Strategies for optimal (re)placements of containers at a container terminal

A search for good strategies for container placement and retrieval by minimizing the number of reshuffles needed.

Johan Streng
Student number 332392
27th June 2012

Supervisor: Prof. Dr. Ir. R. Dekker
Second reader: Dr. A.F. Gabor
Econometrics and Operational Research
Erasmus University Rotterdam

Abstract

This thesis is about strategies to minimize the number of moves stacking cranes need to make to stack and retrieve containers of a container terminal. Several heuristics are compared to each other in different steps. First containers are just removed, then they are both placed and removed while all containers arrive before the first container leaves the container terminal and finally the containers arrive at- and leave the container terminal at the same time.

Table of Contents

1 Introduction

Containers are an essential part in transportation of goods these days. More and more goods are transported by containers each and every year. These containers are transshipped several times during their trip from production- to consuming market.

This transshipment can happen immediately, but most of the time the container has to stay on a stack for a while. For the stacking of container space is needed. However, the amount of containers grows much more rapidly than the space available for the stacking of containers. Therefore more and more efficient container handling systems are needed. To be efficient automated container handling and transportation technology becomes more and more important as taught by Günther and Kim (2006), [1].

Containers are stacked at a so-called container terminal. The terminal basically exists of a set of container blocks A container block can be seen in the picture at the right. Each block also has three dimensions: the stack, the lane and the tier.
In this picture, there are three stacks, seven lanes and 4 tiers. The amount of stacks and lanes determine the amount of space needed on the container terminal, where the amount of tiers give the height of a container block. This number of tiers can never be higher than the maximum number of tiers. Otherwise rail mounted gantry cranes (AMGs) cannot move newly arrived containers over these stacked containers.

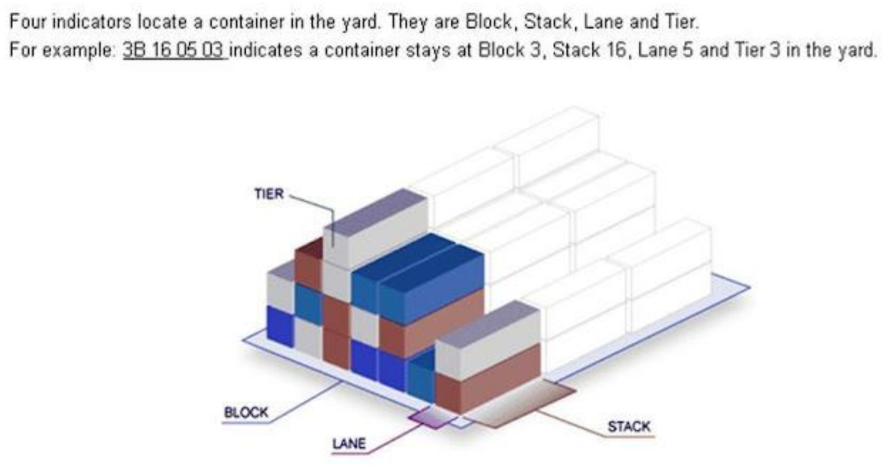

Four indicators locate a container in the yard. They are Block, Stack, Lane and Tier.
For example: 3B 16 05 03 indicates a container stays at Block 3, Stack 16, Lane 5 and Tier 3 in the yard.

fig 1. A container block. Source: [9]

1.1 Container handling

Once a container enters the port, it is unloaded and taken to the container yard. This is, most of the time, done by automated guided vehicles (AGVs). Every port can have a lot of AGVs, but once the container reaches the place containers are stacked, a rail mounted gantry crane does the actual placing.

Unlike the vehicles, these RMGs are bound to a certain block and containers can only be stacked at a certain block by one or maybe two RMGs. These RMGs can handle one container at a time and they are only capable of moving a container that is at the top of a stack. This means that they have to be used efficiently.

Stacking containers on top of each other can be very efficient. Because of the stacking, less space is needed and the RMGs do not need to move as much as when all containers were placed on the ground. This can save huge amounts of time and space, which both can be converted into money.

However, stacking containers also has disadvantages. If a container is not at the top of a stack, but it has to be removed, other containers need to be replaced. This causes extra movement of the RMGs and time-loss. If too much containers need to be replaced, the amount of money won due to less space cannot make up for the time lost due to reshuffling. For reshuffling, see example 1. Reshuffling and space are essential parts of the problem: how to optimally place and, if necessarily, reshuffle containers?

Example 1. Reshuffling containers

Here, container 3 needs to be replaced in order to remove container 1.

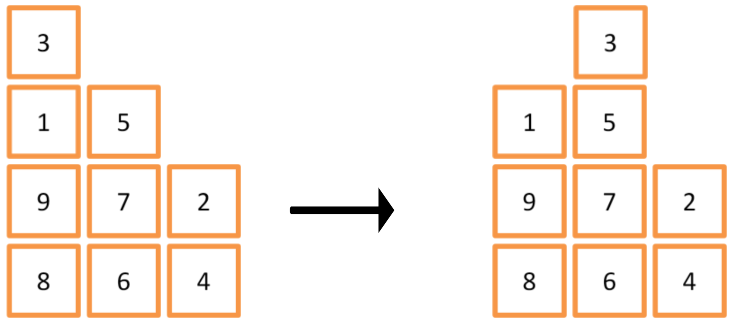


Fig 2. Movement to get container 1

If container number 1 had been on top of container 3, it would have been better

2 Problem description

The problem in this thesis is where to put newly arrived containers and where to place containers that need to be reshuffled, so that the total amount of container moves is minimal. This problem can be split into two sub-problems: the stacking of newly arrived containers and the removal of containers (where reshuffles are needed).

