} \leq LBL_{cp} - \sum_i B_i * X_{S,icp} + M * DB_{cp} , \forall c, p > 0$
37. $MMVY_{cp} \leq M * \sum_i X_{S,icp} \ , \forall c, p > 0$
    If container at place (c,p) is not blocking a container, $MMVY_{cp}$ is equal to the difference in batch between the container at (c,p) and the earliest departing container. Otherwise, it is 0.
38. $MMV_{cp} = (B_{max} - B_{min}) * PB_{cp} + BD_{cp} + MMVY_{cp} , \forall c, p > 0$
    The min-max value for the spot (c,p>0) is either the batch difference between the earliest and latest container + the blocking degree of the container it is blocking, or the value for MMVY if it is not blocking.

39. $MMVT = \sum_{cp} MMV_{cp}$
    The total Min-Max Value is the sum of values over all (c,p)
    Heuristics such as the min-max heuristic or the reshuffle-index heuristic are not possible in a LP.
40. $EBLBY_z \in \{0,1\}$
41. $\sum_z EBLBY_z = 2 * EBLB$
42. $EBLBY_z \geq EBLB_{z+1}$
43. $EBLBSQ = \sum_z EBLBY_z * sqd_z$
44. $EBLBSQRT = \sum_z EBLBY_z * sqrt_z$
    $EBLBSQ$ is the squared value of EBLB, and $EBLBSQRT$ is the square-rooted value of EBLB. EBLB increases in steps of 0.5, so each $EBLBY_z$ represents an increase of 0.5, and sqd_z and $sqrt_z$ represent the derivative between z and $z-1$ of the squared and square-rooted value respectively.
45. $\sum_{zz} BDYY_{zz} = BDT$
46. $BDYY_{zz} \geq BDYY_{zz+1}$
47. $BDSQRT = \sum_z BDYY_{zz} * sqrt_{zz}$
    $BDSQRT$ is the square-rooted value of BD.
48. $C = 1$
    Constant

Appendix B – Problem Instance Generation Script

0. Input:
    - Nr of stacks C
    - Nr of tiers P
    - E[cycles] $E[cycles]$
    - LoS in hours LoS_{time}
    - Avg occupation $E[occ]$
    - Batch timespan $time_{batch}$
    - Fraction 20ft $f[20ft]$
    - Fraction reefer $f[reefer]$
1. Create Batch List
    a. Create x containers equal to $terminal\ capacity * E[occ]$
    b. Determine the total amount of containers in the planning horizon
    c. Max timestep is equal to $LoS_{time} * E[cycles] * time_{batch}$
    d. $LoS_{batch} = \frac{LoS_{hours}}{time_{batch}}$
    e. $E[batch\ size] = \frac{2}{LoS_{batch}} * capacity * E[occ]$
    f. For each timestep, starting at t=0:
        i. Nr incoming containers is drawn from Poisson distribution with as average $\frac{1}{2}E[batch\ size] + \frac{1}{2}(capacity * E[occ] - currently\ present\ containers)$
        ii. Nr leaving containers is drawn from Poisson distribution with average $\frac{1}{LoS_{batch}} * nr\ of\ containers\ currently\ in\ terminal$
        iii. A number of containers equal to the amount leaving and arriving are marked to depart in this timestep
2. Create Container Allowance
    a. Determine fraction of containers of type 0, 1, 2, 3 arriving at terminal
    b. Divide stacks in terminal up in these fractions evenly, rounded down
    c. Remaining stacks are allocated to the container type with least allocated stacks
3. Create Initial Layout and container type allocations
    a. All containers that are present at t=0 are randomly allocated to a stack and are given the container type that matches with the allowed containers on that stack
    b. For each timestep:
        i. Check available empty spots in the terminal of each type
        ii. Allocate each arriving container at time 1 according to the proportion of available spots at that time
        iii. Add available spots of appropriate type for each departing container
4. Create entrances and distance matrix
    a. Create stack locations according to uniform distribution within [0, 200] x value and [0, 300] y value. Create entrance locations along the border of this region.
    b. Calculate distance matrix according to
5. Create sample realizations
    a. For each timestep:
        i. Create 5 random shuffle orders equal to the size of the batch at timestep t

