Erasmus University Rotterdam
Erasmus School of Economics
Konecranes
Master Thesis Econometrics and Management Science

Automating the parameter tuning of a stowage planner

Name student: D. Stapel
Student ID number: 538778

Supervisor: P.C. Bouman
Second assessor: R.M. Badenbroek
Company supervisor: C. Boer
Company supervisor: E. van Wingerden

Date final version: 4 December 2023

Abstract

Terminal Operating Systems are complex systems that contain multiple algorithms with parameters that can be tuned to improve the performance of a container terminal. This parameter tuning is a time-consuming process that is often done by hand. In this thesis we investigate the possibility to automate the tuning process for the stowage planner of a Terminal Operating System. This automation would reduce the labor needed for parameter tuning and it can also improve the performance of the container terminal. We develop our Iterative Parameter Tuning Method and apply it on historic data from a container terminal that is currently live. The Iterative Parameter Tuning Method iteratively estimates the effects of parameters using emulations and a linear regression on the results of those emulations and applies our Parameter Suggestion Model until we get convergence. For our Parameter Suggestion Model, we investigate four different heuristics that model the stowage planner. Our Iterative Parameter Tuning Method reaches convergence within seven iterations and this tuning advice reaches a higher Quay Crane productivity than the benchmark we compare our advice to. Often, the Iterative Parameter Tuning Method already reaches convergence within three iterations. The results show that automation of the tuning process is a possibility.

Contents

1 Introduction

Operating a container terminal is a complex process. A container terminal often has thousands of containers laying in its yard and there are often multiple vessels that need to load and discharge containers. All these containers need to be assigned to locations on the vessel and in the container yard. In most container terminals there is a Terminal Operating System (TOS) that controls all the processes and the people in the container terminal. All those processes depend on each other and a small delay at one process often also impacts the other processes. As a result, small inefficiencies in one part of the container terminal can result in delays and extra costs over the entire container terminal. Improving the TOS can have a huge effect on the performance of a container terminal (C. Boer & Saanen, 2012).

A TOS contains multiple algorithms that it uses to make decisions in the container terminal. Those algorithms have multiple parameters that can be tuned to improve the performance of a container terminal. The goal of this thesis is to investigate the possibility to automate the optimization of the container terminal performance by tuning the parameters of one of the algorithms that the TOS uses. We focus on the stowage planner algorithm that computes the stowage plan for a vessel that assigns containers to specific locations on the vessel.

Currently, functional experts at Konecranes or employees from the container terminal improve the performance of the TOS by tuning the parameters of the algorithms in the TOS by hand. They make use of Key Performance Indicators, computed based on emulations using CONTROLS, the emulation tool from Konecranes. Emulation is a simulation of a virtual container terminal that integrates the real TOS. The most important Key Performance Indicator that they use to determine the performance of the TOS is the Quay Crane (QC) productivity (Jonker et al., 2021).

The tuning process by hand is time consuming, as the functional experts need to configure different settings, start new emulations and interpret the Key Performance Indicators. To be able to interpret all the results from the emulations correctly, the functional experts also need to have enough experience with the parameters and the algorithm. Otherwise, their tuning efforts might not even actually improve the performance of the container terminal. Automating the tuning process would decrease the amount of labor needed to tune the parameters from the TOS. Next to that, parameter tuning would also become available for people with less experience in parameter tuning as the automated parameter tuning would take most of the decisions.

We develop our Iterative Parameter Tuning Method to tune the parameters from the stowage planner. This Iterative Parameter Tuning Method first estimates the effects from the stowage planner parameters on the QC productivity using a linear regression on the results from emulations with stowage plans generated by the stowage planner for different sets of parameter values. Then the Iterative Parameter Tuning Method iteratively suggests a parameter suggestion using our Parameter Suggestion Model and updates the linear regression until we get convergence of the parameter suggestions. In the Parameter Suggestion Model, we can use one of four heuristics that we propose to model the stowage planner. Those heuristics are the Simple Local Search, the Greedy Randomized Adaptive Search Procedure, the Simulated Annealing, and the Large Neighborhood Search heuristic. To evaluate our approach, we use the historic data from a terminal that is currently live.

We find that the Iterative Parameter Tuning Method for each heuristic reaches convergence within seven iterations. For most of the heuristics the Iterative Parameter Tuning Method even reaches convergence within three iterations. The estimated QC productivity of the stowage plan that the stowage planner generates for the tuned parameters is higher than the benchmark we compare our results to. This indicates that the Iterative Parameter Tuning Method is able to tune the parameters successfully. The Iterative Parameter Tuning Method gets the best results if it uses the GRASP heuristic in the Parameter Suggestion Model.

There is one problem with the Iterative Parameter Tuning Method and that is the long computation time of the method. This can make the application of the method on a real-time problem difficult. However, the Iterative Parameter Tuning Method can still be applied in more long term parameter tuning projects that aim to find the best parameter values of the stowage planner on the long term.

The Iterative Parameter Tuning Method that we develop in this thesis is able to tune the parameters of the stowage planner successfully and all the steps of the Iterative Parameter Tuning Method can be fully automated. This shows that it is possible to automate the parameter tuning of the stowage planner.

In Section 2 we give an overview of the relevant literature, while we describe the problem in Section 3. We describe our methods in Section 4 and our computational experiments in Section 5. Finally, we give our conclusion in Section 6.

2 Literature review

In this section we go over the literature that concerns the problem discussed in this thesis. First, we give a small overview about emulation and how we can use it to optimize the performance of a Terminal Operating System (TOS). Then we give an overview on the literature in black-box optimization and lastly, we go over literature on making vessel stowage plans.

2.1 Emulation

Emulation is a way to evaluate the performance of control systems. Compared to a real simulation, a part of the simulation is replaced by the real control system (McGregor, 2002). In this way emulations are closer to real live, which gives a better representation of how the control system would work in real live.

Emulation is not the only option to evaluate the performance of a control system. Other options are full simulation, real-time control and prototyping (C. A. Boer & Saanen, 2008). In Table 1 we give an overview of how we can apply those four options to the testing and tuning of the TOS from a container terminal. Testing the container terminal using prototyping would give the most accurate results. However, as prototyping uses the real TOS and the real terminal, there could be real consequences if something goes wrong while testing.

Table 1: Overview of the different possibilities to test the performance of a TOS

Method	TOS	Terminal
Full simulation	Simulated	Simulated
Real-time control	Simulated	Real
Emulation	Real	Simulated
Prototyping	Real	Real

C. A. Boer & Saanen (2008) developed an emulation tool for container terminals. They call their emulation tool CONTROLS. Other emulation software for container terminals is CHESSCON developed by AKQUINET group (2023). We make use of CONTROLS to tune the performance of a stowage planner from a TOS. C. Boer & Saanen (2012) already apply this successfully. For example, they use CONTROLS to test whether changing the dispatching algorithm and the corresponding parameters would result in improved truck turn-around times. This results in improvements of approximately 30% of those truck turn-around times.

One restriction of emulation is that we cannot speed the emulations up too much compared to real time, as it uses the real TOS. If we speed up the emulation too much, it affects the decision making of the TOS. Because of that, one emulation can take several hours. Comparing different parameter settings thus takes a long time.

2.2 Black-box optimization

Black-box optimization is the optimization of a problem of which it is unknown how the objective function and the corresponding constraints are defined according to Alarie et al. (2021). They mention that optimization problems that require a simulation to be able to evaluate the objective are an example of a black-box optimization problem that occurs often. This corresponds to the

problem we consider in this thesis, where we tune parameters using emulation runs. Black-box optimization is also called derivative-free optimization (Rios & Sahinidis, 2013).

There are numerous ways to solve a black-box optimization problem. One of the classifiers Rios & Sahinidis (2013) uses to distinguish between the different options, is whether the solution approach is direct or model-based. According to Audet (2014), direct search methods are methods that only use the function evaluations of the function they are trying to optimize and not their derivatives.

Generalized pattern search (GPS), proposed by Torczon (1997), is an example of a direct solution approach. In a GPS algorithm, the algorithm improves an initial solution in multiple iterations. Each iteration they define a step size and based on this step size they find a new solution. If this results in an improvement of the objective function, this newfound solution becomes the current best solution. Then they update the step size parameter for the next iteration and repeat the process.

GPS has later been adapted by Audet & Dennis Jr (2006) to be able to handle general constraints and to be able to search in all directions. They call this adapted algorithm Mesh Adaptive Direct-Search (MADS). One advantage of using a direct search algorithm is that this method can be parallelized, which would increase the speed of the algorithm (Conn & Le Digabel, 2013).

Another possibility to solve a black-box optimization problem is using a model-based approach. A model-based approach uses a surrogate model to approximate the real function (Rios & Sahinidis, 2013). New parameter estimates can then be based upon that model. Vu et al. (2017) mentions polynomials, radial basis functions, kriging, support vector machines and mixed surrogate models as possible surrogate models.

Papalexopoulos et al. (2022) and Zhu & Bemporad (2023) explain that it is possible to rewrite some of those surrogate models in Mixed Integer Programming (MIP) formulations. In this way it is also possible to add restrictions on the parameter that you are optimizing. When you apply this methodology, the model becomes specific to a certain problem and as a result it is difficult to apply the same model to multiple different problems.

In our thesis we treat the algorithm of the stowage planner as a black-box. We give the stowage planner the parameters as input and this then generates a stowage planning. We use the surrogate model approach, as we model the stowage planner using a MIP formulation. We also treat the relationship between the stowage planner parameters and their QC productivity as a black-box, because this relationship depends on an emulation and the stowage planner. Again, we use the surrogate model approach, however this time we use a regression to model the relationship.

2.3 Vessel stowage planning problem

Now we examine the literature on how to model the vessel stowage planning problem. There are two main approaches to solve the stowage planning problem. Some papers try to solve the entire stowage planning problem at once, while there are other papers that use decompositions to make solving the problem easier.

Larsen & Pacino (2021) solves the stowage planning problem without decompositions. They give a mathematical formulation, which they solve using an Adaptive Large Neighborhood Search (ALNS) framework. In total they make use of nine different Key Performance Indicators, which they insert in a weighted sum, to assess the quality of their solution.

A decomposition of the stowage planning problem that is often used is the 2-phase hierarchical decomposition used by Pacino (2012). They first distribute groups of containers to sections on the vessel to make a planning that they call the master planning. Then they allocate the individual containers in those groups to specific locations in the sections. This second stage problem is also called the container slot planning problem.

Pacino (2012) solves the master planning in their decomposition approach using a MIP approach and they solve the container slot planning problem for each of those sections using constraint programming. They use multiple objectives in the container slot planning problem. They minimize the number of shifts, the number of normal containers in the reefer spots, the number of stacks used and the number of different port of destinations in a stack.

Parreño et al. (2016) focuses only on the container stowage slot planning problem. They propose an integer programming formulation and find a solution to this formulation using a Greedy Randomized Adaptive Search Procedure (GRASP). They use the same objective as Pacino (2012). Another paper that only focuses on the container stowage slot planning problem is Korach et al. (2020). They also propose an integer programming formulation, however they use a ruin-and-recreate matheuristic to solve their problem.

In this thesis we only focus on the container stowage slot planning problem part of the stowage planning problem. Compared to all the papers described above, we have a completely different objective. The main goal in our thesis is to optimize the Quay Crane (QC) productivity, while the other papers focus more on overstowage/shifts and other performance indicator on the ship itself. Their goal is more to prevent extra inefficiencies in the stowage plannings on the next destinations of the vessels, while we focus on the current container terminal.

There are also papers that do focus on the terminal side of things. One example of such a paper is Monaco et al. (2014), who use the travel time between the yard and the vessel and yard shuffles in their objective function from their Binary Integer Program. To solve their formulation, they make use of a two-step heuristic, where they first construct a feasible solution and then use Tabu Search to improve that solution in a second stage.

3 Problem description

The main goal of this thesis is to investigate whether it is possible to automate the parameter tuning process of the stowage planner algorithm from the Terminal Operating System (TOS) of a container terminal. To give a better understanding of the problem we first give a description of how a container terminal works. Then we explain what the difficulties in stowage planning are and thereafter we give a description of how the stowage planner algorithm works.

3.1 Container terminal explanation and definitions

There are two main operations on container terminals. One of those is the waterside operation that focuses on moving containers from the yard to the vessel and the other way around. The other is the landside operation that focuses on moving containers from trucks and/or trains to the yard and the other way around. As there is time between the arrival of containers in the container terminal and the departure from the container terminal, containers must be stored in a container yard for this time period.

In this thesis we focus on the stowage process where containers are loaded to a vessel. This process makes use of multiple different machines. Rail Mounted Gantry Cranes (RMGs) or Rubber Tired Gantry Cranes transfer containers from the container yard to the transfer bay near the container yard. From this transfer bay Automated Guided Vehicles (AGVs) or Straddle Carries (SCs) pick up those containers. The AGVs or SCs then bring the containers to the transfer zone under the Quay Cranes (QCs). From there the QCs place the containers on the vessel.

We use a few descriptions of positions in the container terminal in our thesis. The yard is divided in multiple distinct parts. We call each of those parts a yard block. We call a stack of containers in a yard block a yard stack. On a vessel we define a vessel location as a slot on the vessel where a container can be stored.

To determine the performance of the stowage plan we use the QC productivity as the Key Performance Indicator. QC productivity denotes the number of containers that one QC handles within an hour. This includes the containers that the QC handles to load a container to the vessel and to discharge a container from the vessel. Jonker et al. (2021) explains that QC productivity is the most important Key Performance Indicator to evaluate the performance of a container terminal. C. Boer & Saanen (2012) also uses the QC productivity in their evaluations of the container terminal performance.

3.2 Vessel stowage plan

We focus on making the vessel stowage plan from the container terminals perspective. Before a container terminal can make their vessel stowage plan, they receive a first stowage plan, called the projections, from the vessel operator. To make the projections vessel operators put all the containers, which need to be loaded onto the vessel, in different stowage groups. The vessel operator then assigns those stowage groups to certain parts of the vessel. Those stowage groups depend on the port of destination from the container, the type of the container and the weight of the container. The container terminal operator then uses those projections to make the second stage of the stowage plan, where it connects specific containers to specific vessel locations.

When making a vessel stowage plan there are a few situations that must be avoided as much as possible. Often, not all containers, which are loaded onto a vessel, have the same port of destination. At the next port, it should be possible to pick up all the containers for that port without too much unnecessary moves. Thus, a proper stowage planning should not include situations where you place a container on another container that has a port of destination that the vessel arrives at earlier. Otherwise, you would have to move containers to be able to reach the containers that are destined for the current port. Normally, container terminal operators do not take this into account, as vessel operators handle those situations in the projections.

Furthermore, the containers need to be distributed in such a way that the weight distribution of the containers is safe on the sea. If all the heavy containers would be allocated to one side of the vessel, then the vessel would lean over towards that side. On the sea this would not be stable and thus would result in dangerous conditions. This optimal distribution of the weights of the containers differs per vessel. Partially, the projections already take this into account. The vessel operator uses the weight class corresponding to the stowage groups as one of the inputs for their projections. However, as not all containers in a stowage group have the same weight, those weights also must be divided across each planned group. One of the situations the container terminal operator tries to avoid in their stowage plan is placing heavier containers above lighter containers. This is called weight inversion as the weights of the containers are inverted compared to the more optimal situation.

Other inefficiencies that the container terminal operator wants to avoid are concerned with picking containers up from the yard if they make a vessel stowage plan. An example of this is when there are two containers that are stacked on top of each other in the yard that both need to be loaded to the same vessel. This causes an unnecessary move, if they need to move the container below before the upper container. Thus, they want to avoid that as much as possible.

There could also arise delays if multiple containers come from the same yard block in close succession. If this occurs the RMGs cannot keep up with the demand for the containers. At a certain point, those delays transfer to other equipment in the container terminal, causing even more delays.

3.3 The stowage planner

In this section we introduce the stowage planner that we tune the parameter from. The stowage planner makes the second stage of the stowage planning, also called the container slot planning, using the projections as an input. The algorithm is build-in in the TOS. Before the container terminal commences stowing an arriving vessel, the container terminal runs the algorithm. This generates a stowage plan that the stowage planner then sets as the stowage plan in the TOS.

To determine the stowage plan the stowage planner makes use of a recursive-based iterative algorithm. We give a pseudocode of this algorithm in Algorithm 1. The first step in this algorithm is to create an initial solution, which is only based on the weight of the containers and the stowage groups. To accomplish this, the stowage planner sorts all the vessel locations for each stowage group based on the optimal weight distribution. For each stowage group, the algorithm then assigns the container with the most weight each time to the first vessel location in the sorted list of vessel locations. Algorithm 2 gives the pseudocode for the initial solution creation.