2.1 Stacking containers

Because of the ever larger ships that only visit a few ports and unload a lot of containers at one port, container terminals do have to handle a lot of containers very quickly. The containers have to be stacked at the quay, but due to limited space containers have to be stacked on top of each other.

Because of this, containers need to be stacked as good as possible. It can save huge amounts of time, and thus money, if a container that has to leave the terminal can be removed immediately instead of first reshuffling other containers on top of it.

Also, one has to take into account there are a lot of types of containers (20ft, 40ft, reefers, etc.). However, different kinds of containers are stacked at a different part of the container terminal and in this thesis we do not take it into account. Only one part of the terminal where only one type of container is handled will be looked at.

Stacking and retrieving containers always happens in blocks. However, for simulation and heuristics it is much easier to have the entire stack of containers at a line, instead of in a block. There is not much difference between these two: a block can easily be transferred to a line by placing the stacks next to each other. Looking at figure 1, in this thesis we got one stack, multiple lanes and multiple tiers.

2.1.1 Perfect information

To stack the containers in the best possible way, information about all containers that have arrived and still have to arrive is required. The best way to stack the container arriving with perfect information about arrival and departure times is investigated by Borgman, van Asperen and Dekker (2010), [2].

2.1.2 Online information

Unfortunately, most of the time it is not possible to know the arrival and departure times of containers that still have to arrive.
The departure of containers on the terminal is also not exactly known. However, due to time slots for carriers and trucks a lot of this uncertainness is taken away. Missing information means during stacking it is only possible to take the already arrived containers into account.

It therefore might be unavoidable that containers are stacked on top of other containers, which have to be removed earlier. This means that, time consuming, reshuffles have to be made and the efficiency level drops. This so called “online” information is most researched the past years. In examples by Borgman, van Asperen, and Dekker, (2010), [3] , Borgman (2009), [4] and Froyland, Koch, Megow, Duane, and Wren (2008), [5] .

2.1.3 Statistic information

When the information about the not jet arrived containers is not available, it might still be predicted. This can be done by statistical analysis of the arrival of containers in the past. This might predict when new containers will arrive and when they will leave again. Hartmann (2004), [6] made an algorithm to stack the containers using statistics to find the optimal way of stacking containers.

2.2 Removing containers

Not only can the stacking of containers, but also the retrieval happen in many ways. The ideal situation would be when all containers are stacked in the right order. This means there is not a single container stacked on top of another container that has to leave earlier. That way, not a single reshuffle is needed.

But it is not possible to have not a single container stacked “wrong” (so with an earlier-leaving container beneath it) in an entire container terminal, where thousands of containers are stacked. It would help if there are containers that leave on the same ship, so that their departure time is exactly the same. However, in this thesis I will only talk about containers being removed one by one in a specific order. One can assume in this part of the terminal are only containers meant to be loaded on trucks.

The minimization of the number of needed reshuffles can be done by different kinds of algorithms. For example, one can use the Beam Search Algorithm by Wu and Ting, [7]. Other examples are the branch and bound algorithm and decision rule used by Kim and Hong (2006), [8].

3 Plan

3.1 Literature

We start this thesis with a section about the literature used. What is already done in this literature and what conclusions can be drawn from it? By doing so, we try to learn a couple of things about container terminals. For example what they look like and how does the stacking of containers work?

3.2 Retrieving containers

Secondly a simple exercise is made for retrieving of containers once they are stacked. This is done both by hand and by the use of some (simple) heuristics, explained later in this thesis. The retrieval of containers is done only for a quay of 1 stack, 5 lanes and 5 tiers.

3.3 Stacking containers

But this thesis is not only about the retrieval of containers from an already existing stack in an optimal way, it is also about the optimal stacking of containers arriving at the terminal. This is the third, and last, part of this thesis. It is split into two different sub parts.

3.3.1 All container arrive before first leaves

For the first sub part, we are going to create a pile of containers. All the containers arrive before the first container leaves the place. Here, the containers that depart the quickest are placed on top as much as possible, the ones directly below that leave after these and so on. This is done at a bay where the amount of spaces is fixed. This pile is made with different amounts of containers. The expected amount of reshuffles is then calculated. This is done for different kinds of quays. We use the standard quay; 1 stack, 5 lanes and 5 tiers. Then we increase the size of the quay; 1 stack, 6-8 lanes and 6-8 tiers. Afterwards the shape of the quay is changed; 1 stack, 10 or 20 lanes and 5 tiers. Finally, a larger quay is created; 1 stack, 120 lanes and 6 tiers.

3.3.2 Online information

The second sub part is about the “online” information of arrivals of containers. In this part several heuristics are tested and compared to each other. This by using simulation to determine the arrivals of containers and the time until the containers leave the terminal again.
Afterwards, we draw conclusions for the algorithms and see how the heuristics hold against each other.

4 Methodology

4.1 Retrieving containers

Several persons already constructed algorithms for the retrieval of containers. One is the beam search algorithm of Wu and Ting [7], the other is made by Kim and Hong [8]. However, in this thesis only a few simple heuristics (explained in chapter 7) are used to retrieve the containers. Furthermore the retrieval is done by hand for one particular stack. This can be done because small retrieval problems are used.

4.2 Stacking containers

Of course, the yard cannot be filled entirely. If done so, it is impossible to remove a container that is not at the top of a stack. Therefore a container terminal can only be filled for α percent of the possible number of containers. If the terminal is filled more than α percent, new containers are automatically lost. These containers will leave on the same ship that sailed them into the harbor. Because there is no data, the arrival of containers is created by simulation. This is the time when a new container needs to be stocked. This container is therefore already unloaded from the ship and put on an AGV. The amount of time this container stays at the stack is determined when it arrives.