Appendix C – Problem Instance Information

Table 24 – Information of containers at t=0 of problem instance 0

ID	type	departure batch	order within departure	arrival batch	order within arrival	exit ID	starting stack	starting tier
0	0	45				22	0	0
1	0	74				22	3	0
2	0	15				20	3	1
3	0	43				22	11	0
4	0	39				22	2	0
5	0	27				22	5	0
6	1	19				22	12	0
7	0	24				20	0	1
8	2	15				22	15	0
9	0	95				20	4	0
10	0	4				21	8	0
11	2	114				20	14	0
12	2	0				21	16	0
13	2	53				21	16	1
14	0	29				20	2	1
15	3	45				21	19	0
16	0	153				21	11	1
17	0	31				20	1	0
18	2	1				20	15	1
19	0	67				22	1	1
20	0	86				22	11	2
21	2	6				20	18	0
22	2	42				21	16	2
23	0	2				20	11	3
24	1	75				22	12	1
25	2	162				22	16	3
26	0	17				22	9	0
27	3	112				22	19	1
28	0	30				21	5	1
29	2	70				22	18	1
30	3	25				22	19	2
31	1	55				21	13	0
32	1	14				21	13	1
33	0	28				21	10	0
34	0	42				20	2	2
35	0	17				21	3	2
36	3	16				22	19	3
37	0	79				21	8	1
38	2	45				20	14	1
39	2	26				20	17	0
40	2	82				22	15	2

41	2	8				21	14	2
42	0	102				21	2	3
43	2	24				21	17	1
44	1	13				21	13	2
45	0	195				20	7	0
46	0	9				21	10	1
47	0	10				21	3	3
48	0	158		0		21	21	
49	0	14		0		21	21	
50	0	59		1		22	20	
51	0	11		1		20	22	
52	0	67		1		21	21	
53	0	34		2		22	21	
54	0	55		2		22	22	
55	2	37		3		20	20	
56	0	26		3		20	20	
57	0	8		3		21	22	
58	0	9		5		21	22	
59	2	8		7		20	21	
60	2	78		7		21	20	
61	0	24		7		20	20	
62	2	44		7		21	20	
63	0	12		8		21	20	
64	0	30		9		20	20	
65	1	21		11		22	22	
66	0	46		11		22	21	
67	0	73		12		21	21	
68	0	90		14		22	20	
69	1	15		14		22	20	
70	0	36		15		20	22	
71	0	79		15		22	20	
72	2	66		15		21	20	
73	2	63		16		22	22	
74	2	150		17		20	22	
75	0	79		17		22	22	
76	2	147		17		20	21	
77	0	62		17		22	21	
78	2	186		18		21	20	
79	2	35		18		21	22	
80	0	63		18		22	21	
81	1	114		18		20	21	
82	0	161		19		22	20	
83	0	132		19		22	20	
84	2	91		19		20	21	
85	0	81		19		21	22	
86	2	47		20		21	21	

87	0	51		23		21	20	
88	0	30		23		22	21	
89	0	72		23		22	21	
90	0	64		24		21	22	
91	0	45		25		22	20	
92	0	75		26		20	20	
93	0	124		26		21	20	
94	0	98		28		21	21	
95	1	146		28		21	21	
96	0	59		29		21	20	
97	2	130		30		20	21	
98	2	67		31		22	22	
99	0	38		32		20	22	
100	0	48		32		21	21	
101	0	72		32		20	20	
102	1	58		33		21	21	
103	0	45		34		21	22	
104	3	57		35		20	20	
105	1	80		35		22	22	
106	0	70		35		20	21	
107	0	178		37		20	21	
108	2	84		38		20	20	
109	2	149		38		22	21	
110	0	47		38		21	20	
111	0	41		39		20	20	
112	0	133		39		22	21	
113	0	64		39		20	22	
114	0	85		39		22	21	
115	0	42		40		22	21	
116	0	49		42		21	22	
117	0	140		42		21	21	
118	0	105		43		21	22	
119	2	58		43		20	22	
120	0	99		43		22	21	
121	0	76		44		20	21	
122	0	121		45		20	21	
123	0	140		45		22	22	
124	0	79		46		21	21	
125	0	62		47		20	22	
126	1	78		48		22	20	
127	0	125		48		20	21	
128	2	103		48		22	20	
129	0	73		49		22	21	
130	0	106		49		22	22	
131	0	76		50		21	22	
132	0	59		50		21	22	