Algorithm 1: Stowage planner pseudocode

Input : S: Set of all stowage groups
L: Set of all vessel locations
C: Set of all containers
SearchDepth : Stowage planner setting
SearchWidth : Stowage planner setting

Output: PL: Stowage planning

PL ← `CreateInitialSolution(`*S, L, C*`);` `// See Algorithm 2 for this function`
$S^s \leftarrow \emptyset$; `// Initialize the set of stowage groups already selected`
$S^n \leftarrow S$; `// Initialize the set of all stowage groups not yet selected`
TotPEN ← Total penalty of the current planning PL;
while $S^n \neq \emptyset$ **do**
  $s \leftarrow$ Stowage group out of S^n with the lowest penalty;
  TempPL ← `ImproveSolution(`*s,*L_s*,*C_s*,SearchDepth,SearchWidth*`);` `// See Algorithm 3 for this function`
  TempPEN ← Total penalty of the planning TempPL;
  if *TempPEN* $\leq$ *TotPen* **then**
    PL ← TempPL;
    TotPEN ← TempPEN;
  end
  $S^s \leftarrow S^s \cup \{s\}$;
  $S^n \leftarrow S^n \setminus \{s\}$;
end

Thereafter, the algorithm aims to improve this initial solution. It computes the total sum of all the penalties for each stowage group. Then it tries to improve the solution of each stowage group once in an order based on the total sum of penalties corresponding to the stowage group.

When the algorithm tries to improve the solution for one of the stowage groups, it starts by completely removing the solution for that stowage group. Then, the algorithm tries to find the optimal container for all the vessel locations without an assigned container. The order in which it considers the vessel locations depends on their expected move time. This expected move time is the planned time that a container is stowed to that vessel location. The algorithm selects the vessel location with the earliest move time that has no assigned container first.

There are two settings that influence how the stowage planner finds the containers for a

Algorithm 2: Function that describes initial solution creation for the stowage planner

Input : S: Set of all stowage groups
L: Set of all vessel locations (split in subsets per stowage group: L_s)
C: Set of all containers (split in subsets per stowage group: C_s)

Output: PL: Intial stowage planning

Function `CreateInitialSolution(`*S, L, C*`):`
  for $s \in S$ **do**
    Sort C_s based on container weight;
    Sort L_s based on their tier and stack;
    Assign each container in C_s to the vessel location in L_s with the same index;
  end
end

Algorithm 3: Recursive algorithm that improves the solution for a stowage group

```
Input  : s: Stowage group
         C_s: Set of container in stowage group s
         L_s: Set of vessel locations in stowage group s
         SearchDepth : Stowage planner setting
         SearchWidth : Stowage planner setting
Output: PL : Updated stowage planning
Function ImproveSolution(s, C_s, L_s, SearchDepth, SearchWidth):
    Sort L_s based on the planned move time;
    foreach l ∈ L_s do
        /* Define O_i as the set of all the different chains of containers
           considered for the first i vessel locations starting from l    */
        O_1 ← Set of the SearchWidth containers with the lowest penalty for location l;
        for i = 2 to SearchDepth do
            foreach o ∈ O_(i−1) do
                O'_i ← Set of the SearchWidth containers with the lowest penalty for
                 location l of the remaining containers;
                foreach o' ∈ O'_i do
                    Add the new chain (add o' after o) to O_i;
                end
            end
        end
        Assign the first container of the lowest total cost chain in O_SearchDepth to vessel
         location l;
    end
end
```

vessel location. This are the settings SearchDepth and SearchWidth. The setting SearchDepth sets the number of different containers that it considers for each vessel location, while the setting SearchWidth defines the number of steps the stowage planner looks forward in deciding the best container for a vessel location. In this thesis we keep those two stowage planner settings constant on a value of five. Increasing both these settings often results in better solutions, however it also increases the computation time of the algorithm.

The stowage planner selects for each vessel location a configurable number of options and then looks forward a configurable amount of steps based on the stowage planner settings SearchDepth and SearchWidth. This creates chains corresponding to different orders of assigning containers to the vessel locations. At the end, the stowage planner selects the container which is the first container in the chain with the lowest total cost. Then it moves on to the next vessel location and it repeats the entire process. We give the pseudocode for this container selection in Algorithm 3.

We give an example of how the stowage planner selects a container for a vessel location in Figure 1. For this example, we assume that the stowage planner setting SearchWidth is set to two and the stowage planner setting SearchDepth is set to three. We are searching for the best container to load to vessel location 1. As stowage planner setting SearchWidth equals two, we select for each vessel location that we consider the two best containers. Figure 1 illustrates two possible containers for the first vessel location: container 1 and container 2. Furthermore, the figure also illustrates the next two vessel locations that the algorithm considers, because stowage

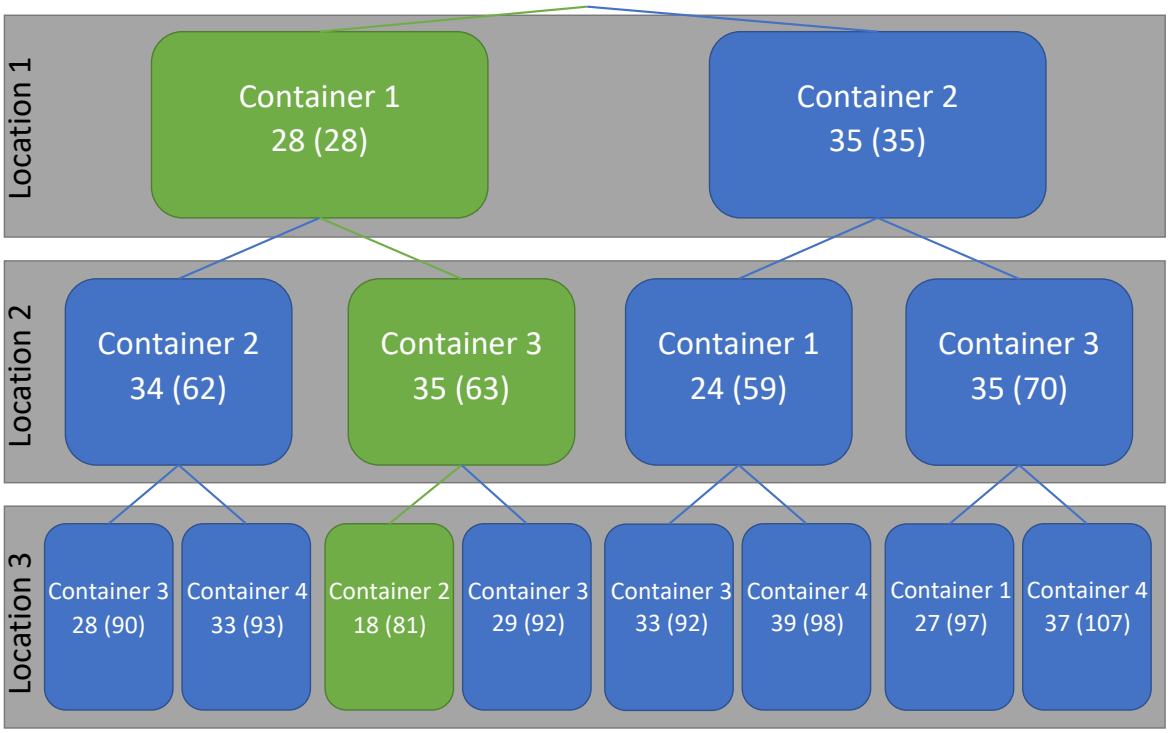

Figure 1: Example of how the stowage planner selects a container for vessel location 1 as defined in Algorithm 3. Each container has a value that denotes the penalty if the container is allocated to the location of interest given the chain above it. Between parentheses it gives the penalty of the entire chain until the container of interest.

planner setting SearchDepth is three. For each vessel location it illustrates the two container options with the lowest penalty given the containers selected in the earlier vessel locations.

As we demonstrate in the example, the stowage planner settings for SearchWidth and SearchDepth result in eight different chains of possible solutions (2^3). The algorithm selects the chain with the lowest total penalty. In this example that is the chain with the green color that has a total penalty of 81 compared to the other chains that all have a total penalty higher than 90. Given these total penalty values, the stowage planner selects container 1 as the container that is allocated to vessel location 1. The stowage planner then repeats this process until it has allocated all containers to a vessel location.

3.4 Stowage planner parameters and penalties

Now we know how the stowage planner works, we give an overview of the parameters from the stowage planner that we consider in this thesis in Table 3. These are the parameters that we tune and some parameters that have the aim to limit weight distributions problems. The decision on which parameters to tune, has been made with the help of a functional expert from Konecranes. We have chosen those parameters, because the functional expert expects that they have the biggest impact on the QC productivity out of all the available parameters.

The first two parameters that we tune are the CCSPA and CLDPA parameters. The stowage

Table 2: An overview of the parameters and penalties that we use from the stowage planner

	Penalty	Parameter	Parameter explanation
Non-weight related penalties	CCSPE	CCSPA	Penalty value for close containers from same block
		CLDPA	Value that sets when two moves are close
	QCSPE	QCSPA	Penalty value for each extra QC containers go to per stack
	YASPE	YASPA	Penalty value for each yard shift
Weight related penalties	WEIPE	WEIPA	Penalty value for each case of weight inversion
	OWDPE	OWDPA	Penalty value for each difference with the optimal weight

planner uses both these parameters to compute the CCSPE penalty. The goal of the CCSPE penalty is to limit the number of times a certain QC handles multiple containers that come from the same yard block in close succession of each other. This is important because it could cause delays if the ASC (Automated Stacking Crane) in that yard block cannot keep up with the demand for containers. Spreading the containers out over time avoids those delays and thus most likely improves the performance of the QCs.

The CLDPA parameter defines the number of moves that the stowage planner considers as close. Thus, if the CLDPA parameter is three, then the stowage planner applies a penalty if a container is moved from the same yard block in the two moves before the current move. We can set the size of the penalty with the CCSPA parameter. The closer two moves from the same yard block are, the more the stowage planner applies the penalty.

Another parameter that we tune is the QCSPA parameter. This parameter sets the size of the penalty for the QCSPE penalty. The goal of the QCSPE penalty is to minimize the number of times that containers in one yard stack are send to multiple different QCs as this increases the chance on yard shifts. It is possible that one of the QCs is behind on schedule and thus containers, scheduled to that QC, are removed from the yard later than planned. If containers in one yard stack go to different QCs, there can be containers below those containers that are stowed earlier then the delayed containers. This causes extra yard shifts. If all the containers in one yard stack go to the same QC this would not be a problem as all the containers in the yard stack will be delayed. If the stowage planner plans containers in one yard stack to more than one QC then it adds a penalty of the size QCSPA to the QCSPE penalty for each extra QC.

The last parameter that we tune is the YASPA parameter belonging to the YASPE penalty. Each time a yard shift has to be applied with at least two containers that have to be stowed to the vessel, the stowage planner adds YASPA to the YASPE penalty. The goal of this penalty is to limit yard shifts and thus delays in the yard which in turn result in a lower QC productivity.

We also explain two penalties and their parameters that we do not tune. These are penalties that limit weight distribution problems. The WEIPE penalty is a penalty for weight inversions. Each time the stowage planner stacks a heavier container above a lighter container from the same stowage group on the vessel, we add the value of the WEIPA parameter to the WEIPE penalty. There is also a penalty for each container that the stowage planner assigns to a vessel location if the weight of the container is heavier than the optimal weight of that vessel location. This is the OWDPE penalty and the OWDPA parameter sets the size of this penalty. Each time the stowage planners assigns a container it also recomputes the optimal weight for the remaining empty vessel locations.

4 Methodology

4.1 General outline of the Iterative Parameter Tuning Method

In this section we describe the steps of our Iterative Parameter Tuning Method that we use to tune the stowage planner parameters. Our Iterative Parameter Tuning Method comes down to the following steps:

- Estimate the effects of the stowage planner parameters on the QC productivity.
- Suggest stowage planner parameters values using the estimated effects.
- Repeat until we get the same parameter suggestion three iteration in a row.

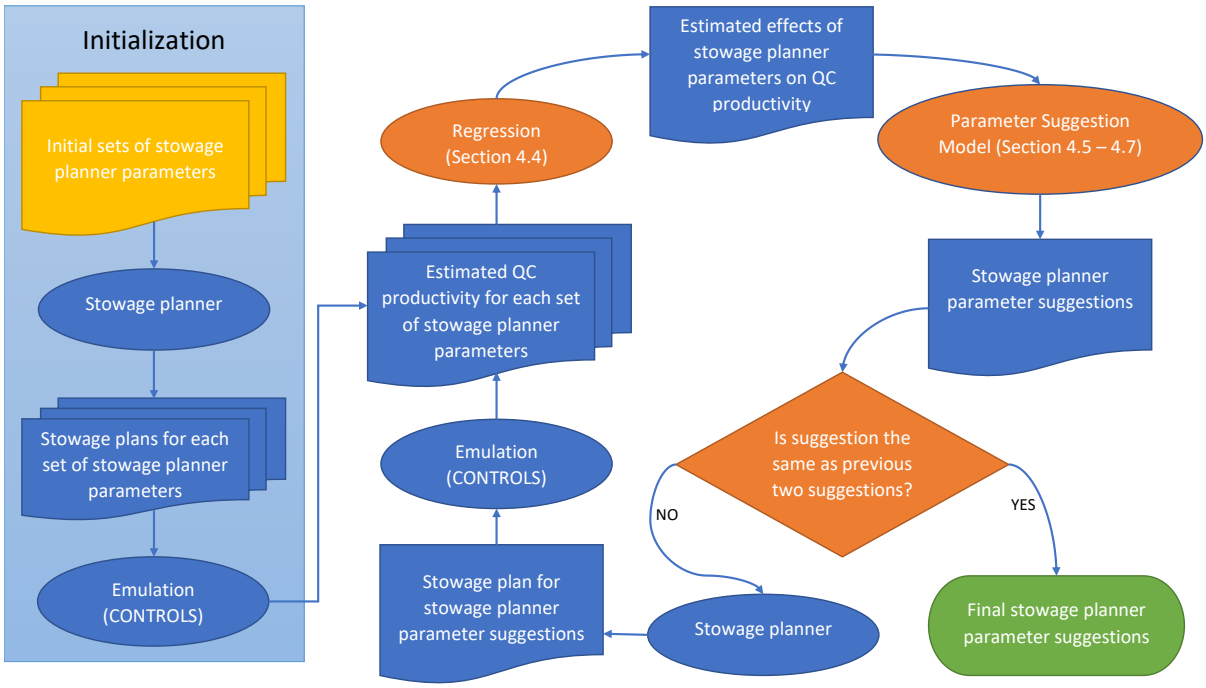

Figure 2: Overview of the Iterative Parameter Tuning Method

Figure 2 gives a more precise overview of all the steps that we take in our Iterative Parameter Tuning Method. Some shapes in this Figure have a distinct color. The yellow color indicates the input of the methodology. The orange color indicates a method or decision that we use and green indicates the output of our methodology. All methods that we use have an oval shape and all decisions a diamond shape. The other shapes indicate inputs and outputs of the methods. If these shapes consist of multiple shapes behind each other, then there are multiple sets of inputs or outputs that follow the same path.

We commence the Iterative Parameter Tuning Method in the initialization phase. The goal of this initialization phase is to generate enough information to be able to do a representative regression from the stowage planner parameters on the QC productivity. We first select multiple sets of stowage planner parameters. For each set of stowage planner parameters, we create a stowage plan using the stowage planner. For each stowage plan we use CONTROLS to get an estimate of the QC productivity for that stowage plan. Emulations do not use a real container

terminal, but a simulated one (C. A. Boer & Saanen, 2008). Therefore, there is randomness in the TOS that snowballs throughout the entire emulation. If the TOS takes a different decision at the start of an emulation, this automatically has an effect on all the next decisions. This means that we can only get an estimated QC productivity for a stowage plan.

After collecting all the estimated QC productivities for the different stowage plans, we apply a linear regression on those results. This regression yields the estimated effects of the stowage planner parameters on the QC productivity, which we use in the next step. We use this regression as a surrogate model for the real relationship between the stowage planner parameters and the QC productivity. We present a more extensive explanation of our regression in Section 4.4.

Using the estimated effects of the stowage planner parameters on the QC productivity, we generate a stowage planner parameter suggestion. For this suggestion we introduce our Parameter Suggestion Model that we explain in more detail in Section 4.7. The goal of this model is to suggest parameters that result in an as high as possible QC productivity. There is one restriction on the parameter suggestion. We want to keep the total weight related penalty from the stowage plan that corresponds to the stowage planner parameter suggestion under a set value. In this way we limit the number of weight related inefficiencies that occur in the parameter suggestion from the Parameter Suggestion Model.

To be able to compute the total weight related penalty we have to compute solutions to the stowage problem given a stowage planner parameter set. We have decided not to use the stowage planner itself as one run of the algorithm can take up to three minutes. In the worst case, our Parameter Suggestion Model needs to compute the total weight related penalty hundreds of times, which would mean that the Parameter Suggestion Model would take hours to get a parameter suggestion. Compared to the long emulation times this is still not long, but it is inconvenient to add this extra computation time.