4.2.1 All container arrive before first leaves

Every container gets a random integer smaller than or equal to the amount of containers. Each integer can only be given once, because the containers leave one by one in order of their number. It does not matter when exactly the containers arrive, only in which order.

4.2.2 Online information

Although not correct in reality, the time a container spends at the terminal is considered to be exponentially or randomly distributed. The containers arrive one at a time. The inter arrival time of the containers is said to be constant. Therefore, every container arrives exactly one time unit after the previous one. Containers that cannot be placed are not put into a queue, but these are lost containers.

5 Data, case generation and implementation

5.1 Data

For this thesis data is generated, not obtained from earlier research. This data is created by using the rand function of the mathematical program Matlab. To make it possible to repeat the research, a fixed seed is used. This seed sometimes is rand('state',str2double('12345')) and sometimes rand('state',str2double('100')). Which seed is used is always stated before the actual experiment.

5.2 Case generation

The arrival times of the containers are not important for our study purpose. The important thing is the order in which the containers arrive at the container terminal. We therefore set the arrival times of the containers on 0,1,2 etc. The departure times of these arriving containers can of course vary, but unfortunately we do not know the distribution of the leaving containers. We therefore use different kinds of discrete distributions:
- the storage time of the containers is uniform distributed on time-interval [0,2]
- the storage time of the containers is exponentially distributed with mean 1

Of course, only for the online information we would like to know the precise time the containers depart. When removed after everything is stacked and just removed, it is only important to know the order of departure as well. Therefore the containers can get the numbers 1, 2, 3, etc.

Every arrival or departure of a container is a time at which actions happen. This means that reshuffles happen at the same time a container arrives of leaves. Every reshuffle that is done to make the departure or arrival possible happens when that container arrives/departs.

5.3 implementation

The simulation, as well as the heuristics to get the number of moves per case, is done in a mathematical program called Matlab. As said in the Data part, for all different kinds of heuristics the same seed is used, so that one is able to compare the outcome of the different heuristics. A Matlab-code of these heuristics is in table A of the appendix. This heuristic is the heuristic described in section 8.2.2.

6 Literature

According to Günther and Kim (2006), [1], the primary objective for a container terminal is to be an interface between two different types of transportations. At least one of these transportations is most of the time done by ship. Due to the increase of container transport, container handling has to become more and more efficient.

Borgman, van Asperen and Dekker (2010), [2] & [3], investigated the quality of some existing algorithms for the placement of containers. They found out that results obtained from smaller stacking areas are also valid for larger stacking areas. Also they showed that, if containers arrive and leave at the same side of the terminal (landside or quayside), models that take the moving distance into account perform better than models that do not. This is not important in this thesis, because we assume containers arrive by ship and leave by truck. Therefore every container has to cross the entire block eventually.

Borgman (2009), [4], also used some algorithms for container stacking. He found out that random stacking and leveling are not very good strategies. The TVR algorithm (Travelling time Versus Reshuffles) performs much better than these two when we do not know information about containers. This algorithm however is not used, because we do not care about the distances (since every container has to cross the entire block). Also, when we have some information about the departure times of the containers it is better to use a LDT (Leveling with Departure Times) algorithm, which does not make use of distances the containers cover.

Froyland, Koch, Megow, Duane and Wren (2008), [5], used an integer programming-based heuristic for the import and export of container at the land-side of a block. Although it performs well, it is not used in this thesis. This because the heuristic is not trying to minimize the total number of moves (or reshuffles), but the amount of resources used (AGVs & RMGs).

Hartmann (2004), [6], did not try to find an optimal way to remove containers. Instead, he tried to make a model that generates real-life scenarios at a container terminal. These scenarios can be: the arrival of a deep-sea vessel, a truck or train etc.

Wu and Ting, [7], used a beam search algorithm for the minimization of reshuffling containers at a container terminal. This beam search looks like a breadth-first branch-and-bound scheme, but it keeps only a few nodes in every step. The algorithm tells which nodes to keep. We do not use this algorithm in this thesis.

Kim and Hong (2006), [8], also focused on the removal of containers and an efficient way to reshuffle. The both used a branch-and-bound and a heuristic based rule to find an optimal solution for the stacking problem. They also proposed a procedure to estimate the expected amount of reshuffles. These branch-and-bound and heuristic rule are also not used in this thesis.

7 Optimal removal of containers

7.1 Problem

In the first part of this thesis, we focus on the removal of containers of a container yard. No new containers will arrive and all the existing containers will be removed. This removal is done both by hand and by the use of simple heuristics.

The container-removal problem looks as follows:

Example 2: container 1 has to be removed first etc. The empty blocks are spaces not used.

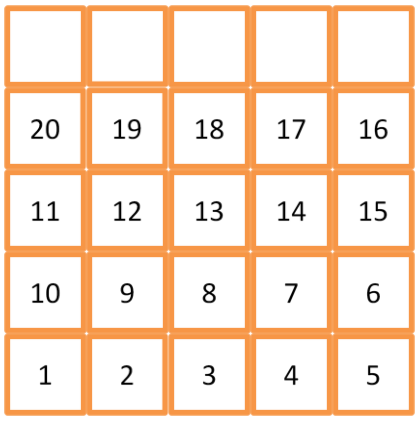


Fig 3. Container block for problem.