133	0	155		51		21	21	
134	3	142		51		21	20	
135	2	121		53		22	22	
136	2	58		54		20	22	
137	0	174		56		20	20	
138	2	118		57		20	21	
139	0	82		59		22	20	
140	2	163		59		21	22	
141	0	68		61		21	22	
142	0	83		62		20	21	
143	2	170		64		21	22	
144	2	92		64		22	21	
145	0	101		65		22	22	
146	0	89		66		20	22	
147	0	147		66		22	22	
148	0	82		68		22	22	
149	2	89		69		20	21	
150	2	96		70		21	20	
151	0	94		70		20	21	
152	0	189		72		21	22	
153	2	97		72		21	22	
154	0	78		72		20	21	
155	0	183		73		22	20	
156	0	77		73		20	20	
157	0	87		74		21	21	
158	0	104		74		20	20	
159	0	87		75		22	21	
160	0	81		75		20	20	
161	0	116		75		21	21	
162	0	167		78		21	20	
163	0	102		79		20	22	
164	2	120		79		20	21	
165	0	135		80		20	20	
166	0	148		80		21	21	
167	0	86		82		21	22	
168	0	118		83		21	20	
169	2	91		88		21	20	
170	2	138		88		22	22	
171	0	129		88		22	20	
172	0	103		89		20	20	
173	1	117		90		20	20	
174	0	97		90		21	22	
175	0	105		92		20	22	
176	0	102		92		20	20	
177	0	116		95		21	22	
178	1	116		96		22	22	

179	0	191		97		21	22	
180	0	114		98		22	20	
181	0	151		99		21	22	
182	2	192		100		22	22	
183	0	106		100		21	20	
184	0	173		101		22	20	
185	2	130		101		22	21	
186	0	118		103		21	22	
187	0	124		103		20	20	
188	2	119		104		20	22	
189	0	114		105		22	22	
190	0	133		107		20	22	
191	0	165		109		20	20	
192	2	151		110		20	22	
193	0	123		110		20	20	
194	0	179		112		22	22	
195	2	148		113		21	21	
196	0	120		114		20	21	
197	0	169		115		21	20	
198	0	156		116		21	22	
199	2	130		118		20	20	
200	0	194		118		21	21	
201	1	141		119		22	22	
202	0	127		120		21	22	
203	0	123		122		20	20	
204	2	184		123		20	20	
205	0	193		125		22	22	
206	1	166		126		22	21	
207	0	140		127		20	22	
208	0	131		128		22	21	
209	0	190		129		20	21	
210	0	175		130		21	20	
211	0	156		131		20	22	
212	2	188		131		20	21	
213	0	193		132		21	20	
214	2	144		134		22	21	
215	2	197		135		21	22	
216	2	196		136		22	20	
217	0	148		137		20	21	
218	1	177		139		22	22	
219	0	148		140		20	20	
220	2	180		141		21	20	
221	0	160		142		20	21	
222	2	191		143		21	21	
223	0	192		144		20	21	
224	0	194		145		20	21	

225	2	182		146		20	21	
226	0	198		147		21	22	
227	1	187		148		22	22	
228	0	185		149		21	20	
229	0	197		150		22	21	
230	2	164		151		20	21	
231	0	172		152		22	21	
232	0	198		153		22	22	
233	0	171		154		22	22	
234	0	176		155		22	22	
235	0	196		156		21	20	
236	0	181		157		20	21	
237	2	168		158		22	22	
238	0	190		159		20	21	
239	1	195		159		21	22	