We treat the algorithm of the stowage planner as a black box and use a Mixed Integer Programming formulation as a surrogate model. We present our MIP formulation of the stowage planner in Section 4.5. As different options to solve the surrogate model, we introduce four different heuristics in Section 4.6. Those four heuristics are a Simple Local Search, a Greedy Randomized Adaptive Search Procedure (GRASP), a Simulated Annealing, and a Large Neighborhood Search (LNS) heuristic.

When we have a stowage planner parameter suggestion, we create a stowage plan using the stowage planner for this parameter suggestion. We evaluate the performance of this stowage plan using CONTROLS and add the resulting estimated QC productivity to estimated QC productivities from the initialization phase and previous iterations. We continue by performing a new regression with the extra added estimated QC productivity and generate a new parameter suggestion with the Parameter Suggestion Model. We continue this iterative process of the Iterative Parameter Tuning Method until the Parameter Suggestion Model gives the same parameter suggestion for three iterations in a row, which we call convergence of the parameter suggestions. We assume that there would be no more further changes if we would continue more iterations when the parameter suggestion already stays the same for three iterations.

4.2 Parameters and sets

We continue by giving an overview of all the parameters, sets and decision variables that we use in our methodology in Table 3.

Table 3: Parameters, sets and decision variables that we use in our methodology

	Sets
P	Stowage planner parameters - {CCSPA, CLDPA, OWDPA, QCSPA, WEIPA, YASPA}
P^t	Stowage planner parameters that we tune - {CCSPA. CLDPA, QCSPA, YASPA}
F	Stowage planner penalties - {CCSPE, OWDPE, QCSPE, WEIPE, YASPE}
F^t	Stowage planner non weight related penalties - {CCSPE, QCSPE, YASPE}
F^c	Stowage planner penalties that we compute per container - {CCSPE, OWDPE, WEIPE, YASPE}
S	Stowage groups corresponding to the vessel
T	Yard blocks where containers are placed before they move to the vessel
Q	QCs that are used to load the vessel
D	Yard stacks where containers are placed before they move to the vessel
L	Locations on the vessel where containers can be stowed to
L_s	Locations on the vessel where containers can be stowed to for stowage group $s \in S$
L_q	Locations on the vessel where containers can be stowed that QC $q \in Q$ handles
C	Containers that need to be loaded onto the vessel
C_s	Containers that need to be loaded onto the vessel for stowage group $s \in S$
C_t	Containers that need to be loaded onto the vessel that come from yard block $t \in T$
C_d	Containers that need to be loaded onto the vessel that come from yard stack $d \in D$
A	Emulations used in the regression
	Parameters
π^p	Constant parameter value assigned for parameter $p \in P$ used to solve the stowage problem
e_{ci}	1 if container $c \in C$ is below container $i \in C$ and $\gamma_{ci} = 1$, 0 otherwise
g_l	The number of locations that we stow before location $l \in L$
k	The maximum allowed total sum of penalties over weight penalties $f \in F \setminus F^t$
l^p	The lower bound on a parameter $p \in P^t$
o_l	Optimal weight of the container placed at location $l \in L$ on the vessel
u^p	The upper bound on a parameter $p \in P^t$
w_c	Weight of container $c \in C$
λ_h	CCSPE multiplier if two containers from the same yard block are stowed h moves after each other
γ_{ci}	1 if containers $c, i \in C$ are stacked in the same yard block, 0 otherwise
ν_{lj}	1 if location $j \in L$ is above location $l \in L$, 0 otherwise
ρ_{lj}	1 if containers get stowed to location $l \in L$ before location $j \in L$, 0 otherwise
r^p	Estimated effect of a parameter $p \in P^t$ on the QC productivity following from a linear regression
r^0	Constant effect on the QC productivity following from a linear regression
QC_a^{prod}	Estimated QC productivity in emulation $a \in A$
b_a^p	Parameter value for parameter $p \in P^t$ in emulation $a \in A$
ss^p	Step size that we use in the Parameter Suggestion Optimization heuristic for parameter $p \in P^t$
	Decision variables
y_p	Parameter value assigned to parameter $p \in P^t$ used in the Parameter Suggestion Model
x_{cl}	1 if container $c \in C$ is allocated to location $l \in L$, 0 otherwise
b_{ch}	1 if in the $h, 1 \leq h \leq \pi^{CLDPA}$ moves to a QC before container $c \in C$ there is a container $i \in C$, such that $a_c = a_i$, 0 otherwise
m_c^f	Penalty value from penalty $f \in F^c$ for container $c \in C$
m_d^{QCSPE}	Penalty value from the QCSPE penalty for yard stack $d \in D$
μ_{dq}	1 if QC $q \in Q$ is used for containers in yard stack $d \in D$
α_{ci}	1 if container $c \in C$ is stowed earlier then container $i \in C$

4.3 Determining optimal weight of the vessel locations

To compute the total weight related penalty, we must know the optimal weight of all the vessel locations. The assumption that the stowage planner makes is that the ideal weight distribution of the containers is to put the heaviest containers on the bottom tier and then place the heaviest containers as close as possible to the middle of each bay. To compute the optimal weights for the vessel locations, we assign containers to vessel locations based on those rules to make a solution with the optimal weight distribution. We then set as the optimal weights for all vessel locations the weight of the container assigned to each vessel location in that solution.

4.4 Regression

To tune the stowage planner parameters, it is important to know the relationship between the parameters and the performance indicator that we use, the QC productivity. We consider this relationship as a black box. Using the stowage planner parameters, we create a stowage plan and then we use emulation to compute the QC productivity. To model the relationship between the stowage planner parameters and the QC productivity we use a simple surrogate model, namely a linear regression.

We use a linear regression as the coefficient from each stowage planner parameter gives an indication of the effects from that parameter on the QC productivity. It indicates if there is a positive or negative effect and also how big that effect is approximately. Another reason that we use a linear regression is the fact that we cannot test hundreds of different stowage planner parameter sets as each emulation takes eighteen hours. More complex regressions need more information to get a good estimate of the coefficients from the extra added complexities and with the limited information available to us collecting more information is not a possibility.

As input for the regression, we use the estimated QC productivities for the different sets of stowage planner parameters. Then we estimate the regression using Ordinary Least Squares (OLS). In OLS we minimize the total squared sum of errors from the observations compared to the equations in (1) (Heij et al., 2004).

$$QC_a^{\text{prod}} = r^0 + r^{CCSPA} b_a^{CCSPA} + r^{CLDPA} b_a^{CLDPA} + r^{QCSPA} b_a^{QCSPA} + r^{YASPA} b_a^{YASPA}, \forall a \in A \tag{1}$$

4.5 Stowage problem model description

In the next sections we explain our Parameter Suggestion Model. We present the Mixed Integer Programming (MIP) formulation that we use to determine the total weight related penalty for a certain set of stowage planner parameters in this section. In Section 4.6 we explain the four heuristics that we use to solve the MIP formulation. Finally, in Section 4.7 we describe the Parameter Suggestion Model that uses the heuristics from Section 4.6.

4.5.1 Objective function and basic restrictions

We present the objective function of our MIP formulation in equation (2a). The goal of the objective is to minimize the total sum of penalties. Furthermore, we make sure that all containers are allocated to one vessel location using equation (2b) and that there is a container assigned to all vessel locations using equation (2c). Equation (2d) defines the binary nature of the x_{cl} variables.

$$\text{minimize} \sum_{f \in F^c} \sum_{c \in C} m_c^f + \sum_{d \in D} m_d^{QCSPE} \tag{2a}$$

$$\sum_{l \in L_s} x_{cl} = 1 \qquad \forall s \in S, c \in C_s \tag{2b}$$

$$\sum_{c \in C_s} x_{cl} = 1 \qquad \forall s \in S, l \in L_s \tag{2c}$$

$$x_{cl} \in \{0, 1\} \qquad \forall s \in S, c \in C, l \in L_s \tag{2d}$$

4.5.2 Restrictions for the weight related stowage planner penalties

We continue by explaining how we model the weight related penalties. These are the OWDPE and the WEIPE penalties. These are the penalties for the difference in the weight of a container with the optimal weight from the vessel location the container is assigned to and for weight inversion.

We model the OWDPE penalty using equation (3a). For each container, this restriction computes the weight difference compared to the optimal weight of the vessel location the container is stowed to. If the container is heavier than the optimal weight, we apply a penalty of π^{OWDPA} multiplied by the weight difference. It is important to note that the stowage planner algorithm recalculates the optimal weight for each container after every container that is assigned a vessel location. This is overly complicated to model in an MIP formulation, as we would have to introduce multiple extra help variables that create extra dependencies. We approximate this penalty by computing the optimal weights only once at the start. This means that the OWDPE penalty computed by the MIP is slightly different from the OWDPE penalty computed by the stowage planner.

Equation (3b) checks for vessel locations if the container allocated to the vessel location above is heavier than the container located to the vessel location itself. If this occurs and both vessel locations belong to the same stowage group, we add a penalty of π^{WEIPA} to the WEIPE penalty.

$$m_c^{OWDPE} \geq \pi^{OWDPA} x_{cl}(w_c - o_l) \qquad \forall s \in S, c \in C_s, l \in L_s \tag{3a}$$

$$m_c^{WEIPE} \geq \pi^{WEIPA}(x_{cl} + x_{ij} - 1) \qquad \forall s \in S, c \in C_s, l \in L_s, i \in C_s, j \in L_s, \nu_{lj} = 1, w_c > w_i \tag{3b}$$

4.5.3 Restrictions for the CCSPE penalty

We continue by modeling the stowage planner penalties that are not weight related. We start with the CCSPE penalty that limits containers coming from the same yard block to the same QC close after each other.

To compute this penalty for each container we make use of help variables. As defined in restriction (4b), b_{ch} indicates whether there is a container from the same yard block in the move $h, 1 \leq h \leq \pi^{CLDPA}$ moves before stowing container $c \in C$. We define the binary nature of b_{ch} in equation (4c). Using these restrictions, we compute the CCSPE penalty for each container in equation (4a). Here we set the penalty for each container c by the lowest h such that b_{ch} equals one as λ_h increases the lower the value of h is.

$$m_c^{CCSPE} \geq \lambda_h \pi^{CCSPA} b_{ch} \qquad \forall c \in C, 1 \leq h \leq \pi^{CLDPA} \tag{4a}$$

$$b_{ch} \geq \gamma_{ci}(x_{cl} + x_{i(l-h)} - 1) \qquad \forall s \in S, c \in C_s, i \in C_s, 1 \leq h \leq \pi^{CLDPA}, l \in L_s, l > h \tag{4b}$$

$$b_{ch} \in \{0, 1\} \qquad \forall c \in C, 1 \leq h \leq \pi^{CLDPA} \tag{4c}$$

4.5.4 Restrictions for the QCSPE and YASPE penalties

The last two penalties where we define restrictions for are the QCSPE and the YASPE penalties. Their aim is to limit the number of QCs containers from one yard stack go to and to limit the number of yard shifts. For the QCSPE penalty we define a restriction in equation (5a), where we add a value of $\frac{\pi^{QCSPA}}{2}$ to the QCSPE penalty each time the stowage solution uses more than one QC to place containers from one yard stack on the vessel. We compute this using the help variable μ_{dq} that indicates whether QC $q \in Q$ is used for containers in yard stack $d \in D$. For this indication we use equation (5b) and we define the binary nature of the help variables μ_{dq} in equation (5c).

Lastly, we define how we compute the YASPE penalty for each container in equation (5d). Again, we use a help variable. We set this help variable α_{ci} to one if container $i \in C$ is stowed before container $c \in C$ in equation (5e). We define the help variable α_{ci} as a binary variable in equation (5f).

$$m_d^{QCSPE} \geq \frac{\pi^{QCSPA}}{2} \left(\sum_{q \in Q} \mu_{dq} - 1 \right) \qquad \forall d \in D \tag{5a}$$

$$\mu_{dq} \geq x_{cl} \qquad \forall d \in D, q \in Q, c \in C_d, l \in L_q \tag{5b}$$

$$\mu_{dq} \in \{0, 1\} \qquad \forall d \in D, q \in Q \tag{5c}$$

$$m_c^{YASPE} \geq \pi^{YASPA} \alpha_{ci} \qquad \forall c \in C, i \in C, e_{ci} = 1 \tag{5d}$$

$$\alpha_{ci} \geq (x_{cl} + x_{ij} - 1)\rho_{lj} \qquad \forall t \in T, c \in C_t, i \in C_t, j \in L, l \in L \tag{5e}$$

$$\alpha_{ci} \in \{0, 1\} \qquad \forall t \in T, c \in C_t, i \in C_t \tag{5f}$$

4.5.5 Entire stowage MIP formulation

$$\min \sum_{f \in F^c} \sum_{c \in C} m_c^f + \sum_{d \in D} m_d^{QCSPE}$$

$$\sum_{l \in L_s} x_{cl} = 1 \qquad \forall s \in S, c \in C_s$$

$$\sum_{c \in C_s} x_{cl} = 1 \qquad \forall s \in S, l \in L_s$$

$$m_c^{OWDPE} \geq \pi^{OWDPA} x_{cl}(w_c - o_l) \qquad \forall s \in S, c \in C_s, l \in L_s$$

$$m_c^{WEIPE} \geq \pi^{WEIPA}(x_{cl} + x_{ij} - 1) \qquad \forall s \in S, c \in C_s, l \in L_s, i \in C_s, j \in L_s, \nu_{lj} = 1, w_c > w_i$$

$$m_c^{CCSPE} \geq \lambda_h \pi^{CCSPA} b_{ch} \qquad \forall c \in C, 1 \leq h \leq \pi^{CLDPA}$$

$$b_{ch} \geq \gamma_{ci}(x_{cl} + x_{i(l-h)} - 1) \qquad \forall s \in S, c \in C_s, i \in C_s, 1 \leq h \leq \pi^{CLDPA}, l \in L_s, l > h$$

$$m_d^{QCSPE} \geq \frac{\pi^{QCSPA}}{2} \left(\sum_{q \in Q} \mu_{dq} - 1 \right) \qquad \forall d \in D$$

$$\mu_{dq} \geq x_{cl} \qquad \forall d \in D, q \in Q, c \in C_d, l \in L_q$$

$$m_c^{YASPE} \geq \pi^{YASPA} \alpha_{ci} \qquad \forall c \in C, i \in C, e_{ci} = 1$$

$$\alpha_{ci} \geq (x_{cl} + x_{ij} - 1)\rho_{lj} \qquad \forall t \in T, c \in C_t, i \in C_t, j \in L, l \in L$$

$$x_{cl} \in \{0, 1\} \qquad \forall s \in S, c \in C, l \in L_s$$

$$\mu_{dq} \in \{0, 1\} \qquad \forall d \in D, q \in Q$$

$$b_{ch} \in \{0, 1\} \qquad \forall c \in C, 1 \leq h \leq \pi^{CLDPA}$$

$$\alpha_{ci} \in \{0, 1\} \qquad \forall t \in T, c \in C_t, i \in C_t$$

4.6 Heuristics to solve the stowage problem

Implementing the MIP formulation that we present in Section 4.5 results in out of memory errors due to the high number of variables and constraints. Therefore, we need to define heuristics to find solutions. We propose four different options. Those options are a Simple Local Search, a Greedy Randomized Adaptive Search Procedure (GRASP), a Simulated Annealing, and a Large Neighborhood Search (LNS) heuristic.

Table 4: Overview of the different heuristics and their characteristics

Heuristic	Characteristic	Section
Simple Local Search	Basic heuristic	Section 4.6.1
GRASP	Uses multiple starting solutions to improve the results	Section 4.6.2
Simulated Annealing	Sometimes accepts a worse solution to improve the results	Section 4.6.3
LNS	Explores larger neighborhoods to improve the results	Section 4.6.4

In Table 4 we give a brief overview of the four heuristics. It shows the main characteristic of the four heuristics and the section in which we further explain the heuristic. The Simple Local Search heuristic is the most simplistic heuristic that we use. The GRASP heuristic uses multiple starting solutions and the LNS heuristics uses larger neighborhoods to find improved results compared to the Simple Local Search heuristic. The Simulated Annealing heuristic sometimes accepts a worse temporary solution to improve the final solution.

4.6.1 Simple Local Search heuristic

The first heuristic that we introduce is our Simple Local Search heuristic. Our Simple Local Search heuristic consists of two parts. A constructive heuristic that creates a first solution and a local search heuristic that improves the solution found by the constructive heuristic. We give a pseudocode of our Simple Local Search heuristic in Algorithm 4.