As one can see, it is not possible to simply remove the containers one by one. Some reshuffles have to be done as well.

7.1.1 Upper and Lower bound

An upper and lower bound for this problem can be found easily. To start with the lower bound; one can easily see that a lower bound of the number of moves necessarily is 38. This number includes removal moves (20) and reshuffle moves (18). The lower bound of the number of reshuffles is calculated by the number of containers on top of an earlier-leaving container (15) plus the second time the three containers on top of container 1 have to be replaced (3). The first time, namely, they have to be placed on top of another stack with a container that has to leave earlier.

Usually, it is not that simple to create a lower bound like this. An easy (but worse) alternative is the number of containers stacked on top of a container that has to leave earlier.

An upper bound can be created by always replacing the containers as far to the left as possible. This way, an upper limit of 45 is obtained. This upper limit contains removals (20) and reshuffles (25).

7.1.2 Creation of quay

The quay above is not the only way twenty containers can be stacked in 5 lanes with tier 4. The retrieving of containers is done by creating a stack containing twenty containers. These containers are placed in five lanes of four high. The places of the particular containers are chosen randomly, but for all heuristics, the same seed (seed: str2double('12345')) is used. Although it is possible to place the containers in 20! (=2.43*10^18) different ways, in our research the placement is only done 10 times 100 times (so in total 1,000 times) and 10 times 1,000 times (so 10,000 times). Three different aspect are registered: the amount of moves made to clear every one of the 1,000 or 10,000 stacks, the mean of these moves used to clear the stacks and the standard deviation of this mean.

7.2 Method

7.2.1 Heuristic one: always left heuristic

This heuristic is a very simple one. If a container that needs to be removed is on top of a stack, it is simply removed, just like all other heuristics. If another container is on top of it, however, this container is reshuffled in a very simple way. This reshuffle container is placed as far to the left as possible. This means it is placed on top of the containers in the lane that is the most left, not maximally filled lane.

7.2.1.1 Quality of heuristic one

In section 7.1.1 this heuristic is referred to as an upper limit of the solution. This can be explained, by the fact that if place left is optimal, one should place the reshuffle container left. If it is not optimal, one should place the container somewhere else. This heuristic is expected to be worse than all the other heuristics.

7.2.2 Heuristic two: based on entire piles

If the problem above is solved by hand, one of course uses some (or in this case all) of the same assumptions this second heuristic is based on. In the figure on page 10 (fig. 4), the steps for removing a container using heuristic one are given. Step 1 is to check whether the container that has to be removed is on top of a lane. If that is true, the container is simply removed. If not, the top container of the lane in which the removal container is stacked will be reshuffled. This container will from now on be called the reshuffle container.

Once determined the reshuffle container has to be reshuffled, various things are checked. The reshuffle container has a departure time. Based on this time, as well as the time of the other containers, decisions are made. The time of the reshuffle container is compared to the times of other containers, except of course of the containers in the same lane as the reshuffle container.

The first check, so step number 2, is whether there exists at least one lane for which all containers have a later departure time than the reshuffle container. If so, check whether at least one of these lanes is not filled maximally, step 3. When maximally filled, it is impossible to put another container on top of it. When at least one of these stacks is not maximally filled, the reshuffle container is stacked on the, not maximally filled, lane with the container that leaves the quickest after the reshuffle container on top.

If there are no lanes for which all containers have a later departure time than the reshuffle container, or these lanes are all maximally filled, we arrive at step number 4. Here, the next check is made: is there an empty place (so, a lane of zero containers). If so, the reshuffle container is placed on the floor. If not, a next check is made.

Since this check has to be made, we know the reshuffle container has to be placed on a lane with a container that has to leave earlier. This container therefore will have to be removed at least one other time. The check made now is step number 5 in the heuristic.
Now we check whether there exists at least one lane for which the top container has to leave before the reshuffle container. If that is the case, it is better to place the reshuffle container over there, because when reshuffling is necessarily to get a container in that lane, the reshuffle container can be placed with the top container directly on top of it.
Of course, if there exists at least one such lane, one has to check whether this/these stack(s) are filled maximally or not. If one or more such lane exists, the container is placed on top of the container that leaves the latest, so again the closest to the reshuffle container.

When there is no lane for which the top container leaves before the reshuffle container, the reshuffle container is placed on the lane for which the top container leaves the soonest after it. This is step 5.2. This is not per se the container that leaves the soonest after it, but it is the container that leaves the soonest after it on a lane that is not filled maximally. Because this is the last possibility, there is always a possible place to stack the reshuffling container.