Table 25 - Distance matrix of problem instance 0

0.00E+00	3.32E+01	3.02E+01	6.13E+01	1.35E+02	1.58E+02	4.46E+01	8.58E+01	7.62E+01	3.12E+01
3.32E+01	0.00E+00	4.49E+01	9.35E+01	1.17E+02	1.82E+02	1.20E+01	1.00E+02	6.83E+01	6.36E+01
3.02E+01	4.49E+01	0.00E+00	5.96E+01	1.60E+02	1.81E+02	5.62E+01	1.15E+02	1.04E+02	3.59E+01
6.13E+01	9.35E+01	5.96E+01	0.00E+00	1.89E+02	1.43E+02	1.05E+02	1.07E+02	1.27E+02	3.01E+01
1.35E+02	1.17E+02	1.60E+02	1.89E+02	0.00E+00	1.93E+02	1.08E+02	1.11E+02	6.30E+01	1.62E+02
1.58E+02	1.82E+02	1.81E+02	1.43E+02	1.93E+02	0.00E+00	1.88E+02	8.97E+01	1.49E+02	1.47E+02
4.46E+01	1.20E+01	5.62E+01	1.05E+02	1.08E+02	1.88E+02	0.00E+00	1.04E+02	6.48E+01	7.53E+01
8.58E+01	1.00E+02	1.15E+02	1.07E+02	1.11E+02	8.97E+01	1.04E+02	0.00E+00	5.94E+01	9.25E+01
7.62E+01	6.83E+01	1.04E+02	1.27E+02	6.30E+01	1.49E+02	6.48E+01	5.94E+01	0.00E+00	1.01E+02
3.12E+01	6.36E+01	3.59E+01	3.01E+01	1.62E+02	1.47E+02	7.53E+01	9.25E+01	1.01E+02	0.00E+00
1.31E+02	1.45E+02	1.60E+02	1.44E+02	1.28E+02	6.66E+01	1.47E+02	4.54E+01	9.38E+01	1.35E+02
1.11E+02	1.13E+02	1.41E+02	1.47E+02	7.41E+01	1.19E+02	1.11E+02	4.51E+01	4.80E+01	1.27E+02
1.05E+02	9.38E+01	1.32E+02	1.54E+02	3.82E+01	1.58E+02	8.82E+01	7.35E+01	2.85E+01	1.29E+02
1.17E+02	1.48E+02	1.31E+02	7.98E+01	2.03E+02	7.19E+01	1.58E+02	9.34E+01	1.45E+02	9.54E+01
8.64E+01	5.92E+01	1.04E+02	1.47E+02	6.49E+01	1.98E+02	4.83E+01	1.09E+02	5.08E+01	1.17E+02
1.67E+02	1.87E+02	1.93E+02	1.62E+02	1.79E+02	3.05E+01	1.91E+02	8.78E+01	1.43E+02	1.62E+02
1.13E+02	1.13E+02	1.43E+02	1.50E+02	6.90E+01	1.24E+02	1.11E+02	4.96E+01	4.65E+01	1.30E+02
2.19E+02	2.42E+02	2.43E+02	2.04E+02	2.34E+02	6.23E+01	2.47E+02	1.45E+02	2.01E+02	2.10E+02
6.00E+01	8.75E+01	8.34E+01	6.07E+01	1.44E+02	9.79E+01	9.57E+01	4.66E+01	8.29E+01	5.25E+01
1.04E+02	1.21E+02	1.33E+02	1.17E+02	1.24E+02	7.10E+01	1.24E+02	2.08E+01	7.81E+01	1.07E+02
1.66E+02	1.46E+02	1.90E+02	2.21E+02	3.20E+01	2.19E+02	1.36E+02	1.41E+02	9.50E+01	1.94E+02
2.25E+02	2.55E+02	2.39E+02	1.84E+02	2.89E+02	9.84E+01	2.64E+02	1.80E+02	2.39E+02	2.03E+02
8.88E+01	1.20E+02	8.11E+01	2.89E+01	2.18E+02	1.54E+02	1.32E+02	1.32E+02	1.55E+02	5.79E+01

Appendix D – Pearson's Correlation Between Future Costs and (Composite) Features

Table 26 - Pearson's correlation between values of features and total, distance, reshuffle, or penalty costs