Algorithm 4: Pseudocode of the Simple Local Search heuristic

```
Output: solution : The solution from the simple local search
Function SimpleLocalSearch():
    solution ← GenerateGreedyRandomSolution(1);          // See Algorithm 5
    solution ← LocalSearch(solution);                    // See Algorithm 6
end
```

In Algorithm 5 we give the pseudocode for our greedy constructive algorithm. This constructive algorithm sorts for each stowage group the containers based on the container weight and the vessel locations based on their optimal weight. Then the algorithm assigns each container randomly to one of the first few vessel locations without an assigned container. The set of options that the algorithm can assign the container to is called the restricted candidate list.

Algorithm 5: Heuristic to generate a (greedy random) solution to the stowage problem

```
Input  : restrictedCandidateListLength : The number of containers considered for
         each vessel location
Output: solution : The solution generated by the function
Function GenerateGreedyRandomSolution(restrictedCandidateListLength):
    foreach stowage group s ∈ S do
        Sort C_s based on the weight of the container;
        Sort L_s based on the optimal container weight of the vessel location;
        foreach container c ∈ C_s do
            Assign to container c a randomly selected vessel location from the first
             restrictedCandidateListLength vessel locations in L_s that are not yet
             assigned to a container;
        end
    end
end
```

For our Simple Local Search heuristic, we make use of a restricted candidate list with a length of one. This makes sure that the algorithm allocates all containers to vessel locations without any weight penalties, because each container has the weight of the optimal weight from its vessel location. If each container has the optimal weight of the vessel location the container is allocated to then there are also no weight inversions within stowage groups.

In the second part of our Simple Local Search heuristic, we improve the solution with a local search. We describe this local search in Algorithm 6. In our local search we try to improve the solution by searching through different neighborhoods. If we find a solution with a lower objective value than the original solution in one of these neighborhoods then we select this solution, otherwise we retain the old solution.

Algorithm 6: Local search heuristic to improve a stowage solution

```
Input  : solution : The solution before the local search
Output: solution : The solution after the local search
Function LocalSearch(solution):
    maxPositionDifference ← number of containers in the largest stowage group;
    while improved solution are found do
        for positionDifference ← 1 to maxPositionDifference do
            Find best swap between containers where the difference between the assigned
             vessel locations equals positionDifference;
            if the best swap results in an improvement of the solution then
                Update solution by swapping the containers;
            end
        end
    end
end
```

Before we define the neighborhoods that we use in our local search, we first explain some of the key characteristics of the stowage problem that we consider. In the stowage problem we assume that the order in which the vessel locations receive containers from the QCs is already known. This means that we can write a solution to the stowage problem as the order of the containers, based on the vessel location the container is assigned to.

We use this to define our neighborhood for the local search. This neighborhood consists of all the solutions where we swap two containers from the same stowage group in the container order. To accelerate the local search, we split this neighborhood in multiple different neighborhoods. One for each position difference in the order between two vessel locations.

Algorithm 6 commences by looping over all the different neighborhoods. For each neighborhood we select the best swap of two containers and update the solution if that swap results in an improvement of the solution. While there are improvements for at least one of the neighborhoods we continue this iterating over the different neighborhoods.

4.6.2 GRASP heuristic

The Simple Local Search heuristic has one big problem and that is that it only searches for solutions that are in the direct neighborhood of the current best found solution. This neighborhood can contain the global optimal solution, however more often than not this is not the case. The heuristic gets stuck in a local optimum, as it cannot reach the optimal solution directly via the different neighborhoods that the heuristic considers. To improve the solution the algorithm finds, it would be good add diversification to the heuristic.

There are different options to improve the diversification of a heuristic. One option is by using a Greedy Randomized Adapted Search Procedure (GRASP). This procedure is proposed by Feo & Resende (1995). The main idea of GRASP is to generate multiple starting solutions and then apply local search to each of those solutions. This generates multiple solutions of which in the end GRASP selects the best solution.

We demonstrate our implementation of GRASP for the stowage problem in Algorithm 7. Again, we use Algorithm 5 to generate greedy solutions. This time, however, we use a restricted

Algorithm 7: Pseudocode for the GRASP heuristic derived from Feo & Resende (1995)

```
Input  : restrictedCandidateListLength : The length of the restricted candicate list
         used to generate the solutions
         maxIterationCount : The number of iteration that the GRASP does
Output: bestSolution : The best solution found by the GRASP
Function GreedyRandomizedAdaptiveSearchProcedure(maxIterationCount,
 restrictedCandidateListLength):
    currentIteration ← 1;
    bestSolution ← GenerateGreedyRandomSolution(restrictedCandidateListLength);
    bestSolution ← LocalSearch(bestSolution);                    // See Algorithm 6
    currentIteration ← currentIteration + 1;
    while currentIteration ≤ maxIterationCount do
        tempSol ← GenerateGreedyRandomSolution(restrictedCandidateListLength);
        tempSol ← LocalSearch(tempSol);                          // See Algorithm 6
        if ObjectiveValue(tempSolution) < ObjectiveValue(bestSolution) then
            bestSolution ← tempSolution;
        end
        currentIteration ← currentIteration + 1;
    end
end
```

candidate list length of more than one to create random solutions. After the greedy algorithm has generated a solution, we apply the local search from Algorithm 6 to improve the solution. We repeat this a fixed number of iterations and after each iteration we keep track of the current best solution.

4.6.3 Simulated Annealing heuristic

Another possibility to add more diversification to a heuristic is to apply Simulated Annealing. One of the first who applied this algorithm was Kirkpatrick et al. (1983). They based their algorithm on the process of annealing, where the temperature of the metal is slowly decreased to create a metal with certain properties. The main idea of the algorithm is to go through different solutions in the neighborhood of the current solution. Based on the temperature the algorithm decides whether it should accept a solution. This temperature decreases slowly which decreases the chance that the algorithm selects a solution.

In Algorithm 8 we give our implementation of Simulated Annealing for the stowage problem. Again, the algorithm first generates a solution using Algorithm 5 with a restricted candidate list length of one. Then it continues applying new iterations of Simulated Annealing until the temperature has decreased to under one. In each iteration the algorithm first selects a stowage group. This selection occurs randomly, however the chance of selecting one of the stowage groups is proportional to the size of the stowage group. We have added this to ensure that the algorithm spreads its time evenly over the containers, as the size of the stowage group can differ quite substantially.

When the stowage group is selected, the algorithm selects randomly two containers from the set of all containers in the stowage group. It then computes the difference in the objective value if the two selected containers are swapped. Using Algorithm 9 the Simulated Annealing heuristic

Algorithm 8: Simulated Annealing derived from Kirkpatrick et al. (1983)

```
Input  : temperature : The starting temperature of the simulated annealing algorithm
         coolingFactor : The rate with which the temperature is set to cool down
Output: solution : The found solution using simulated annealing
Function SimulatedAnnealing(temperature, coolingFactor):
    solution ← GenerateGreedyRandomSolution(1);          // See Algorithm 5
    while temperature > 1 do
        s ← Randomly select stowageGroup where the chance on each stowage group
          depends on the number of containers in the stowage group;
        c1, c2 ← Randomly select two different containers from stowage group s;
        difference ← The difference in the objective value if c1 and c2 are swapped;
        random ← Randomly generated double between 0 and 1;
        if random < GetAcceptanceProbability(temperature, difference) then
            Swap c1 and c2 in solution;
        end
        temperature ← temperature * coolingFactor;
    end
end
```

Algorithm 9: Function to determine the chance that a solution should be accepted for simulated annealing

```
Input  : temperature : The temperature at the time of the function evaluation
         difference : The difference in objective value between the old and new solution
Output: acceptanceProbability: The chance that the new solution is accepted
Function GetAcceptanceProbability(temperature, difference):
    if improvement > 0 then
        acceptanceProbability ← 1;
    else
        acceptanceProbability ← e^(-improvement/temperature)
    end
end
```

computes the probability that it should accept this new solution. If there is an improvement in the objective value, then the heuristic always accepts the new solution and thus sets the acceptance probability to one. Otherwise, the acceptance probability is equal to $e^{-\frac{improvement}{temperature}}$. This is always a value between zero and one. The algorithm then generates a random number between zero and one. If this random number is lower than the acceptance probability, then the algorithm accepts this solution and it swaps the two containers in the current solution. Otherwise, it retains the old solution. After each iteration, the Simulated Annealing heuristic decreases the temperature by multiplying it with the cooling down factor.

4.6.4 Large Neighborhoods Search using a ruin-and-recreate heuristic

Another option to improve the heuristic compared to the Simple Local Search heuristic is to increase the size of the neighborhoods. By increasing the size of the neighborhoods, it becomes easier to escape local minima and this likely results in a better solution. One option to do this is using Large Neighborhood Search (LNS) in combination with a ruin-and-recreate heuristic.

Schrimpf et al. (2000) already applies this to a similar problem. They mention that a LNS heuristic that works in combination with a ruin-and-recreate heuristic each iteration destroys (ruins) a part of the solution, while it keeps the rest of the solution constant. Thereafter, the algorithm recreates a solution for the part of the solution that it destroyed earlier. It does this by solving the MIP formulation.

Algorithm 10: Pseudocode of the LNS using a ruin-and-recreate heuristic.

```
Input  : numberOfIterations : The number of iterations of the algorithm
         replanContainerCount : The number of containers selected to replan
Output: solution : The found solution using LNS with the ruin-and-recreate algorithm
Function BreakAndRepairAlgorithm(numberOfIterations, replanContainerCount):
    solution ← GenerateGreedyRandomSolution(1);            // See Algorithm 5
    currentIterationCount ← 1;
    while currentIterationCount ≤ numberOfIterations do
        s ← Randomly selected stowageGroup where the chance on each stowage group
          depends on the number of containers in the stowage group;
        numberOfContainers ← Minimum(replanContainerCount, (number of containers in s)/2);
        C ← Set of numberOfContainers randomly selected containers from s;
        Optimally replan containers in set C using MIP formulation;
        currentIterationCount ← currentIterationCount + 1;
    end
end
```

In Algorithm 10 we describe our implementation of LNS for the stowage problem. There are two settings that define the algorithm. There is a numberOfIterations setting that defines the number of iterations that the algorithm performs and there is a replanContainerCount setting. The replanContainerCount setting defines the maximum number of containers that LNS selects to be replanned.

Also in this algorithm we generate a first solution with a restricted candidate list length of one using Algorithm 5. In each iteration the algorithm selects randomly one of the stowage groups. The probability that the algorithm selects a certain stowage group is once again proportional to the number of containers that are part of the stowage group. The algorithm then determines how many containers it must select from the stowage group. To ensure that the algorithm does not get stuck trying to select the last few not selected containers we only allow this number to be at maximum half of the containers in the stowage group. After the containers have been selected, the algorithm solves the stowage plan for those containers to optimality given the solution for the other containers. This does not result in optimal solutions for the entire stowage problem, but only locally given a certain situation for the other containers.

4.7 Parameter Suggestion Model

In this section we define the Parameter Suggestion Model that we use to give our stowage planner parameter suggestions for the parameter tuning. In the Parameter Suggestion Model, we use the ComputeWeightRelatedPenalties($y^{CCSPA}, y^{CLDPA}, y^{QCSPA}, y^{YASPA}$, stowageOptimizer) function that we define in Algorithm 11. This function computes the total weight related penalty corresponding to a given set of stowage planner parameters. We compute this total

weight related penalty by solving the stowage problem for the selected set of parameters using one of the four heuristics that we mention in Section 4.6. The stowageOptimizer denotes the heuristic that we use.

Algorithm 11: Function to compute the total weight related penalty

Input : $y^p, \forall p \in P$: The parameter value set constant for stowage planner parameter p
stowageOptimizer : The heuristic used to solve the stowage problem
Output: totalWeightPenalty: The total weight related penalty in a stowage solution
Function `ComputeWeightRelatedPenalties(`$y^{CCSPA}, y^{CLDPA}, y^{QCSPA}, y^{YASPA}$, *stowageOptimizer*`):`
  stowageSolution ← Solution to the stowage problem for parameter values $y^{CCSPA}, y^{CLDPA}, y^{QCSPA}$ and y^{YASPA} using stowageOptimizer;
  totalWeightPenalty ← Total weight related penalty in stowageSolution;
end

We define our Parameter Suggestion Model that we use to compute our parameter suggestions in equations (7a) - (7d). This Parameter Suggestion Model aims to find the values for the stowage planner parameters that maximize the QC productivity, while it limits the weight related infeasibilities. We define the objective of maximizing the estimated QC productivity in equation (7a). We base this objective on the results from the regression that we define in Section 4.4. There are three restrictions on this objective. We define the maximum and minimum values of the stowage planner parameters in equation (7b). We add the restriction on the weight related penalties in equation (7c), where k denotes the maximum weight related penalty that we allow. In equation (7d) we define that the stowage planner parameter values that the Parameter Suggestion Model selects must always be integer.

$$\text{maximize } r^0 + r^{CCSPA}y^{CCSPA} + r^{CLDPA}y^{CLDPA} + r^{QCSPA}y^{QCSPA} + r^{YASPA}y^{YASPA} \tag{7a}$$

$$l^p \leq y^p \leq u^p, \quad \forall p \in P^t \tag{7b}$$

$$\text{ComputeWeightRelatedPenalties}(y^{CCSPA}, y^{CLDPA}, y^{QCSPA}, y^{YASPA}) \leq k \tag{7c}$$

$$y^p \in \mathbb{Z}, \quad \forall p \in P^t \tag{7d}$$

4.7.1 Solving the Parameter Suggestion Model

Finding the solution to the model in equations (7a) - (7d) is not an easy problem as a function evaluation of ComputeWeightRelatedPenalties($y^{CCSPA}, y^{CLDPA}, y^{QCSPA}, y^{YASPA}$, stowageOptimizer) is expensive. Another problem is that all the different heuristics that we use to solve the stowage problem each give solutions that are most likely not optimal. The relationship between the stowage planner parameters and the total weight related penalty is, therefore, only an approximation.

Before we explain the algorithm that we use to get a solution to the Parameter Suggestion Model, we first explain an important property of the problem. We use a linear regression to estimate the effects of the stowage planner parameters on the QC productivity. Thus, the estimated effect of a change in a stowage planner parameter is constant. Therefore, it would be easy to determine the optimal solution if the restriction on the total weight related penalty would be left out. If the effect of stowage planner parameter $p \in P^t$ (r^p) is positive then we

would select the upper bound on that parameter, namely u^p. We would select the lower bound on that parameter (l^p) if the effect of the parameter is negative. We use this observation to determine the initial values for the stowage planner parameters in our algorithm to solve the model. Algorithm 12, that we use to initialize the parameters, demonstrates this.

Algorithm 12: Function to initialize the stowage planner parameters

Output: $y^p, \forall p \in P^t$: The parameter value selected by the function
Function `InitializeParameters()`:
  foreach *parameter* $p \in P^t$ **do**
    if $r^p > 0$ **then**
      $y^p \leftarrow u^p$;
    else
      $y^p \leftarrow l^p$;
    end
  end
end

In Algorithm 13 we define our Parameter Suggestion Optimization heuristic. We first generate an initial solution using Algorithm 12 and then compute the total weight related penalty for this solution. If this total weight related penalty is lower or equal to k, then we accept the solution as our suggestion. Otherwise, we must seek a solution with a lower total weight related penalty.

Algorithm 13: Pseudocode of our Parameter Suggestion Optimization heuristic

Input : stowageOptimizer : The heuristic used to solve the stowage problem
Output: $y^p, \forall p \in P^t$: The parameter suggestion for stowage planner parameter p
Function `GetParameterSuggestion(`*stowageOptimizer*`)`:
  $y^{CCSPA}, y^{CLDPA}, y^{QCSPA}, y^{YASPA} \leftarrow$`InitializeParameters();`
  currentWeightRelatedPenalty ← `ComputeWeightRelatedPenalties(`y^{CCSPA}, y^{CLDPA}, y^{QCSPA}, y^{YASPA}, *stowageOptimizer*`);` `// See Algorithm 11`
  while *currentWeightRelatedPenalty* $> k$ **do**
    bestParameter ← `SelectParameterWithMostImprovement(`*0,* ss^{CCSPA}, $ss^{CLDPA}, ss^{QCSPA}, ss^{YASPA}$, *stowageOptimizer*`);` `// See Algorithm 14`
    if *bestParameter* = *null* **then**
      bestParameter ← `SelectParameterWithMostImprovement(`*-100000,* $\frac{y^{CCSPA}-l^{CCSPA}}{2}, \frac{y^{CLDPA}-l^{CLDPA}}{2}, \frac{y^{QCSPA}-l^{QCSPA}}{2}, \frac{y^{YASPA}-l^{YASPA}}{2}$, *stowageOptimizer*`);` `// See Algorithm 14`
    end
    $y^{bestParameter} \leftarrow y^{bestParameter} - ss^{bestParameter}$;
    currentWeightRelatedPenalty ← `ComputeWeightRelatedPenalties(`$y^{CCSPA}, y^{CLDPA}, y^{QCSPA}, y^{YASPA}$, *stowageOptimizer*`);` `// See Algorithm 11`
  end
end

We make an assumption that helps us select a new set of values for the stowage planner parameters. This assumption is that decreasing the values of the non-weight related stowage planner parameters eventually results in a lower total weight related penalty. In the ideal

situation, where we can solve the stowage problem optimally, this assumption holds according to the following reasoning.