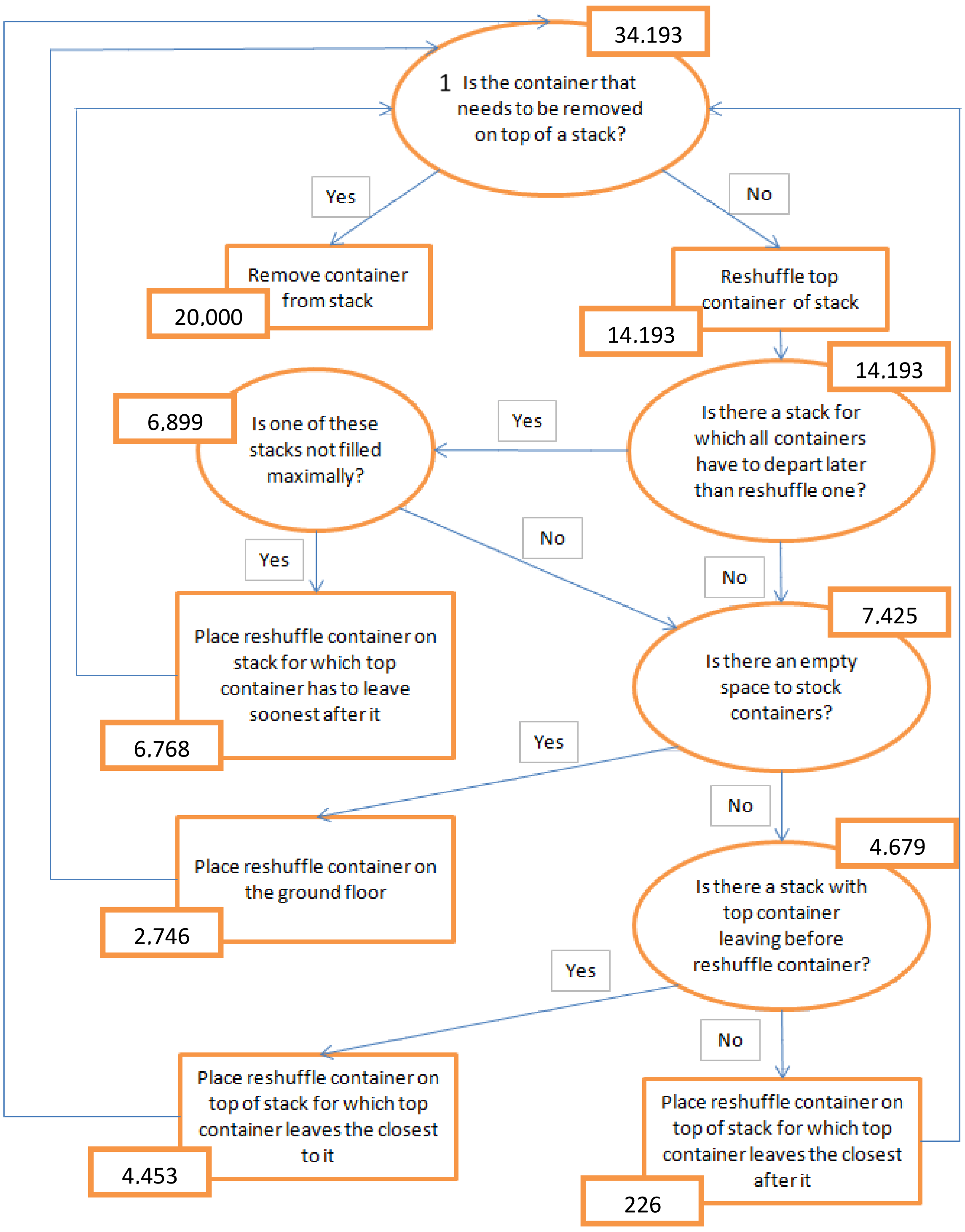

34.193
1 Is the container that needs to be removed on top of a stack?
Yes
No
Remove container from stack
20.000
Reshuffle top container of stack
14.193
14.193
Is there a stack for which all containers have to depart later than reshuffle one?
Yes
6.899
Is one of these stacks not filled maximally?
Yes
No
No
Place reshuffle container on stack for which top container has to leave soonest after it
6.768
7.425
Is there an empty space to stock containers?
Yes
No
Place reshuffle container on the ground floor
2.746
4.679
Is there a stack with top container leaving before reshuffle container?
Yes
No
Place reshuffle container on top of stack for which top container leaves the closest to it
4.453
Place reshuffle container on top of stack for which top container leaves the closest after it
226

Fig 4. Decision tree for removal of a container

7.2.2.1 Quality of heuristic two

The decision tree in figure 4 does not always have to give the best results. For instance, it might not always be optimal to place the reshuffle container on the lane for which the top container has to leave the soonest after the reshuffle container. An example for when it is, and when it is not optimal are given in example 2, on the next page of the paper. What the best way to place a container is, is different for each container. A good idea might be to give a penalty for the different inconveniences and add these penalties for every possible place. We then put the container on the spot where the penalty is minimal. The idea of a heuristic with a penalty is explained in section 7.2.4.

Example 3. Optimal placement

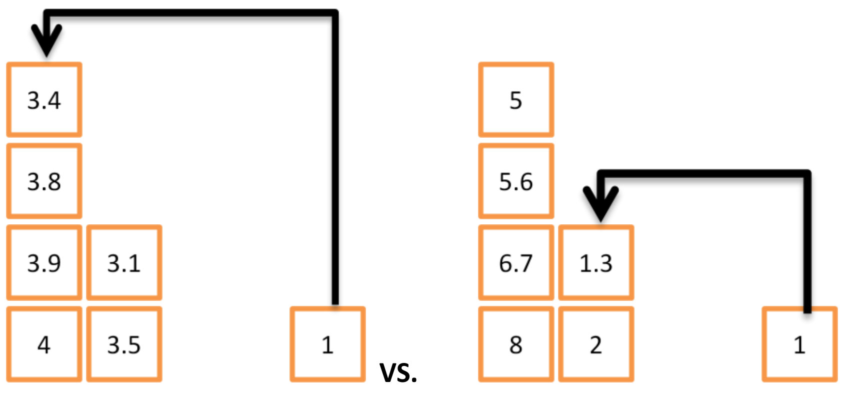


Fig 5. Better to put container on top of high stack

Fig 6. Better to put container on top of container leaves soonest after

In figure 5 it is best to place the container at the higher stack, because it then blocks only 1 available space for containers with a departure time between 1 and 3, instead of 3 free spaces. In figure 6 however, it is better to place the container at the lower stack, because the difference in departure times is only 0.3 instead of 4. So in figure 6 there is a much smaller probability that a new arrived container has to be placed on the ground.