Feature name	Total	Distance	Reshuffle	Penalty
√RIH	0.512	0.263	0.562	0.483
RIH	0.510	0.256	0.558	0.488
√MMH	0.509	0.265	0.560	0.476
MMH	0.507	0.258	0.556	0.481
RIH*MMH	0.495	0.238	0.540	0.486
RIH^2	0.494	0.236	0.538	0.487
MMH^2	0.492	0.239	0.537	0.480
RIH*EBLB	0.452	0.236	0.582	0.362
RIH*SOS	0.451	0.235	0.580	0.361
MMH*EBLB	0.449	0.236	0.579	0.356
MMH*SOS	0.447	0.235	0.577	0.355
RIH*ASH	0.439	0.191	0.465	0.462
MMH*ASH	0.437	0.193	0.463	0.458
RIH*OCC	0.423	0.183	0.443	0.450
√LAEBLB	0.414	0.241	0.579	0.273
LAEBLB	0.410	0.231	0.577	0.275
√EBLB	0.408	0.249	0.578	0.251
√SOS	0.407	0.249	0.576	0.249
EBLB	0.404	0.240	0.579	0.252
SOS	0.403	0.239	0.577	0.250
√BS1*LAEBLB	0.348	0.276	0.396	0.216
MMV	0.346	0.219	0.494	0.203
BD	0.338	0.191	0.478	0.224
BS1*EBLB	0.337	0.278	0.381	0.198
SOS*BS1	0.336	0.277	0.380	0.198
US	0.335	0.210	0.475	0.201
RIH*BS1	0.333	0.269	0.304	0.254
EEBLB	0.333	0.177	0.466	0.236
MMH*BS1	0.330	0.269	0.300	0.249
BS1*BD	0.323	0.259	0.366	0.198
BS1*MMV	0.309	0.270	0.337	0.175
BS1*US	0.293	0.261	0.311	0.167
FOC1	0.243	0.196	0.329	0.109
FOC2	0.216	0.178	0.290	0.094
BLD	0.199	0.081	0.235	0.197
SLBL	0.167	0.133	0.207	0.091
TDLB	0.138	0.049	0.081	0.203
SSH	0.116	0.015	0.068	0.231
USP2	0.107	0.004	0.132	0.145
NIS1	0.102	0.125	0.049	0.063

USP1	0.101	0.020	0.024	0.267
MWSP2	0.095	0.003	0.023	0.235
HUSP	0.094	0.004	0.063	0.163
NES	0.092	0.051	0.078	0.099
SHA0	0.092	0.051	0.078	0.099
NIS2	0.087	0.111	0.041	0.050
USP0	0.084	0.002	0.015	0.178
SHA4	0.084	0.035	0.036	0.201
ASH	0.084	0.031	0.041	0.193
USP3	0.083	0.014	0.066	0.126
FIC1	0.080	0.014	0.019	0.178
FIC2	0.079	0.013	0.002	0.167
MWSP1	0.070	0.003	0.022	0.177
NIC1	0.063	0.113	0.009	0.029
SHA1	0.058	0.009	0.041	0.112
NIC2	0.053	0.102	0.016	0.022
SHA2	0.016	0.031	0.005	0.073
SHA3	0.014	0.042	0.032	0.036

Appendix E – Standard Configuration of the ADP Algorithm Used in Testing

- Weights updating algorithm:
    - Recursive Least Squares (see Section 4.4.1)
        - Rho value of 0.1
        - Harmonic step size with a delta of 0.5
- Choice policy: Decreasing epsilon-greedy
    - Starting epsilon: 0.2
    - Multiplication factor per iteration: 0.99
- No pre-training
- Discount factor: 0.99
- Single-stage optimization method: Partial search tree, 1 attempt

Appendix F – Average Contribution to the Value Function Approximation per Feature