Assume that we have two sets of stowage planner parameters. In the first set of parameter values only the weight related parameters have a positive value, while we set the other parameters to zero. In the second set of stowage planner parameters the weight related parameters have the same value as in the first set, but the non-weight related parameters also have a positive value.

When you optimize the stowage problem for the first set, you only take the weight related penalties into account. Thus, the optimal solution for this parameter set is also the optimal total weight related penalty. If you keep the weight related parameters constant, it is not possible to find a solution with a lower total weight related penalty. In the second set, you also include the other penalties in your optimization. The solution for the first set is also a feasible solution for this set. However, most likely it is possible to improve the overall solution for the second set by trading off weight related penalties for non-weight related penalties. If the non-weight related parameters increase and the weight related parameter remain constant, the total weight related penalty can only increase or remain constant.

In our case, neither the stowage planner nor our heuristics solve the stowage problem optimally. Therefore, there may be cases where the total weight related penalty increases, when we increase one of the non-weight related parameters. We assume that on average our solutions found by the heuristics perform the same as the optimal solutions, meaning that eventually decreasing the non-weight related parameters, results in a lower total weight related penalty.

We base one of the principles that we use to decrease the total weight related penalty on the gradient descent algorithm. According to Ruder (2016), gradient descent algorithms are algorithms that optimize the objective function by using the gradient of the function that it optimizes. Each iteration the algorithm takes a step in the direction of the steepest descent of the objective function until it cannot find a direction that would improve the solution further. We apply this by searching for the parameter that gives the most improvement of the total weight related penalty per estimated QC productivity. In the next paragraph we explain how we compute this improvement.

We first compute this improvement of the total weight related penalty per estimated QC productivity by lowering the stowage planner parameter of interest by one step size (ss^p). We then select the non-weight related stowage planner parameter with the highest improvement. If none of the stowage planner parameters that we tune indicate an improvement for this step size, then we recompute the improvement of the total weight related penalty for a different step size. This time we take half the difference between the lower bound of the stowage planner parameter (l^p) and the current value of the parameter (y^p). This time we select the parameter with the highest improvement, even if this improvement is negative.

Algorithm 14 gives the pseudocode for the parameter selection function that selects the best parameter for given step sizes. The function loops over all the weight related stowage planner parameters that have a value higher than the lower bound plus the stepSize (ss^p) of that parameter. For each of those parameters the function computes the improvement and then it selects the stowage planner parameter with the most improvement if that improvement is high enough.

Next to the step sizes the parameter selection function also has two other inputs. One of those

Algorithm 14: Pseudocode for the parameter selection function

Input : startBestImprovement : Sets how high an improvement should be to be accepted as the current best parameter
stepSize$^p, \forall p \in P^t$: The amount we lower parameter p each step
stowageOptimizer : The heuristic used to solve the stowage problem

Output: bestParameter: The stowage planner parameter that results in the best improvement

Function `SelectParameterWithMostImprovement(`*startBestImprovement, stepSize*^{CCSPA}, *stepSize*^{CLDPA}, *stepSize*^{QCSPA}, *stepSize*^{YASPA}, *stowageOptimizer*`):`

```
bestImprovement ← startBestImprovement;
bestParameter ← null;
foreach parameter p ∈ P^t do
    if y^p ≥ l^p + ss^p then
        y^p ← y^p - stepSize^p;
        penaltyImprovement ← currentWeightRelatedPenalty -
         ComputeWeightRelatedPenalties(y^CCSPA, y^CLDPA, y^QCSPA, y^YASPA,
         stowageOptimizer);
        improvement ← penaltyImprovement / (stepSize^p r^p);
        if improvement > bestImprovement then
            bestImprovement ← improvement;
            bestParameter ← p;
        end
        y^p ← y^p + stepSize^p;
    end
end
end
```

inputs is the startBestImprovement setting. This setting defines how high the best improvement of one of the stowage planner parameters should be for the algorithm to return a stowage planner parameter. If we set startBestImprovement to zero, then the algorithm will give null as output if there is no improvement for any of the non-weight related stowage planner parameters. If we set startBestImprovement to -1000000, then the algorithm will almost always select a stowage planner parameter, even if the best improvement is negative. Furthermore, we have the input stowageOptimizer that denotes the heuristic that we use to solve the stowage problem.

When the algorithm has decided which stowage planner parameter to decrease, we decrease this stowage planner parameter by the step size of the stowage planner parameter. We then recompute the weight related penalty. We repeat this process until we find a solution with a total weight related penalty lower than k. The stowage planner parameter suggestion from the Parameter Suggestion Model then equals to the parameter values that result in a low enough total weight related penalty.

5 Computational experiments

In this section we evaluate the performance of the Iterative Parameter Tuning Method using computational experiments. We first explain the data and benchmarks that we use, then we look at the different heuristics that we propose to model the stowage planner and then we evaluate the Iterative Parameter Tuning Method itself. Lastly, we shortly discuss the results of the computational experiments. We use Java 8 as programming language for the Iterative Parameter Tuning Method. For the LNS heuristic we make use of CPLEX 22.1.0. The laptop that we use for the computations has 32.0 GB RAM and an i7-11850H 2.5 GHz processor.

5.1 Description of the data

To evaluate the performance of our methodology we run emulations using historic data of a real container terminal that is currently live. This container terminal consists of two quays. On the west quay there are five QCs that can simultaneously stow two vessels. Furthermore, there are twelve yard blocks with two RMGs each and there are in total fifteen Shuttle Carriers (SCs).

In the historic data that we use, there are two vessels that are ready to stow containers from and to the yard. Three of the five QCs service the first vessel that arrives at the container terminal with a total of 616 containers on board. These containers must all be discharged to the container yard, while there are 750 containers that must be loaded to the vessel. The two remaining QCs service the second vessel in this part of the container terminal. These QCs discharge 897 containers from the vessel to the container terminal and 485 containers the other way around. There are no containers that stay on the vessel. The data also includes original stowage plans for both vessels, information about the 10,974 containers in the yard blocks on the west quay and activities from trucks on the land side of the container terminal.

Using our methodology we optimize the stowage plan of the first vessel only, as the stowage planner only makes stowage plans for one vessel. It takes approximately seventeen hours to complete all the planned moves for this vessel under the original planning. To ensure that it is possible to load and discharge the entire vessel within one emulation, even for inefficient stowage plans, we use an emulation length of eighteen hours.

Before we can use the data set for our emulations, we have to make some changes to it. The original truck activity information only consists of truck activities for the first ten hours of the emulation. It is important that we have truck activity for the entire eighteen-hour long emulation, as the truck activity ensures that there is a representative workload for the RMGs in the container yard. If we would not add extra truck activity, then the RMGs would have more time to move the containers from and to the seaside of the container terminal. This would limit the number of disruptions caused by the fact that the RMGs of one yard block cannot handle the demand for container moves. Without truck activities, we would get distorted effects from the parameters on the QC productivity. We add extra truck activity by reversing the activity from the first eight hours and then adding that after the first ten hours.

Now we have given a description of the container terminal and the data set that we use, we give some extra information about how we use the stowage planner. For each set of stowage planner parameters, we generate one stowage plan using the stowage planner. We continue

running emulations, until we get three emulations that finish the entire stowage process of the first vessel within eighteen hours. Sometimes we have to run more than three emulations, as there can be disruptions during each emulation that can even completely block an entire CONTROLS run. We approximate the QC productivity corresponding to a set of stowage planner parameter by taking the average of the three finished emulations from that stowage planner parameter set to combat the randomness in emulations. If the Parameter Suggestion Model suggests a set of parameter values for which we already have results, then we do not run addition emulations.

In Table 5 we define the upper and lower bounds on the parameters that we tune. We also give the parameter values from the standard settings of the stowage planner. We can technically set the parameter value to an infinite positive value, but we use the bounds on the parameters in our methodology as starting point for the Parameter Suggestion Model. Therefore, we define the bound in such a way that we expect that the optimal solutions fall within the bounds.

Table 5: Stowage planner parameter information

Parameter	Standard parameter value	Lower bound	Upper bound
CCSPA	6	1	12
CLDPA	8	2	8
QCSPA	70	0	140
YASPA	100	0	200

For the parameters QCSPA and YASPA we set the lower bound to zero, as that is the lowest possible value. We set the upper bound of the two parameters to two times the standard parameter value, since we expect that the optimal parameter value stays close to the standard parameter value. Parameters CCSPA and CLDPA have a positive lower bound, as they both influence the CCSPE penalty. If we would set one of these parameters to their minimum value, then the CCPSE penalty would always become zero. This gives some unwanted effects on the regression, as one of the parameters could be set to any value without influencing the resulting penalty. To make sure this situation does not occur, we set the lower bounds of CCSPA and CLDPA such that we always can have a positive CCSPE penalty. For the parameter CCSPA this is a lower bound of one, but for the CLDPA parameter it is a lower bound of two as its minimum value equals one. We set the upper bound of CCSPA to two times the standard value but set the upper bound of CLDPA to the standard value to keep computation times of the stowage planner down.

Next to the lower and upper bounds of the stowage planner parameters, we also have to define the maximum total weight related penalty that we allow. We base our maximum on the solution that the stowage planner gives if it uses the standard parameters. The total weight related penalty of the stowage planning that the stowage planner creates is 1,897. We set our limit on the total weight related penalty to 2,000. This ensures that the parameter values that we suggest result in stowage plans that have not too much weight infeasibilities.

5.2 Benchmarks

To show the value of our methodology we benchmark our results to two different solutions applied to the same data set. We derive the first benchmark from the original stowage plan from the

data set. With this stowage plan, we run three emulations to estimate the QC productivity that corresponds to this benchmark. We give the results of those three emulations in Table 6. Our estimate of the QC productivity for the first benchmark is the average of the three emulations and that is 29.25 with a standard deviation of 0.39. This stowage plan is not created using the stowage planner directly, but it is optimized by employees from the container terminal. It gives an indication of what kind of QC productivity is achievable. It is also important to notice that the three runs for the first benchmark are successful on the first try. This means that this stowage plan is also robust for issues with the TOS and issues in the emulation software.

Table 6: Overview of the emulation results for the two benchmarks

Benchmark	Run 1	Run 2	Run 3	Average	Standard deviation
Benchmark 1	29.34	29.38	29.04	29.25	0.15
Benchmark 2	28.55	28.15	28.49	28.40	0.18

We also have a second benchmark. This benchmark is the solution of the stowage planner with the standard parameter values that we provide in Table 7 from the common stowage planner strategy of the container terminal. This benchmark provides an indication of the QC productivity that our tuned parameters should reach at minimum, as we can reach that QC productivity without any further tuning. This is the main benchmark we compare our results to. In Table 6 we present the results of the three successful emulations that have an average QC productivity of 28.40 with a standard deviation 0.18. This is almost one container per hour per QC lower than the first benchmark. For this benchmark we need to run five emulations to get three acceptable runs. Thus, this benchmark is also less robust than the first benchmark.

Table 7: Standard parameter values

Parameter type	Parameter	Parameter value
Weight related parameters	OWDPA	200
	WEIPA	501
Tune parameters	CCSPA	6
	CLDPA	8
	QCSPA	70
	YASPA	100

5.3 Evaluating the heuristics to solve the stowage problem

In this section we compare the four different heuristics that we describe in Section 4.6 as options to solve the stowage problem in the Parameter Suggestion Model. We solve the stowage problem for the four heuristics and the stowage planner with the standard parameter values from the second benchmark that we provide in Table 7 to compare the solutions of the different heuristics and the stowage planner to each other.

The first heuristic that we consider is the Simple Local Search heuristic from Section 4.6.1. This heuristic finds a solution with a total penalty of 12,343 in 1.136 seconds. The three other heuristics have settings that we can adjust and we must select those setting before we can use the heuristics in the Parameter Suggestion Model.

For the selection of the settings from the heuristics we take three things into account. We consider the total weight related penalty and the total penalty of the stowage planning that the heuristic generates and the computation time of the heuristic. The aim of the heuristics is to serve as a surrogate model for the stowage planner to approximate the weight related penalty that corresponds to a set of stowage planner parameter values. Therefore, we select the settings of the heuristics in such a way that they get a stowage planning with a total weight related penalty close to the total weight related penalty of 1,897 that the stowage planner realizes as we show in Table 8, where the solution of the stowage planner corresponds to the second benchmark.

Table 8: Total weight related penalty for the stowage solutions from the two benchmarks

Stowage plan	Total weight related penalty
Benchmark 1	506
Benchmark 2	1897

If there are multiple different sets of settings that have a total weight related penalty equally close to the total weight related penalty that the stowage planner reaches, then we select one of those options based on the total penalty and the computation time of the solutions. If possible, we select the option that has the lowest total penalty, as that indicates a better stowage plan, and the shortest computation time. If there is no such option, we select an option based on what we decide is the better overall option. An important note for the selection of the settings for the heuristics is that we base the selection of the settings on one instance of stowage planner parameters. There is no guarantee that the settings that we select also give a good estimation of the total weight related penalty for other stowage planner parameter values.

5.3.1 Settings selection for the GRASP heuristic

The first heuristic where we select the settings for is the GRASP heuristic from Section 4.6.2. For the GRASP heuristic we can select the number of iterations and the length of the restricted candidate list that the algorithm uses to generate the starting solutions of the algorithm. In Table 9 we provide an overview of all the different settings that we test for the GRASP heuristic and the penalties of the corresponding stowage planning. We observe that when the number of iterations increases, the computation time also increases and the total penalty decreases or stays constant. There does not appear to be any significant effect of changing the restricted candidate list length.

We continue by selecting one combination of the two settings that we use in the rest of the thesis. As we see in Table 9, there are two combinations of options that have a total weight related penalty the closest to the total weight related penalty of 1,897 for the second benchmark. That are the combination of a restricted candidate list length of three with ten iterations and a restricted candidate length of three with 25 iterations that both get a total weight related penalty of 1,880. We select the option with the lowest computation time and the same total penalty and thus use a restricted candidate list length of three and ten iterations for the GRASP heuristic from here onwards.

Table 9: Results from the GRASP heuristic for different settings

Number of iterations	Length restricted candidate list	OWDPE	WEIPE	Total weight penalty	CCSPE	QCSPE	YASPE	Total penalty	Computation time (s)
3	2	1260	0	1260	1914	4025	4200	11399	2.418
5	2	1260	0	1260	1914	4025	4200	11399	5.555
10	2	1420	0	1420	2112	3780	3800	11112	11.811
25	2	1780	0	1780	1776	3500	3800	10856	26.471
3	3	1920	0	1920	2130	3710	3600	11360	3.473
5	3	1780	0	1780	1734	3850	3900	11264	5.374
10	3	1880	0	1880	2310	3465	3300	10955	10.788
25	3	1880	0	1880	2310	3465	3300	10955	26.829
3	5	1380	0	1380	2208	3535	3800	10923	3.192
5	5	1380	0	1380	2208	3535	3800	10923	5.503
10	5	1380	0	1380	2208	3535	3800	10923	10.722
25	5	1380	0	1380	2208	3535	3800	10923	26.894
3	10	2280	0	2280	2280	3780	3600	11940	3.109
5	10	2280	0	2280	2280	3780	3600	11940	5.908
10	10	2000	0	2000	2316	3815	3700	11831	10.914
25	10	1780	0	1780	1752	3850	3700	11082	27.518

5.3.2 Settings selection for the Simulated Annealing heuristic

The next heuristic that we select the settings for is the Simulated Annealing heuristic from Section 4.6.3. For the Simulated Annealing heuristic there are two settings that we must set and that are the starting temperature and the cooling factor. In Table 10 we show the ten combinations of settings that result in the total weight related penalty the closest to the weight related penalty that the stowage planner achieves. We present the results for all the settings that we consider in Table 29 in Appendix B. From those results we notice that the closer the cooling factor is to one, the lower the total penalty becomes and the higher the computation time becomes. The starting temperature also has a positive effect on the computation time, as the computation time increases when the starting temperature increases. However, there does not seem to be any effect from the starting temperature on the total penalty.