However, it might cost too much time for the amount of times such a choice actually needs to be made for a container. It is therefore a good idea to count the number of times each stage in the decision tree is visited.

To check the quality improvement of each step in the heuristic, the containers are at first reshuffled randomly. After, the steps are added one by one, and if the container is not reshuffled after all added steps, the container is placed randomly. Then average number of moves after each added step is counted and these numbers are compared. Results are in section 7.3.

7.2.3 Heuristic three: based on top containers

This heuristic looks very much like the second heuristic. The difference is that this heuristic only takes the containers that are on top of a pile into account, instead of the entire piles. This heuristic also bears a striking resemblance to the LDT-algorithm for placing containers investigated in the papers of Borgman, van Asperen and Dekker (2010), [2] & [3]. There only is an extra in this heuristic: a reshuffle container cannot be placed at the same lane it came from.

When a container needs to be reshuffled, all other top containers are checked. When there is at least one top container that leaves later than the “reshuffle” container the reshuffle container is placed on top of the container that leaves the latest after it. Of course, only the lanes not filled maximally are considered. This step pretty much looks like step 5.2 in heuristic one. The only difference is that now we do not know if there is a container in the pile that leaves before the reshuffle container.

When it is not possible to place the reshuffle container on top of a container leaving later, the heuristic checks whether there is an empty spot. If so, the reshuffle container is placed there on the ground. This step is the same as step 4 in heuristic one.

If there is also no space available on the ground, the reshuffle container is placed on top of a container that has to leave before the reshuffle container. The reshuffle container is placed on top of the container that leaves the closest to the reshuffle container and that is stacked in a not maximally filled lane. This step looks like step 5 in heuristic one, only now we know for sure such a place is available.

7.2.3.1 Quality of heuristic three

Heuristic number two leaves out step 2 of heuristic one, and also the third step is made one time less (not after step number 2). This heuristic only compares the reshuffle container with the top container instead of the entire pile. Because done so, it is much more likely the reshuffle container is placed above a container that has to leave earlier. As a result more reshuffles are expected and also more moves in total than by the use of heuristic one. Also, the limitations for the previous heuristic hold for this heuristic as well.

Even though this heuristic seems a simpler version of the second heuristic (where departure times of entire piles are checked), it still might be better to use this one. This heuristic has potentially a much lower computation and running time than the previous one.

7.2.4 Penalty based heuristic

The last heuristic we are going to use is the penalty based heuristic. If a container needs to be reshuffled, this heuristic will consider all lanes and assign a penalty for the placing of the reshuffle in every lane. The lane that gets the minimal total penalty will be the lane in which the reshuffle container will be placed.

How are these penalties computed? First of all, a container cannot be reshuffled to the same lane it already was. Therefore, this lane will get a penalty so high, it is impossible for other lanes to get a higher penalty. The same happens for lanes that are already filled to the maximum.

The rest of the lanes will get lower penalties. Since we do not like reshuffles to be made much, we give a high penalty for placing a container on top of a container that leaves before the reshuffle container. When this is unavoidable, we would like the reshuffle container to be placed on top of a container that leaves later than this reshuffle container. Therefore, an extra penalty is given for placing the container on top of a later leaving container, once known the reshuffle container leaves later than any of the containers in that lane.

To solve the problem of heuristic one and two, which check the entire pile and top container respectively, we also give penalties for the absolute value of the difference in time the reshuffle container and the container on top of a lane leave. This penalty however is much smaller than the penalties indicated before. A last penalty is given for the number of spaces a container might block. The higher a pile is, the less spaces are blocked and the lower the penalty will be.

This penalties all have a value; these will be stated now. The lane in which a container is before it will be reshuffled, as well as the maximally filled lanes, will get a penalty of 10^10. This is quite high, but one can be sure that these lanes are not picked to reshuffle the container.

If a container is placed at a lane with at least one container that leaves earlier than this reshuffle container, a penalty of 1000 will be awarded. If the reshuffle container is also placed on top of a container that leaves after the reshuffle container, an extra penalty of 100 is given to that lane.

A penalty of 5 times the difference between the departure time of the top container and the reshuffle container is awarded to each lane, and it does not matter whether the top container leaves earlier or later (so the absolute value of the difference is used). For example: if the container that leaves first arrives and is placed on top of the container that leaves tenth, the penalty of this lane will be increased by 45 (5*10-1).

The last penalty awarded to each lane is the squared difference between the maximum stacking height minus one and the actual stacking height of that lane after the placement of the reshuffle container.

How large the penalties should be determined before. Everyone has his or her own preferences for the reshuffling and therefore can himself choose the height of the penalties. The penalties above are chosen so that the number of reshuffles was minimal (of course, compared to other penalties tried), and therefore also the total number of moves.

7.2.4.1 Quality of penalty based heuristic

This penalty based heuristic will most of the time reshuffle in the same way heuristic one (witch compared the departure time of the reshuffle container to the departure time of all containers in the lanes), but sometimes it will put a container on a higher stack so that we do not block to many spaces. Therefore this heuristic is expected to be somewhat better than heuristic one, but not much and for sure not significantly better.

7.3 Results

In the results we will do different things. First, we take a look at the heuristic shown in figure 4. We will try to figure out what happens if the steps are added one by one. After, we take a look at the results of removing the containers, when they are stacked in the way of figure 3. If done by hand, one should be able to remove all containers in 38 moves (18 reshuffles), the same amount as the lower bound, calculated in section 7.1.1 and also the different heuristics are compared to each other.