Table 27 - average contribution to the VFA per feature from experiment 5.4

Feature	0	1	2	3	4	5	6	7	8	9	10	11	12	avg
EBLB	0.040	0.028	0.031	0.023	0.036	0.037	0.025	0.037	0.026	0.018	0.038	0.024	0.031	0.030
E-EBLB	0.042	0.027	0.034	0.020	0.036	0.034	0.026	0.040	0.025	0.018	0.040	0.015	0.034	0.030
LA-EBLB	0.034	0.021	0.028	0.019	0.033	0.039	0.023	0.037	0.027	0.013	0.038	0.021	0.025	0.028
TDLB	0.049	0.046	0.050	0.064	0.041	0.039	0.057	0.049	0.045	0.052	0.052	0.049	0.055	0.050
BD	0.028	0.026	0.022	0.013	0.043	0.027	0.014	0.029	0.018	0.009	0.031	0.015	0.018	0.023
US	0.041	0.021	0.040	0.020	0.036	0.040	0.030	0.034	0.029	0.018	0.040	0.021	0.034	0.031
BLD	0.040	0.022	0.033	0.036	0.032	0.032	0.034	0.035	0.025	0.033	0.040	0.032	0.029	0.032
ASH	0.058	0.051	0.051	0.068	0.049	0.046	0.070	0.054	0.046	0.049	0.068	0.053	0.055	0.055
SSH	0.045	0.034	0.040	0.055	0.037	0.037	0.051	0.042	0.033	0.037	0.050	0.037	0.038	0.041
NES	-0.003	0.015	0.013	0.004	0.005	0.001	0.001	0.015	0.022	0.013	-0.006	0.016	0.023	0.009
USP0	0.012	0.035	0.016	0.023	0.013	0.014	0.023	0.012	0.028	0.032	0.008	0.034	0.019	0.021
USP1	-0.019	0.017	-0.002	-0.004	-0.002	-0.005	-0.002	-0.003	-0.003	0.006	-0.013	0.014	-0.003	-0.001
USP2	0.007	0.025	0.016	0.023	0.008	0.002	0.023	0.007	0.038	0.030	0.009	0.035	0.015	0.018
USP3	-0.001	0.020	0.001	-0.023	-0.015	-0.015	0.002	-0.005	0.018	0.014	-0.005	0.021	-0.010	0.000
HUSP	0.010	0.030	0.015	0.022	0.012	0.011	0.023	0.010	0.034	0.032	0.007	0.035	0.017	0.020
NIS1	0.051	0.052	0.044	0.068	0.047	0.056	0.061	0.049	0.048	0.058	0.057	0.055	0.053	0.054
NIS2	0.056	0.063	0.044	0.064	0.047	0.056	0.062	0.049	0.071	0.063	0.058	0.066	0.053	0.058
NIC1	0.025	0.014	0.026	0.030	0.033	0.022	0.034	0.020	0.015	0.027	0.022	0.018	0.027	0.024
NIC2	0.023	0.017	0.027	0.031	0.035	0.023	0.032	0.020	0.019	0.030	0.018	0.022	0.027	0.025
FIC1	0.017	0.022	0.016	0.016	0.015	0.029	0.018	0.017	0.020	0.019	0.017	0.015	0.012	0.018
FIC2	0.011	0.023	0.009	0.018	0.011	0.006	0.020	0.021	0.020	0.019	0.011	0.017	0.020	0.016
FOC1	0.044	0.021	0.044	0.039	0.045	0.066	0.033	0.053	0.019	0.025	0.041	0.021	0.048	0.038
FOC2	0.043	0.019	0.040	0.040	0.040	0.055	0.033	0.045	0.019	0.029	0.040	0.024	0.040	0.036
MWSP1	-0.002	-0.001	-0.002	-0.001	-0.003	-0.004	-0.003	-0.005	0.000	0.000	-0.004	0.000	-0.004	-0.002
MWSP2	-0.005	-0.001	-0.002	-0.002	-0.007	-0.005	-0.006	-0.012	0.000	0.000	-0.006	0.001	-0.008	-0.004
MMV	0.028	0.057	0.025	0.028	0.036	0.032	0.025	0.036	0.049	0.081	0.031	0.041	0.034	0.039
MMH	0.046	0.040	0.039	0.050	0.045	0.050	0.046	0.050	0.041	0.051	0.049	0.045	0.050	0.046
RIH	0.044	0.040	0.039	0.050	0.043	0.048	0.042	0.048	0.042	0.051	0.047	0.045	0.049	0.045
EBLB^2	0.020	0.023	0.039	0.016	0.029	0.040	0.013	0.031	0.020	0.012	0.023	0.020	0.018	0.023
EBLB^0.5	0.063	0.052	0.049	0.056	0.052	0.049	0.053	0.051	0.051	0.028	0.057	0.046	0.055	0.051
BD^2	0.016	0.024	0.058	0.020	0.039	0.030	0.008	0.027	0.015	0.006	0.016	0.008	0.008	0.021
BD^0.5	0.041	0.034	0.036	0.034	0.050	0.036	0.035	0.041	0.035	0.036	0.045	0.032	0.038	0.038
NES^2	0.001	0.000	-0.003	-0.004	-0.003	-0.001	-0.002	-0.011	0.011	0.001	0.000	0.003	0.017	0.001
BLD^2	0.038	0.012	0.025	0.015	0.025	0.022	0.020	0.029	0.016	0.018	0.028	0.019	0.021	0.022
HUSP^2	0.001	0.016	0.001	-0.003	-0.003	0.004	0.002	-0.006	0.021	0.016	-0.004	0.017	0.000	0.005
C	0.053	0.055	0.056	0.070	0.061	0.046	0.072	0.053	0.056	0.056	0.059	0.063	0.062	0.059