Table 10: Results from the Simulated Annealing heuristic for different settings

Temperature	Cooling Factor	OWDPE	WEIPE	Total weight penalty	CCSPE	QCSPE	YASPE	Total penalty	Computation time (s)
5	0.999	1800	0	1800	12468	5670	20300	40238	0.010
10	0.999	1960	0	1960	11412	5215	18500	37087	0.010
5	0.9999	2000	0	2000	4686	5110	7100	18896	0.061
10	0.9999	2000	0	2000	4230	4935	6500	17665	0.070
50	0.99999	1820	0	1820	1626	3115	2100	8661	0.555
100	0.99999	1960	0	1960	1812	2730	1900	8402	0.668
500	0.99999	1820	0	1820	1836	2870	1900	8426	0.822
100	0.999999	1780	0	1780	1482	2065	1900	7227	4.982
500	0.999999	1800	0	1800	1542	2205	1700	7247	6.942
10000	0.999999	1760	0	1760	1488	2100	1800	7148	10.464

For the selection of the settings for the Simulated Annealing heuristic we again select the combination of settings with the weight related penalty the closest to that of the stowage planner. There are two sets of settings that are the closest to the total penalty of 1,897 with a total penalty of 1,960. The option with a starting temperature of ten and a cooling factor of 0.9999 has a high total penalty with 37,087 and a low computation time with 0.010 seconds. The other options with a starting temperature of 100 and a cooling factor of 0.99999 still has a low computation

time of only 0.668 seconds and much lower total penalty of only 8,402. Therefore, we select the second option and use a starting temperature of 100 and a cooling factor of 0.99999 in the rest of this thesis for the Simulated Annealing heuristic.

5.3.3 Settings selection for LNS

Lastly, we also select the settings for the LNS heuristic that we describe in Section 4.6.4. We have to select the number of iterations that the LNS heuristic does and the number of containers that the LNS heuristics replans optimally each iteration. In Table 11 we present the settings that result in a total weight related penalty that is close to the total weight related penalty of the stowage planner. We see that the higher the number of iterations and the higher the number of containers we consider, the higher the computation time is, but also the lower the total penalty is. We give the full results for all the settings options that we consider in Table 30 in Appendix B.

Table 11: Results from the LNS heuristic for different settings

Number of iterations	Number of containers	OWDPE	WEIPE	Total weight penalty	CCSPE	QCSPE	YASPE	Total penalty	Computation time (s)
100	10	1900	0	1900	10218	5460	18200	35778	10.200
100	30	1820	0	1820	3654	3850	6200	15524	113.553
200	25	1820	0	1820	2496	3360	3400	11076	96.821
200	30	1880	0	1880	2646	3360	2900	10786	176.469
300	5	1880	0	1880	10014	6090	19500	37484	16.863
300	10	2020	0	2020	5388	4830	8700	20938	23.386
300	20	1800	0	1800	2844	3325	3500	11469	221.507
500	5	1900	0	1900	7830	5705	15300	30735	20.905
500	15	1840	0	1840	2682	3325	3800	11647	89.491
500	20	1840	0	1840	2082	3255	2900	10077	255.297

Again, there are two sets of settings that have the closest total weight related penalty to the total weight related penalty that the stowage planner reaches. We get a total weight related penalty of 1,900 for 100 iterations and ten considered containers and for 500 iterations and five considered containers. The computation time of the second option is two times the computation time of the first option, while the second option reaches a total penalty that is 5,043 higher than the total penalty of the first option. There is no clear best option, however we select the first option with the lower computation time, as the second option still has a relatively high total penalty that is higher than the total penalty that the stowage planner reaches. Thus, from here on we use 100 iterations and we consider ten containers in each iteration of the LNS heuristic.

5.3.4 Comparison between the different heuristics

Now we have selected the settings for all the heuristics, we can compare the heuristics to each other and to the stowage planner. For each heuristic we give the resulting penalty values of the stowage plan that the heuristic creates for the standard parameters in Table 12. We also add the resulting penalty values of the stowage plans from the two benchmarks, where the second benchmark corresponds with the stowage planner solution. If we compare our heuristics, we see that the Simple Local Search heuristic and Simulated Annealing heuristic have the shortest computation times. The Simulated Annealing heuristic also reaches the lowest total penalty

with 8,402. The second-best heuristic in terms of the total penalty is GRASP with a total penalty of 10,955.

We observe that the stowage plan made by LNS has a total penalty of 35,778. That is more than 20,000 higher than the total penalty of the other heuristics despite it being one of the slower heuristics. The fact that each iteration takes approximately 0.1 seconds and only changes ten containers is the main cause for this high penalty. There are not enough opportunities to improve the solution far enough.

Table 12: Stowage planning result for different Algorithms

Algorithm	OWDPE	WEIPE	Total weight penalty	CCSPE	QCSPE	YASPE	Total penalty	Computation time (s)
Benchmark 1	506	0	506	2634	2625	7200	12965	-
Benchmark 2	394	1503	1897	1566	2870	16700	23033	164.88
Simple Local Search	1640	0	1640	2448	3955	4300	12343	1.136
GRASP	1880	0	1880	2310	3465	3300	10955	10.788
Simulated Annealing	1960	0	1960	1812	2730	1900	8402	0.668
LNS	1900	0	1900	10218	5460	18200	35778	10.200

If we compare the stowage plans from the heuristics with the benchmarks, we see that, except for LNS, they all achieve a lower total penalty than the benchmarks do. Here it is important to notice that our implementation of the PENTWE parameter differs a little bit from the implementation used in the stowage planner, as we explain in Section 4.5.2, which could cause a small difference in the penalties mentioned for the benchmark compared to the heuristics. Especially for the second benchmark, which is the solution that the stowage planner gives for the standard parameters, we see a difference in the total penalty of more than 10,000 compared to the total penalties of the heuristics. This shows that there is a significant difference between the solutions of the heuristics and the solution from the stowage planner.

We can see that same difference if we look specifically at the WEIPE penalty from the different solutions. The solution from the stowage planner has a WEIPE penalty of 1,503, while the solutions from the heuristics all have a WEIPE penalty of zero. There are also big differences in the other penalties, however this difference does not have to be a problem. The goal of the heuristics is to mimic the performance of the stowage planner to estimate the total weight related penalty for a certain parameter set. This total weight related penalty is almost equal to the total weight related penalty that the stowage planner reaches for most heuristics due to the way we select the settings for the heuristics. For the Simple Local Search heuristic this does not hold, as we cannot set the any settings for this heuristic. Therefore, the Simple Local Search heuristic has a total weight related penalty of 1,640 which is 257 lower than the total weight related penalty of the stowage planner.

We also examine the stowage plan with the lowest total penalty that we find for each of the heuristics for all the possible settings that we give in Table 13. Here we observe that the Simulated Annealing heuristic reaches the lowest total penalty with a total penalty of 6,359. The LNS heuristic also reaches quite a low total penalty with a total penalty of 9,035. Both these methods do take quite a long time to reach those penalties. The total penalty of solution from the stowage planner, benchmark 2, is almost than 14,000 higher than the total penalty of the stowage plan of both the LNS heuristic and the Simulated Annealing heuristic. This shows that the stowage planner is not able to find a good solution for the stowage problem.

Table 13: Stowage plans with the lowest found total penalties for the different heuristics

Algorithm	OWDPE	WEIPE	Total weight penalty	CCSPE	QCSPE	YASPE	Total penalty	Computation time (s)
Benchmark 1	506	0	506	2634	2625	7200	12965	-
Benchmark 2	394	1503	1897	1566	2870	16700	23033	164.88
Simple Local Search	1640	0	1640	2448	3955	4300	12343	1.136
GRASP	1780	0	1780	1776	3500	3800	10856	26.471
Simulated Annealing	1420	0	1420	1554	1785	1600	6359	319.982
LNS	1740	0	1740	2190	2905	2200	9035	234.235

There are multiple reasons for this difference. One of the reasons is that the stowage planner considers more parameters and penalties than that we do. Consequently, the stowage planner trades some of the performance on the penalties that we consider off against the extra penalties that the stowage planner considers, which results in higher penalties for the parameter that we consider. Another reason for this is the low parameter value of five for both the SearchDepth parameter and SearchWidth parameter in the standard parameters. The stowage planner would give better results for higher values of those parameters. A last reason for this difference is that the stowage planner only consists of one iteration as we explain in Algorithm 1 in Section 3.3. This means that the stowage planner has not enough opportunities to reach an optimal solution.

5.4 Evaluation of the Iterative Parameter Tuning Method

In this section we evaluate the performance of the Iterative Parameter Tuning Method that we develop. We first apply the Iterative Parameter Tuning Method with an initialization phase that consists of fifteen sets of stowage planner parameter values. Those fifteen sets contain the original parameter set and fourteen corner points to cover the entire feasible parameter space. Six of those corner points have parameter values for the CCSPA parameter and the CLDPA parameter that are not on the lower or upper bound of the parameter value to add a difference between the two parameters for the regression. This is needed because for those two parameters it is not possible to have one parameter with the lower bound value and the other parameter with the upper bound value, as this would always result in a zero penalty for the CCSPE penalty.

In Table 14 we show the results from the emulations for the fifteen selected parameter sets. In this table we see that we often need more than ten emulations for the parameter sets with low CCSPA and CLDPA parameter values to get three successful emulation runs. For the other parameter sets we often only need four to seven emulations to get the three successful runs. This indicates that having low values for the CCSPA and the CLDPA parameter results in less robust stowage plans.

The set of parameter values that reaches the highest average QC productivity of 29.26 over its three successful emulation runs is the set with a CCSPA parameter of twelve, a CLDPA parameter of eight, a QCSPA parameter of 140 and a YASPA parameter of zero. The worst average QC productivity is 27.01 and that is for the parameter set with a CCSPA parameter of zero, a CLDPA parameter of one, a QCSPA penalty of 140 and a YASPA parameter of 200.

If we look at the total weight related penalty of the parameters sets, we see that there are three sets that have a total weight related penalty above the limit of 2,000 that we have set. This shows that considering this total weight related penalty is useful, as there are parameter

Table 14: Emulation results for the large set of parameters sets

Parameter values				Number	QC	QC	QC	Average	Standard	Total weight
CCSPA	CLDPA	QCSPA	YASPA	of runs	Prod 1	Prod 2	Prod 3		deviation	penalty
0	1	0	0	12	27.69	27.37	26.84	27.30	0.35	543
0	1	0	200	6	29.43	28.28	29.65	29.12	0.60	1849
0	1	140	0	15	28.45	28.12	27.71	28.09	0.30	551
0	1	140	200	16	26.17	27.43	27.42	27.01	0.59	2470
12	8	0	0	11	28.43	29.46	29.35	29.08	0.46	535
12	8	0	200	8	29.01	28.93	29.65	29.20	0.32	1516
12	8	140	0	5	28.23	29.77	29.78	29.26	0.73	719
12	8	140	200	7	29.53	28.20	27.62	28.45	0.80	2624
6	8	0	0	7	27.62	28.00	27.37	27.66	0.26	46
6	8	140	200	5	27.8	28.16	27.97	27.98	0.15	3071
6	4	0	0	4	27.76	28.67	27.76	28.06	0.43	543
6	4	140	200	5	27.31	27.71	27.53	27.52	0.16	1849
12	4	0	0	4	27.2	28.30	27.08	27.53	0.55	543
12	4	140	200	7	27.07	27.39	27.6	27.35	0.22	1865
6	8	70	100	5	28.55	28.15	28.49	28.40	0.18	1897

sets that are not feasible. We give a full overview of all the penalties of the stowage plan that the stowage planner generates for the all the parameter sets in Table 31 of Appendix C.

We present the results from the first regression on the emulation results from Table 14 in Table 15. This regression has an R^2 of only 0.200, which indicates that the regression can only explain 20% of the variance in the emulation results. The only parameter with a coefficient that is significantly different from zero with 90% confidence is the CLDPA parameter. Most parameters have a positive effect on the QC productivity, except for the QCSPA parameter that has a negative effect. The regression indicates that QC productivity improves with 0.003 if we decrease the parameter value of QCSPA with one. In the next subsections we analyze the

Table 15: Results of the regression after the initialization phase

Constant	CCSPA	CLDPA	QCSPA	YASPA	R^2
27.670** (0.285)	0.001 (0.037)	0.116* (0.060)	-0.003 (0.002)	0.001 (0.001)	0.200

Standard errors between parentheses, * $p < 0.10$, ** $p < 0.05$

performance of the Iterative Parameter Tuning Method for the four different heuristics that we use in the Parameter Suggestion Model to model the stowage planner. Thereafter, we also apply the Iterative Parameter Tuning Method using less sets of parameters in the initialization phase.

5.4.1 Iterative Parameter Tuning Method using the Simple Local Search heuristic

We present the parameter suggestion from the Parameter Suggestion Model with the Simple Local Search heuristic in Table 16 and the results from the regressions that we use as input for the Parameter Suggestion Model in Table 17. The Iterative Parameter Tuning Method ends in three iterations, as it generates the same parameter suggestion in every iteration. To compute the QC productivity corresponding to the parameter suggestion we need four emulation runs, which indicates that this parameter suggestion creates a robust stowage plan. The stowage plan that the stowage planner creates for the parameter suggestion is also feasible as the total weight related penalty of the stowage plan is below the boundary of 2,000. We give an overview of all the penalty values for the solution that the stowage planner creates in Appendix C Table 32.

Table 16: Emulation results of the parameter suggestions for the Simple Local Search heuristic

Iteration	Parameter values				Number of runs	QC Prod 1	QC Prod 2	QC Prod 3	Average	Standard deviation	Total weight penalty
	CCSPA	CLDPA	QCSPA	YASPA							
1-3	6	5	0	200	4	28.94	28.06	28.50	28.50	0.36	1356

Table 17: Results of the regressions for the Simple Local Search heuristic

Iteration	Constant	CCSPA	CLDPA	QCSPA	YASPA	R^2
1	27.670** (0.285)	0.001 (0.037)	0.116* (0.060)	-0.003 (0.002)	0.001 (0.001)	0.200
2	27.677** (0.276)	0.001 (0.035)	0.116* (0.058)	-0.003 (0.002)	0.001 (0.001)	0.205
3	27.681** (0.268)	0.001 (0.035)	0.117** (0.057)	-0.003* (0.002)	0.001 (0.001)	0.210

Standard errors between parentheses, * $p < 0.10$, ** $p < 0.05$

The average QC productivity of the final parameter suggestion from the Iterative Parameter Tuning Method using the Simple Local Search heuristic is 28.50. This is only slightly higher than the QC productivity of 28.40 for the second benchmark. If we look at the regressions in Table 17 we see no substantial changes in the second and third iteration compared to the first iteration. The only difference is that in the third iteration the coefficient of the QCSPA parameter becomes significantly different from zero with a confidence level of at least 90%.

5.4.2 Iterative Parameter Tuning Method using the GRASP heuristic

In Table 18 we present the three parameter suggestions from the Parameter Suggestion Model with the GRASP heuristic. For the computation of the average QC productivity, we need four emulation runs, which means that also for the GRASP heuristic we get a robust parameter suggestion. With a total weight related penalty of only 366, the stowage planner generates a feasible stowage planning. Table 33 in Appendix C gives the full overview of all the penalties for that stowage planning.

Table 18: Emulation results of the parameter suggestions for the GRASP heuristic

Iteration	Parameter values				Number of runs	QC Prod 1	QC Prod 2	QC Prod 3	Average	Standard deviation	Total weight penalty
	CCSPA	CLDPA	QCSPA	YASPA							
1-3	12	6	0	200	4	29.20	29.24	30.10	29.51	0.42	366

Table 19: Results of the regressions for the GRASP heuristic

Iteration	Constant	CCSPA	CLDPA	QCSPA	YASPA	R^2
1	27.670** (0.285)	0.001 (0.037)	0.116* (0.060)	-0.003 (0.002)	0.001 (0.001)	0.200
2	27.664** (0.287)	0.001 (0.037)	0.126** (0.060)	-0.004** (0.002)	0.001 (0.001)	0.259
3	27.661** (0.285)	0.001 (0.037)	0.133**(0.059)	-0.005** (0.002)	0.002 (0.001)	0.310

Standard errors between parentheses, * $p < 0.10$, ** $p < 0.05$

The stowage planning that the stowage planner creates for the parameter suggestion reaches an average QC productivity of 29.51. This is more than one container per hour per QC higher than the QC productivity of the second benchmark and it is even higher than the QC productivity of the first benchmark. Thus, the Iterative Parameter Tuning Method with the GRASP heuristic can tune the stowage planner parameters in three iterations to such a level that it can compete with the original planning for the data set. In Table 19 we present the results from the regressions that are the input to the Parameter Suggestion Model. The directions of the

effects stay the same across the different iterations, however in the second and third iteration we see that the effects of both the CLDPA parameter and the QCSPA parameter are significantly different from zero with a confidence level of at least 95%.

5.4.3 Iterative Parameter Tuning Method using the Simulated Annealing heuristic

We continue with the Iterative Parameter Tuning method with the Simulated Annealing heuristic of which we give the parameter suggestions of the different iterations in Table 20. Once again, we only need three iterations of the Iterative Parameter Tuning Method as the Parameter Suggestion Model suggest three times the same parameter values. We also again only need four emulation runs to get three successful runs. In Table 34 of Appendix C we give an overview of the penalties from the solution that the stowage planner creates for the parameter suggestion. The total weight related penalty of this solution is under 2,000, which means that this solution is feasible.