7.3.1 Heuristic step by step

Before we take a look at performance of the different kinds of heuristics, we check for heuristic two (that checks departure times of entire piles) how much every step in the heuristic contributes to the reducing of the number of reshuffles.
Below is a table in which the steps of heuristic one are compared by their number of moves. Also the reducing in moves if the steps are added one by one is shown. Of course step 3 is not taken into account. This step (check whether pile is full) is used in all other steps. One has to take into account that there were 1,000 different kinds of stacks created, all with 20 containers. Therefore 20,000

times a container is removed, while on top of a lane. The difference between step 1 and 2 is 20,000 times the removal of a container.

Steps taken.	Average No. of moves	Runtime (sec.)	Total no. times step considered
Step 1/remove if top container	53.165	0.703	34,193
Step 2/ reshuffle on pile all containers leave after	37.293	1.302	14,193
Step 3/ is pile maximally filled?	n.a.	n.a.	14,098
Step 4/ reshuffle on ground	38.666	0.790	7,425
Step 5/ reshuffle on container that leaves before	44.948	1.622	4,679
Step 5.2/reshuffle on container that leaves after	46.714	1.858	226
Steps 1,2(,3) and 4	35.232	1.230	n.a.
Steps 1,2(,3),4 and 5	34.183	1.293	n.a.
All Steps	34.193	1.271	n.a.

Table 1. The number of moves using different steps of first heuristic.

As one can see in the first column of the table all steps make sure a lower number of steps is used than when using random placement. Some steps have more effect than other. Especially the steps where the reshuffle container is placed on top of a pile only containing containers that leave after the reshuffle container and the step where the reshuffle container is placed on the ground floor if possible seem to work very well.

If we add the steps one by one we see that in general the number of moves (and reshuffles) decreases when a step is added. This only does not hold true for the last step. If we once reach step 5.2 without placing the reshuffle container, it seems better to place this container randomly on top of a container instead of on top of the first leaving one.
In the rest of the results of chapter 7, these insights are not taken into account. We only reach this step in 226 cases, when 1,000 different stacks are used. This means an average of 0.226 times this step is taken per stack. Changing the heuristic for this would perhaps give some better results, but it will cost too much time for too few gains.

If we take a further look at the number of times we reach the steps, we see that the second and third step are taken a lot of times, compared to the total number of times we enter the heuristic (34,193; the times we reach step 1). Afterwards, the number of times a step is reached is around half the times the previous step was reached. The number of times each step was reached is also placed in the figure of the heuristic in chapter 7.2.2.

7.3.2 Different Heuristics

Now, let's see if the heuristics work good or not at the stack of containers from example 2.

The first heuristic, the always left heuristic, can be considered as an upper limit, as said in section 7.1.1 and 7.2.1.1, and the number of moves is equal to 45. The other three heuristics work clearly better: number three (only look at top container) uses 41 moves and heuristic two and the penalty based heuristic only use 38 moves, the lower limit.

Now, to check whether heuristic number two, the heuristic that check entire piles, and the penalty based heuristic are not just better in this particular case, the 1,000 and 10,000 different scenarios were created. In the rest of the section, we check which heuristic is the best.
It is split into two parts. First, the mean of the number of moves made by the four heuristics and their standard deviations will be discussed. And secondly all 1,000 (not the 10,000) different amounts of moves are compared to each other in order to find out which heuristic is the best.

7.3.2.1 Mean and standard deviations

In the table below are the means and standard deviations of the four heuristics. The standard deviations are the standard deviations of 10 means over times 100 trials. The mean is the average of these 10 means. As one can see, the first heuristic, where a reshuffle container is placed left as much as possible, is much worse than the other heuristics. There is no test needed to show that this heuristic is significantly worse than the other three.

Heuristics	**Mean**	**Standard deviation**	**Runtime (sec.)**
One/ left	42.103	0.6969	0.622
Two /entire pile	34.193	0.3417	1.271
Three/top container	35.358	0.4000	1.289
Four/ penalty based	34.186	0.3223	0.771

Table 2A: The results for removing containers, using four heuristics and 1,000 cases.

However, is the second heuristic significantly better than the third one? To find an answer to this question, we take a look at the difference between the two heuristics. The difference between these two heuristics has a mean of 35.358-34.193=1.165 and a standard deviation of √(0.3417^2+0.4000^2)=0.5261. It is therefore possible to make a 95% confident interval: [1.165-1.645*0.5261,1.165+1.645*0.5261]=[0.2996,2.0304]. Since zero is not in the 95% confidence interval, one can say there is significant reason to assume the means in the second heuristic are lower than the means in the third heuristic. And therefore there is significant reason to assume the second heuristic to be better than the third.
The reason why this third heuristic was used – because it would save a lot of running time – also does not seem to hold true. The runtime of both heuristics seem to be comparable.

If we take a look at the second and fourth heuristic we see that, just like the prediction, the penalty based heuristic is slightly better than the heuristic that checks entire lanes for their departure times, but the difference is by far not significant. In fact, they are almost equal (in 1,000 cases there was a total difference of 7 moves, so an average of 0.007). However, we do see that the penalty based heuristic is a lot faster than the heuristics that compares departure times (entire piles and top containers). It therefore still seems better to use the penalty based heuristic, because when problems become larger, it can be very valuable to save half of the time.