Appendix G – Overview of the Composition of Each Tested Feature Set

Table 28 - Overview of the composition of each tested feature set

Features	sfs 1	sfs 2	sfs 3	sfs 4	sfs 5	sfs 6	sfs 7	sfs 8	sfs 9	sfs 10	full	red. full	Heur-istic	new 1	new 2	new 3	amount used
ASH	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	2
BD	0	0	0	0	1	1	0	0	0	0	1	1	0	0	0	1	5
BLD	1	0	0	0	1	1	0	0	0	0	1	1	0	0	1	1	7
EBLB	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	3
EEBLB	1	0	0	0	1	1	0	0	0	0	1	1	0	0	1	1	7
FIC1	1	1	0	0	1	1	1	0	1	0	1	1	0	1	0	0	9
FIC2	1	0	0	0	1	1	0	1	1	0	1	1	0	1	0	0	8
FOC1	1	1	0	0	1	1	1	1	1	0	1	1	0	1	0	0	10
FOC2	1	1	1	1	1	1	1	1	1	1	1	1	0	1	0	0	13
HUSP	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	2
LAEBLB	0	0	0	0	0	1	0	0	0	0	1	1	0	0	0	0	3
BS1*LAEBLB	0	1	0	1	0	0	1	1	0	1	0	0	0	0	0	0	5
BS1*MMV	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
BS1*US	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	2
MMH*BS1	0	1	1	1	0	0	0	1	1	1	0	0	0	0	0	0	6
SOS*BS1	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	1
MMH	1	0	0	0	1	1	0	0	0	0	1	1	1	0	1	1	8
MMV	1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	1	6
MWSP1	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	3
MWSP2	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	2
NES	1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	5
NIC1	0	0	0	0	0	1	0	0	0	0	1	1	0	1	0	1	5
NIC2	0	1	1	1	0	0	1	1	1	0	1	1	0	1	0	0	9
NIS1	1	0	0	0	1	1	0	0	0	0	1	1	0	1	0	1	7
NIS2	1	0	0	0	1	1	0	0	0	0	1	1	0	1	0	1	7
RIH	1	0	0	0	1	1	0	0	0	0	1	1	1	0	1	1	8
√BD	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	3
√EBLB	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1	1	3
√MMH	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
√RIH	0	1	1	1	0	0	1	1	1	1	0	0	1	0	0	0	8
√SOS	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
SHA4	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	2
SLBL	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
BD^2	1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	5
BLD^2	1	0	0	0	1	1	0	0	0	0	1	1	0	0	0	0	5
$EBLB^2$	1	0	0	0	1	1	0	0	0	0	1	1	0	0	1	1	7
$HUSP^2$	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
MMH^2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1
NES^2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
RIH^2	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	1

SSH	1	0	0	0	1	1	0	0	0	0	1	1	0	0	1	0	6
TDLB	0	1	1	1	0	0	1	1	1	1	1	1	0	0	1	1	11
US	0	0	0	0	0	1	0	0	0	0	1	1	0	0	1	1	5
USP0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	2
USP1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
USP2	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1
USP3	1	0	0	0	1	1	0	1	1	0	1	1	0	0	0	0	7
Set Size	**14**	**7**	**4**	**5**	**14**	**16**	**6**	**8**	**8**	**4**	**23**	**17**	**4**	**10**	**6**	**14**	