Table 20: Emulation results of the parameter suggestions for the Simulated Annealing heuristic

Iteration	Parameter values				Number of runs	QC Prod 1	QC Prod 2	QC Prod 3	Average	Standard deviation	Total weight penalty
	CCSPA	CLDPA	QCSPA	YASPA							
1-3	10	8	0	200	4	28.76	29.62	28.55	28.98	0.46	1877

Table 21: Results of the regressions for the Simulated Annealing heuristic

Iteration	Constant	CCSPA	CLDPA	QCSPA	YASPA	R^2
1	27.670** (0.285)	0.001 (0.037)	0.116* (0.060)	-0.003 (0.002)	0.001 (0.001)	0.200
2	27.661** (0.279)	0.001 (0.036)	0.121** (0.058)	-0.003 (0.002)	0.001 (0.001)	0.237
3	27.656** (0.273)	0.001 (0.035)	0.123** (0.056)	-0.003 (0.002)	0.001 (0.001)	0.266

Standard errors between parentheses, * $p < 0.10$, ** $p < 0.05$

The average QC productivity of the three successful emulations is 28.98. This is higher than the second benchmark, but lower than the first benchmark. In Table 21 we present the results of the regressions that we use. As in the previous two sections, there are no big differences between the regressions in the different iterations. This is an indication that the set of parameter sets that we define in the initialization phase gives a good indication of how the QC productivity reacts on the stowage planner parameters.

5.4.4 Iterative Parameter Tuning Method using the LNS heuristic

The last heuristic where we apply the Iterative Parameter Tuning Method for is the LNS heuristic and we show the results of this in Table 22. The Iterative Parameter Tuning Method with the LNS heuristic needs seven iterations before it converges. The parameter values in the row of iteration 5-7 are the final parameter tuning suggestions from the Iterative Parameter Tuning Method. The total weight related penalty of all iterations stays below 2,000 meaning that all the parameter suggestion result in feasible stowage plan. In Table 35 from Appendix C we give a full overview of all the penalties from the stowage plan that the stowage planner creates for that iteration.

In each iteration we need approximately ten emulation runs to get three successful runs. This indicates that the stowage plans that we create using the stowage planner with this method are

Table 22: Emulation results of the parameter suggestions for the LNS heuristic

Iteration	Parameter values				Number of runs	QC Prod 1	QC Prod 2	QC Prod 3	Average	Standard deviation	Total weight penalty
	CCSPA	CLDPA	QCSPA	YASPA							
1	6	5	0	140	8	26.90	28.77	28.04	27.90	0.77	1386
2-3	8	4	0	190	11	28.57	28.16	27.23	27.99	0.56	1849
4	1	8	0	200	9	28.11	28.34	28.62	28.36	0.21	1364
5-7	12	8	0	0	11	28.43	29.46	29.35	29.08	0.46	535

not robust to disruptions in the emulation. Over the iterations we see that the QC productivity increases from 27.90 to 29.08. The final parameter suggestion has a QC productivity that is higher than the QC productivity of the second benchmark. This parameter suggestion even almost reaches the QC productivity of the first benchmark. Based on Table 22 we also note that the parameter suggestions for the second and third iteration are the same, but that the parameter suggestion changes for the fourth iteration. This shows the reason that we only say that the parameter suggestions converge after three parameter suggestion that are the same. It is possible that the parameter suggestion still changes in a next iteration if we get the same parameter suggestion multiple iterations in a row.

Table 23: Results of the regressions for the LNS heuristic

Iteration	Constant	CCSPA	CLDPA	QCSPA	YASPA	R^2
1	27.670** (0.285)	0.001 (0.037)	0.116* (0.060)	-0.003 (0.002)	0.001 (0.001)	0.200
2	27.633** (0.283)	0.003 (0.037)	0.114* (0.060)	-0.002 (0.002)	0.000 (0.001)	0.180
3	27.622** (0.276)	0.001 (0.035)	0.118** (0.058)	-0.002 (0.002)	0.000 (0.001)	0.174
4	27.614** (0.271)	-0.001 (0.035)	0.120** (0.056)	-0.002 (0.002)	0.000 (0.001)	0.170
5	27.619** (0.264)	0.006 (0.028)	0.109** (0.046)	-0.002 (0.002)	-0.001 (0.001)	0.171
6	27.632** (0.261)	0.011 (0.028)	0.114** (0.046)	-0.002 (0.002)	-0.000 (0.001)	0.206
7	27.641** (0.258)	0.013 (0.027)	0.117** (0.045)	-0.002 (0.002)	-0.001 (0.001)	0.237

Standard errors between parentheses, * $p < 0.10$, ** $p < 0.05$

We show the results of the regressions that we use for the Iterative Parameter Suggestion Model with the LNS heuristic in Table 23. In this table we see that for the parameters CCSPA and YASPA the sign of the coefficient changes. In the fourth iteration the sign of the CCSPA parameter changes for one iteration from positive to negative. We see the effect of this change in Table 22, where the parameter suggestion in the fourth iteration is one while it is eight and twelve in the previous and next iteration. For the YASPA parameter the sign changes from positive to negative from iteration five onwards. In Table 22 we see that the suggestion for the YASPA parameter changes to zero as of iteration five.

5.4.5 Comparison between the different heuristics

In this section we compare the final parameter suggestions that the Iterative Parameter Tuning Method suggests for the different heuristics. In Table 24 we give an overview of those final parameter suggestions and some statistics about those parameter suggestions. We notice that the Iterative Parameter Tuning Method gives a different parameter advice for each heuristic. One of the parameter suggestions differs substantially from the other three parameter suggestions. That is the parameter suggestion using the LNS heuristic which is the only parameter suggestion with a YASPA parameter of zero. The stowage plan following from this parameter suggestion is the least robust stowage plan as we need eleven emulations to get three successful runs, but it still reaches a high average QC productivity of 29.08.

Table 24: Overview of the final parameter suggestions for the different heuristics and the benchmarks we compare the final parameter suggestions to

Heuristic	Final parameter suggestion				Number of iterations	Number of runs	Average QC Prod	Total weight penalty	Computation time per iteration (s)
	CCSPA	CLDPA	QCSPA	YASPA					
Simple Local Search	6	5	0	200	3	4	28.50	1356	6.76
GRASP	9	7	0	200	3	4	29.51	366	159.85
Simulated Annealing	10	8	0	200	3	4	28.98	1877	3.66
LNS	12	8	0	0	7	11	29.08	535	140.10
Benchmark 1	-	-	-	-	-	3	29.25	506	-
Benchmark 2	-	-	-	-	-	5	28.40	1897	-

The other three heuristics differ only in the CCSPA and CLDPA parameters. The cause for this difference is the fact that whether a parameter suggestion gets accepted, depends on the heuristic that we use. The Simulated Annealing heuristic seems to accept a parameter suggestion for higher parameter values than the GRASP heuristic and the Simple Local Search heuristic as it has a final parameter suggestion with higher parameter values than the other heuristics.

One explanation for this difference in when a parameter suggestion gets accepted can be that a heuristic that reaches a better stowage planning in terms of the total penalty accepts a parameter suggestion earlier. The reasoning for this could be that a stowage planning with a lower total penalty on average also has a lower total weight related penalty. As we show in Table 12, for the standard parameter values the Simulated Annealing heuristic reaches the lowest total penalty, while the GRASP heuristic has the second lowest total penalty and the Simple Local Search heuristics the highest total penalty of the three heuristics. This is the same order that we have in Table 24 in terms of the boundary when a heuristic accepts a parameter suggestion.

Overall, the heuristic that results in the best parameter suggestion is the GRASP heuristic. The parameter suggestion from the GRASP heuristic has the highest average QC productivity, needs only four emulation runs for three successful emulations and has the lowest total weight related penalty. The Iterative Parameter Tuning Method also only needs three iterations with the GRASP heuristic. The only disadvantage of the GRASP heuristic is the relatively long computation time of on average 159.85 seconds per iteration. Compared to the time an emulation takes this still is a short time.

The LNS heuristic results is the second-best parameter estimate in terms of the average QC productivity. However, the Iterative Parameter Tuning Method has to perform more iterations with the LNS heuristic than with all the other heuristics. One cause for this could be that the LNS heuristic reaches stowage plans with a total penalty substantially worse than that of the other heuristics. As we have explained earlier in this section, there seems to be an effect between the total penalty that a heuristic reaches and the point where the Parameter Suggestion Model accepts a parameter suggestion. This would mean that parameter suggestion using the LNS heuristic get accepted less, resulting in worse parameter suggestions in terms of the average QC productivity. Therefore, the LNS heuristic needs more iterations to get to good parameter suggestions.

Also, the number of emulations needed to get three successful emulation runs indicates that using the LNS heuristic is not one of the best options. The cause for those extra emulation runs is most likely the fact that the final parameter suggestion using the LNS heuristic suggests a parameter value of zero for the YASPA parameter, causing extra yard shifts and thus extra disruptions in the emulation.

5.4.6 Shorter initialization phase

As we show in Table 24, for three of the four heuristics, we only need three iterations to get convergence. This is the lowest possible number of iterations for our Iterative Parameter Tuning Method as we define convergence as getting the same parameter suggestion for three iterations in a row. The fact that we can find a parameter suggestion in the lowest possible number of iterations indicates that we possibly do not need all the sets of parameter values from Table 14 to get good parameter suggestions. This would reduce the computation time of the Iterative Parameter Tuning Method, as we would have to run less emulations in the initialization phase. Therefore, we also apply the Iterative Parameter Tuning Method with only the five parameter sets that we give in Table 25. We do this only with the Simulated Annealing heuristic.

Table 25: Emulation results for the small set of parameters sets

Parameter values				Number of runs	QC Prod 1	QC Prod 2	QC Prod 3	Average	Standard deviation	Total weight penalty
CCSPA	CLDPA	QCSPA	YASPA							
0	1	0	0	12	27.69	27.37	26.84	27.30	0.35	543
0	1	0	200	6	29.43	28.28	29.65	29.12	0.60	1849
0	1	140	0	15	28.45	28.12	27.71	28.09	0.30	551
12	8	0	0	11	28.43	29.46	29.35	29.08	0.46	535
6	8	0	0	7	27.62	28.00	27.37	27.66	0.26	46

We present the parameter suggestions in the different iterations of the Iterative Parameter Tuning Method with the Simulated Annealing heuristic in Table 26. Compared to the original larger initialization phase, we now need two extra iterations to get to convergence. The final parameter suggestion is exactly the same parameter suggestion as we get for the original larger initialization phase. This shows that we do not need an initialization phase with fifteen sets of parameters, but that we can also reach the same results with an initialization phase with five sets of parameters. Overall, this decreases the number of emulations that we need to run and thus decreases the total computation time of the entire Iterative Parameter Tuning Method.

Table 26: Emulation results of the parameter suggestions with a shorter initialization phase

Iteration	Parameter values				Number of runs	QC Prod 1	QC Prod 2	QC Prod 3	Average	Standard deviation	Total weight penalty
	CCSPA	CLDPA	QCSPA	YASPA							
1	12	2	140	200	9	26.83	29.29	26.19	27.44	1.34	1849
2	1	8	0	200	9	28.11	28.34	28.62	28.36	0.21	1364
3-5	10	8	0	200	4	28.76	29.62	28.55	28.98	0.46	1877

One observation that we can make on the parameter suggestions from the first three iterations of the Iterative Parameter Tuning Method is that they differ significantly from each other each iteration. The cause of this are the results from the regression that we show in Table 27 that contain multiple changes of the sign of the coefficients. The first regression with the shorter initialization phase differs substantially from the first regression of the original initialization

phase. The first regression with the shorter initialization phase has a negative coefficient for the CLDPA parameter, while this effect is positive for the original initialization phase as can be seen in Table 15. Also the sign of the QCSPA parameter differs in comparison to the first regression for the original initialization phase.

Table 27: Results of the regressions with a shorter initialization phase compared to the result of the final regression with the original initialization phase from Table 21 for the Simulated Annealing heuristic

Iteration	Constant	CCSPA	CLDPA	QCSPA	YASPA	R^2
Original	27.656** (0.273)	0.001 (0.035)	0.123** (0.056)	-0.003 (0.002)	0.001 (0.001)	0.266
1	27.450** (0.347)	0.236** (0.069)	-0.150 (0.102)	0.006* (0.003)	0.009** (0.002)	0.760
2	27.716** (0.738)	-0.051 (0.094)	0.135 (0.188)	-0.002 (0.006)	0.004 (0.004)	0.132
3	28.066** (0.522)	0.004 (0.053)	0.018 (0.092)	-0.004 (0.005)	0.002 (0.002)	0.103
4	28.020** (0.486)	0.011 (0.048)	0.024 (0.086)	-0.004 (0.004)	0.002 (0.002)	0.164
5	28.001** (0.459)	0.013 (0.045)	0.026 (0.082)	-0.004 (0.004)	0.002 (0.002)	0.206

Standard errors between parentheses, * $p < 0.10$, ** $p < 0.05$. The Original row shows the results of the final regression with the original initialization phase. The other rows show the results of the regressions with a shorter initialization phase.

As the first regression does not give a good estimation of the effects from the stowage planner, the first parameter suggestion is not of the highest quality. We can see this in the average QC productivity of the first iteration that is only 27.44, while we also need nine emulation runs to get three successful runs. In the second iteration we see that the signs of the coefficients become more similar to the results from the regressions for the original initialization phase. This results in a better parameter suggestion with an average QC productivity of 28.36. In the third iteration the signs of the coefficients become the same as the signs of the regressions for the original initialization phase. This shows that the Iterative Parameter Tuning Method corrects itself if the first regression gives incorrect results. Those results provide low quality parameter suggestions, which give more information for the next iteration. This then improves the regression results and following parameter suggestions.

5.5 Discussion of the computational experiments

The final goal of our Iterative Parameter Tuning Method is to automate the parameter tuning for an algorithm, specifically in this case for the stowage planner. For our computational experiments, we have not yet automated the different steps of the method. We only test whether the steps that we would take to automate the entire process, would result in good parameter tuning. If we use a long initialization phase with fifteen different sets of parameters, the Iterative Parameter Tuning Method gives for all the heuristics that we use a final parameter suggestion which reaches an average QC productivity higher than the second benchmark, the stowage plan that the stowage planner makes with the original parameter values. For the final parameter suggestion that the Iterative Parameter Tuning Method creates with the GRASP heuristic, the corresponding stowage plan even reaches an average QC productivity higher than the first benchmark, the original planning from the data set.

The high average QC productivities from the final parameter suggestion demonstrate that the Iterative Parameter Tuning Methods works as we expect it to work. After we have selected

a heuristic and the settings of the heuristic, we can tune the parameters of the stowage planner without the need to take any decisions ourselves except for the selection of the parameter sets in the initialization phase. The parameter tuning also results in feasible and robust stowage plans that can reach a high QC productivity.

There is one disadvantage of our Iterative Parameter Tuning Method and that is the total computation time of the method. In our application of the Iterative Parameter Tuning Method with the GRASP heuristic for the initial large initialization phase, we use a total of 121 emulation runs to get three successful runs for each set of parameters values that we consider. If this is not done in parallel, this would take at least 90 days to complete the Iterative Parameter Tuning Method, as one emulation takes eighteen hours in this case. Even if we use the shorter initialization phase from Section 5.4.6, we still use 73 emulations and this would still take at least 54 days. This computation time can theoretically be shortened to around a week, if we first do all emulation runs from the initialization phase in parallel and then for each iteration all emulations for that iteration in parallel.

There are also other problems that we do not consider in our computational experiments. One of those problems is that we only investigate the parameter tuning of four parameters and that we only consider five penalties. The real stowage planner uses more parameters and penalties. We expect that the Iterative Parameter Tuning Method still performs the same, if we add those extra parameters and penalties, but there is no guarantee for that due to the added extra complexity.

We also test our Iterative Parameter Tuning Method only on one vessel in one container terminal in one data set. The vessel, the container terminal and the data set are all things that can influence the performance of our Iterative Parameter Tuning Method. It is possible that the performance of the Iterative Parameter Tuning Method decreases when we change one of those things. For example, if we would have to stow more containers to a vessel, this would add extra computational complexity to the heuristics. This could possibly lead to less optimal parameter tuning advice by the Iterative Parameter Tuning Method. For the results that we use we are also dependent on the performance of the emulations. The QC productivities for the emulations can differ more than a value of one container per hour per QC if we run an emulation for one stowage plan multiple times. It is possible that we would get different results if we would run all the emulations again.

6 Conclusion

The goal of this thesis is to investigate whether it is possible to automate the parameter tuning process of the stowage planner algorithm.

We present our Iterative Parameter Tuning Method to tune the stowage planner parameters. After an initialization phase that we use to estimate the effects of the stowage planner with a linear regression, the Iterative Parameter Tuning Method iteratively generates a parameter suggestion using our Parameter Suggestion Model and updates the linear regression. We get our final parameter tuning advice when the parameter suggestions from the Parameter Suggestion Model converge. To evaluate the performance of a parameter suggestion, we run an emulation with the stowage plan that the stowage planner generates for that parameter suggestion to estimate the QC productivity.

For our Parameter Suggestion model, we propose four different heuristics as options to model the stowage planner. These four heuristics are the Simple Local Search, the GRASP, the Simulated Annealing, and the LNS heuristic. We find that for all the four different heuristics the Iterative Parameter Tuning Method generates a final parameter suggestion that reaches a QC productivity higher than the benchmark we compare our parameter suggestion to. Especially when we use the GRASP heuristic, we get a final parameter suggestion that reaches a high QC productivity.

There is one problem with the Iterative Parameter Tuning Method and that is the computation time of tens of days. This limits the possibilities of the Iterative Parameter Tuning Method to tune the parameters of the stowage planner for each vessel in real-time. However, even with the long computation time the Iterative Parameter Tuning Method is still useful for more long term parameter optimization projects that aim to find the best stowage planner parameter settings in the long term.

We conclude that our results indicate that it is possible to automate the tuning process of the parameters from the stowage planner using our Iterative Parameter Tuning Method. Each step of the Iterative Parameter Tuning Method can be completely automated and it does not need any additional inputs after initializing the method. The Iterative Parameter Tuning Method also gives good working final tuning results, as the final parameter advice of the method reaches a high QC productivity.

There are still numerous future research possibilities on automating the tuning process. First it would be interesting to investigate whether it is possible to decrease the total computation time of the Iterative Parameter Tuning Method, as that is the biggest problem with the current implementation. It would also be interesting to investigate whether the Iterative Parameter Tuning Method would also perform the same for different vessels or container terminals as we only look at one vessel in one container terminal in this thesis. Another future research direction is to investigate whether it is possible to apply the Iterative Parameter Tuning Method to the parameter tuning of other algorithms such as the vehicle dispatching algorithm.

References

AKQUINET group. (2023). *Container optimization with chesscon software.* Retrieved from `https://www.chesscon.com/` (Accessed on May 12, 2023)

Alarie, S., Audet, C., Gheribi, A. E., Kokkolaras, M., & Le Digabel, S. (2021). Two decades of blackbox optimization applications. *EURO Journal on Computational Optimization*, *9*, 100011.

Audet, C. (2014). *A survey on direct search methods for blackbox optimization and their applications*. Springer.

Audet, C., & Dennis Jr, J. E. (2006). Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on optimization*, *17*(1), 188–217.

Boer, C., & Saanen, Y. (2012). Testing, tuning and training terminal operating systems: a modern approach. In *International conference on logistics and maritime systems (logms)* (pp. 25–35).

Boer, C. A., & Saanen, Y. (2008). Controls: Emulation to improve the performance of container terminals. In *2008 winter simulation conference* (pp. 2639–2647).

Conn, A. R., & Le Digabel, S. (2013). Use of quadratic models with mesh-adaptive direct search for constrained black box optimization. *Optimization Methods and Software*, *28*(1), 139–158.

Feo, T. A., & Resende, M. G. (1995). Greedy randomized adaptive search procedures. *Journal of global optimization*, *6*, 109–133.

Heij, C., de Boer, P., Franses, P. H., Kloek, T., van Dijk, H. K., et al. (2004). *Econometric methods with applications in business and economics*. Oxford University Press.

Jonker, T., Duinkerken, M., Yorke-Smith, N., de Waal, A., & Negenborn, R. (2021). Coordinated optimization of equipment operations in a container terminal. *Flexible Services and Manufacturing Journal*, *33*, 281–311.

Kirkpatrick, S., Gelatt Jr, C. D., & Vecchi, M. P. (1983). Optimization by simulated annealing. *science*, *220*(4598), 671–680.

Korach, A., Brouer, B. D., & Jensen, R. M. (2020). Matheuristics for slot planning of container vessel bays. *European Journal of Operational Research*, *282*(3), 873–885.

Larsen, R., & Pacino, D. (2021). A heuristic and a benchmark for the stowage planning problem. *Maritime Economics & Logistics*, *23*, 94–122.

McGregor, I. (2002). The relationship between simulation and emulation. In *Proceedings of the winter simulation conference* (Vol. 2, pp. 1683–1688).

Monaco, M. F., Sammarra, M., & Sorrentino, G. (2014). The terminal-oriented ship stowage planning problem. *European Journal of Operational Research*, *239*(1), 256–265.

Pacino, D. (2012). *Fast generation of container vessel stowage plans: using mixed integer programming for optimal master planning and constraint based local search for slot planning.* IT-Universitetet i København.

Papalexopoulos, T. P., Tjandraatmadja, C., Anderson, R., Vielma, J. P., & Belanger, D. (2022). Constrained discrete black-box optimization using mixed-integer programming. In *International conference on machine learning* (pp. 17295–17322).

Parreño, F., Pacino, D., & Alvarez-Valdes, R. (2016). A grasp algorithm for the container stowage slot planning problem. *Transportation Research Part E: Logistics and Transportation Review*, *94*, 141–157.

Rios, L. M., & Sahinidis, N. V. (2013). Derivative-free optimization: a review of algorithms and comparison of software implementations. *Journal of Global Optimization*, *56*, 1247–1293.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*.

Schrimpf, G., Schneider, J., Stamm-Wilbrandt, H., & Dueck, G. (2000). Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, *159*(2), 139–171.

Torczon, V. (1997). On the convergence of pattern search algorithms. *SIAM Journal on optimization*, *7*(1), 1–25.

Vu, K. K., d'Ambrosio, C., Hamadi, Y., & Liberti, L. (2017). Surrogate-based methods for black-box optimization. *International Transactions in Operational Research*, *24*(3), 393–424.

Zhu, M., & Bemporad, A. (2023). Global and preference-based optimization with mixed variables using piecewise affine surrogates. *arXiv preprint arXiv:2302.04686*.

Appendices

Appendix A Abbreviations

Table 28: Overview of the abbreviations used in this thesis

Abbreviation	Full description
AGV	Automated Guided Vehicle
ALNS	Adaptive Large Neighborhood Search
ASC	Automated Stacking Crane
GPS	Generalized Pattern Search
GRASP	Greedy Randomized Adaptive Search Procedure
LNS	Large Neighborhood Search
MADS	Mesh Adaptive Direct-Search
MIP	Mixed Integer Programming
OLS	Ordinary Least Squares
QC	Quay Crane
RMG	Rail Mounted Gantry Crane
SC	Straddle Carrier / Shuttle Carrier
TOS	Terminal Operating System

Appendix B Results from the settings selection for heuristics

Table 29: Full results for the Simulated Annealing heuristic for different settings

Temperature	Cooling Factor	OWDPE	WEIPE	Total weight penalty	CCSPE	QCSPE	YASPE	Total penalty	Comput time (s)
5	0.9	80	0	80	27528	13160	39900	80668	0
10	0.9	80	0	80	27516	13160	39900	80656	0
50	0.9	260	0	260	26796	13020	39700	79776	0.008
100	0.9	280	0	280	26496	13020	39600	79396	0.018
500	0.9	420	501	921	26046	13020	38600	78587	0
1000	0.9	420	501	921	26226	13020	38800	78967	0
5000	0.9	580	501	1081	26322	12880	38900	79183	0.01
10000	0.9	760	501	1261	26454	12740	39000	79455	0.002
5	0.99	460	0	460	23910	12810	36100	73280	0
10	0.99	580	0	580	23946	12740	34700	71966	0.009
50	0.99	720	0	720	21540	12110	32900	67270	0.01
100	0.99	1200	0	1200	20208	12110	32400	65918	0.01
500	0.99	2680	1002	3682	18486	11550	32900	66618	0.008
1000	0.99	2980	1002	3982	17430	11340	32300	65052	0.01
5000	0.99	3280	4008	7288	18588	11900	31000	68776	0
10000	0.99	4460	4008	8468	18606	11690	31700	70464	0.01
5	0.999	1920	0	1920	12972	10850	21200	46942	0.02
10	0.999	1880	0	1880	13020	10150	19500	44550	0.02
50	0.999	2540	0	2540	8850	9730	17600	38720	0.02
100	0.999	2720	0	2720	9846	8750	15200	36516	0.02
500	0.999	3540	0	3540	8628	9800	13300	35268	0.03
1000	0.999	4420	0	4420	8436	10220	15800	38876	0.028
5000	0.999	4580	0	4580	6918	9590	12800	33888	0.04
10000	0.999	4460	0	4460	8802	9800	13400	36462	0.044
5	0.9999	2100	0	2100	4842	7910	8200	23052	0.053

Table 29 continued from previous page

Tempe-rature	Cooling Factor	OWDPE	WEIPE	Total weight penalty	CCSPE	QCSPE	YASPE	Total penalty	Comput time (s)
10	0.9999	2240	0	2240	3990	7070	7900	21200	0.069
50	0.9999	2540	0	2540	3234	6790	5100	17664	0.099
100	0.9999	2420	0	2420	3168	7140	3700	16428	0.101
500	0.9999	2520	0	2520	3558	7210	4400	17688	0.135
1000	0.9999	3120	0	3120	3228	6930	4000	17278	0.157
5000	0.9999	2720	0	2720	2796	7000	4800	17316	0.192
10000	0.9999	3280	0	3280	3108	7000	4400	17788	0.2
5	0.99999	1640	0	1640	2082	6090	4400	14212	0.303
10	0.99999	1840	0	1840	1698	5180	3600	12318	0.349
50	0.99999	1880	0	1880	1764	4760	2800	11294	0.571
100	0.99999	2000	0	2000	1830	4550	2400	10780	0.663
500	0.99999	1900	0	1900	1788	4480	2500	10668	0.85
1000	0.99999	2160	0	2160	1674	4480	2700	11014	0.879
5000	0.99999	2220	0	2220	1902	4270	2200	10592	1.098
10000	0.99999	2040	0	2040	2052	4690	2000	10782	1.192
5	0.999999	1700	0	1700	1770	4200	2900	10470	1.916
10	0.999999	1480	0	1480	1758	4130	3000	10368	2.548
50	0.999999	1920	0	1920	1650	3850	1800	9220	4.422
100	0.999999	1840	0	1840	1488	3710	2100	9138	5.019
500	0.999999	1840	0	1840	1662	3430	1800	8732	7.078
1000	0.999999	1800	0	1800	1614	3710	1800	8924	7.849
5000	0.999999	1880	0	1880	1722	3500	1900	9002	9.778
10000	0.999999	1940	0	1940	1590	3780	2000	9310	11.822
5	0.9999999	1700	0	1700	1674	3290	2500	9164	23.189
10	0.9999999	1440	0	1440	1728	3570	2000	8738	23.964
50	0.9999999	1680	0	1680	1758	3220	1600	8258	40.887
100	0.9999999	1840	0	1840	1590	3080	1700	8210	48.632
500	0.9999999	1660	0	1660	1698	3220	1700	8278	69.864
1000	0.9999999	1820	0	1820	1710	3010	1900	8440	75.501
5000	0.9999999	1700	0	1700	1614	2870	2100	8284	94.192
10000	0.9999999	1740	0	1740	1566	3290	1600	8196	106.186
5	0.99999999	1400	0	1400	1596	1855	1700	6551	196.568
10	0.99999999	1420	0	1420	1554	1785	1600	6359	319.982
50	0.99999999	1580	0	1580	1554	1785	1500	6419	523.762
100	0.99999999	1640	0	1640	1542	1715	1500	6397	520.166
500	0.99999999	1540	0	1540	1542	1785	1500	6367	794.465
1000	0.99999999	1580	0	1580	1500	1820	1500	6400	1008.241
5000	0.99999999	1540	0	1540	1584	1750	1500	6374	1194.31
10000	0.99999999	1500	0	1500	1566	1820	1500	6386	1321.462

Table 30: Full results for the LNS heuristic for different settings

Number of iterations	Number of containers	OWDPE	WEIPE	Total weight penalty	CCSPE	QCSPE	YASPE	Total penalty	Computation time (s)
100	5	1120	0	1120	18750	6230	28800	54900	6.679
100	10	1900	0	1900	10218	5460	18200	35778	10.200
100	15	1300	0	1300	6906	4760	11600	24566	16.809
100	20	1640	0	1640	5334	4095	6400	17469	69.033
100	25	1580	0	1580	3888	3780	5400	14648	46.338
100	30	1820	0	1820	3654	3850	6200	15524	113.553
200	5	1500	0	1500	12960	5950	23600	44010	11.772
200	10	2220	0	2220	6564	5075	10700	24559	16.025
200	15	1480	0	1480	4902	4165	7500	18047	40.232
200	20	1680	0	1680	3444	3745	4300	13169	190.735
200	25	1820	0	1820	2496	3360	3400	11076	96.821
200	30	1880	0	1880	2646	3360	2900	10786	176.469
300	5	1880	0	1880	10014	6090	19500	37484	16.863
300	10	2020	0	2020	5388	4830	8700	20938	23.386
300	15	1680	0	1680	3384	3850	4800	13714	47.303
300	20	1800	0	1800	2844	3325	3500	11469	221.507
300	25	1740	0	1740	2610	3255	2600	10205	173.606
300	30	1760	0	1760	2514	3115	2600	9989	219.280
500	5	1900	0	1900	7830	5705	15300	30735	20.905
500	10	2060	0	2060	4332	4305	5600	16297	34.218
500	15	1840	0	1840	2682	3325	3800	11647	89.491
500	20	1840	0	1840	2082	3255	2900	10077	255.297
500	25	1740	0	1740	2190	2905	2200	9035	234.235
500	30	1760	0	1760	1932	2905	2600	9197	316.172

Appendix C Penalty values from stowage planner solutions

Table 31: Penalty values from the stowage planner solutions for the large set of parameter sets

Parameter values				Total	Total weight	Penalty				
CCSPA	CLDPA	QCSPA	YASPA	penalty	penalty	OWDPE	WEIPE	CCSPE	QCSPE	YASPE
0	0	0	0	543	543	42	501	0	0	0
0	0	0	200	25649	1849	346	1503	0	0	23800
0	0	140	0	6711	551	50	501	0	6160	0
0	0	140	200	20160	2470	466	2004	0	6090	11600
12	8	0	0	3595	535	34	501	3060	0	0
12	8	0	200	29356	1516	514	1002	3840	0	24000
12	8	140	0	9235	719	218	501	2496	6020	0
12	8	140	200	43886	2624	620	2004	2772	6090	32400
6	8	0	0	1264	46	46	0	1218	0	0
6	8	140	200	46919	3071	566	2505	1758	6090	36000
6	4	0	0	621	543	42	501	78	0	0
6	4	140	200	49383	1849	346	1503	234	6300	41000
12	4	0	0	723	543	42	501	180	0	0
12	4	140	200	45875	1865	362	1503	720	6090	37200
6	8	70	100	23033	1897	394	1503	1566	2870	16700

Table 32: Penalty values from the stowage planner for the Simple Local Search heuristic

Iteration	Parameter values				Total	Total weight	Penalty				
	CCSPE	CLDPE	QCSPE	YASPE	penalty	penalty	OWDPE	WEIPE	CCSPE	QCSPE	YASPE
1-3	6	5	0	200	12186	1356	354	1002	1230	0	9600

Table 33: Penalty values from the stowage planner for the GRASP heuristic

Iteration	Parameter values				Total penalty	Total weight penalty	Penalty				
	CCSPA	CLDPA	QCSPA	YASPA			OWDPE	WEIPE	CCSPE	QCSPE	YASPE
1-3	9	7	0	200	14095	366	366	0	4329	0	9400

Table 34: Penalty values from the stowage planner for the Simulated Annealing heuristic

Iteration	Parameter values				Total penalty	Total weight penalty	Penalty				
	CCSPA	CLDPA	QCSPA	YASPA			OWDPE	WEIPE	CCSPE	QCSPE	YASPE
1-3	10	8	0	200	18187	1877	374	1503	6910	0	9400

Table 35: Penalty values from the stowage planner for the LNS heuristic

Iteration	Parameter values				Total penalty	Total weight penalty	Penalty				
	CCSPA	CLDPA	QCSPA	YASPA			OWDPE	WEIPE	CCSPE	QCSPE	YASPE
1	6	5	0	140	8910	1386	384	1002	804	0	6720
2-3	8	4	0	190	11507	1849	346	1503	728	0	8930
4	1	8	0	200	12100	1364	362	1002	1136	0	9600
5-7	12	8	0	0	3595	535	34	501	3060	0	0

Table 36: Penalty values from the stowage planner with a shorter initialization phase

Iteration	Parameter values				Total penalty	Total weight penalty	Penalty				
	CCSPA	CLDPA	QCSPA	YASPA			OWDPE	WEIPE	CCSPE	QCSPE	YASPE
1	12	2	140	200	53945	1849	346	1503	456	38640	13000
2-3	1	8	0	200	12100	1364	362	1002	1136	0	9600
4	10	8	0	200	18187	1877	374	1503	6910	0	9400