To verify this, also the same heuristics were used to create 10 times 1,000 cases and compare their means, standard deviations and runtimes. The results are on the next page:

Heuristics	Mean	Standard deviation	Runtime (sec.)
One / left	41.9420	0.1408	7.339
Two /entire pile	34.0626	0.1024	18.561
Three /top container	35.2401	0.0841	16.588
Four/ penalty based	34.0579	0.1044	7.720

Table 2B: The results for removing containers, using four heuristics and 10,000 cases.

As one can see the results are quite similar to the results of 1,000 observations, the standard deviations are only much smaller. Therefore in the next part only the 1,000 cases are considered.

7.3.2.2 Comparison of heuristics every case

The retrieval of containers is done with 1,000 different cases of 20 containers stacked. For the different heuristics, the same seed is used so that the results again can be compared.

If we compare the second (where the reshuffle container is compared to entire piles) and third heuristic (where the reshuffle container is only compared to the top container), we see that in 65.2% of the cases the second heuristic is better than the third, while the third is better in only 12.6% of the cases. In the remaining 22.2% of the cases both heuristics perform the same. Due to this results it is clear that the second heuristic performs most of the time better, or at least not worse, than the third heuristic.

If we compare the second and first (always place as left as possible) heuristic, it is very clear that the second heuristic is better than the first one. In 98.8% of the cases, the second heuristic actually performs better, and in 1% the performance is the same. Only in 0.2% the first heuristic is better. However, this is negligible.

Also the third heuristic is much better than the first. In 95.8% of the cases, the third heuristic gives a better result than the first heuristic. In 2.5% of the cases they perform the same and in 1.7% of the cases the first heuristic is better. Even though the third heuristic is a lot better than the first, it still does not do as well as the second heuristic, compared to the first.

If we compare the fourth (penalty based) heuristic with the second one, one can see that they are most of the time (94.8% of the cases) equal. Only sometimes any of the two heuristics is better. Also, when we compare the fourth heuristic to the first or third, the same results as with heuristic two are found.

Comparison	Heuristic one (left) better	Heuristic two (all) better	Heuristic three (top) better	Heuristic four better (penalty)	Heuristics equal
Heuristic one vs. two	2	988	Inapplicable	Inapplicable	10
Heuristic one vs. three	17	Inapplicable	958	Inapplicable	25
Heuristic one vs. four	2	Inapplicable	Inapplicable	987	11
Heuristic two vs. three	Inapplicable	652	126	Inapplicable	222
Heuristic two vs. four	Inapplicable	16	Inapplicable	11	973
Heuristic three vs. four	Inapplicable	Inapplicable	120	653	227

Table 3: comparison of heuristics

There is one strange result in the table: the results in 7.3.2.1 one could see that the penalty based heuristic performs little better than the heuristic which compares departure times of entire piles. In the table however this heuristic performs 16 times better than the penalty based heuristic and only 11 times it performs worse. This would indicate that the second heuristic is slightly better than the fourth instead of the other way around. Therefore, another table is added, which will show the difference in moves of these two heuristics. Also an example is given of when the penalty based heuristic is better than the heuristic comparing departure times.

Difference	Penalty based heuristic	Comparison based heuristic
Times 1 move less needed	4	15
Times 2 moves less needed	5	1
Times 3 moves less needed	1	0
Times 7 moves less needed	1	0
Total times better	11	16

Table 4: the number of times the two heuristics used a given amount of steps less

As one can see, the heuristic which compares departure times of entire piles is better in 16 cases, but it only needs 1 reshuffle less in 15 of these cases. The penalty based heuristic however, is only better 11 times, but 7 of these times it needs more than one reshuffle less. One time it even needs 7 reshuffles less. Therefore, in total the penalty based heuristic uses somewhat less moves (and reshuffles), but the comparison based heuristic is better more often. As an example, below is the stack for which the penalty based heuristic uses 7 reshuffles less than the comparison heuristic.

Example 4. The stack for which the penalty based heuristic is much better

7	2	8	18	11
14	13	17	20	6
10	1	15	3	16
4	9	5	19	12

Fig 7. The stack for which penalty based heuristic is much better

The number of moves made by the penalty based heuristic is 35 for this stack, while the number of moves made by the heuristic that compares the departure times of entire piles uses 42 moves.

Beside simple one-on-one comparisons, where only two heuristics are compared, also all three heuristics are compared in one time. The results are in the table below:

The best heuristics	**Number of times the best**
Heuristic one (left)	
Heuristic two (all)	10
Heuristic three (top)	119
Heuristic four (penalty)	7
Heuristics one and two	
Heuristics one and three	
Heuristics one and four	
Heuristics two and three	1
Heuristics two and four	637
Heuristics three and four	2
Heuristics one, two and three	
Heuristics one, two and four	2
Heuristics one, three and four	1
Heuristics two, three and four	218
All heuristics	3
Total	1000

Table 5: number of times a heuristic was the best one

As one can see, the second and fourth heuristics were many times the best one (87.1% and 87.0%). Most of the time they were the best together (63.7%), or together plus heuristic three (21.8%). Heuristic three is the best in only 34.4% of the cases and heuristic one in 0.6%. This result thus again shows that the second and fourth heuristics are comparable. They both are better than the third, which is way better than the first one.

It might be a good idea to get the solution of the best heuristic in every case and see how many moves are then necessarily. The results are below:

Heuristic	Mean moves	Standard deviation of moves
Best in every single case	33.943	0.2986

Table 6: the mean number of moves and standard deviation of the best heuristic in every case.

Compared to the results in table 2A, this heuristic is not much better than the heuristic that compares the departure time of the reshuffle container to departure times of entire piles and the penalty based heuristic. It is somewhat smaller, yes, but the difference is not significant.

8 Stacking and removing containers I

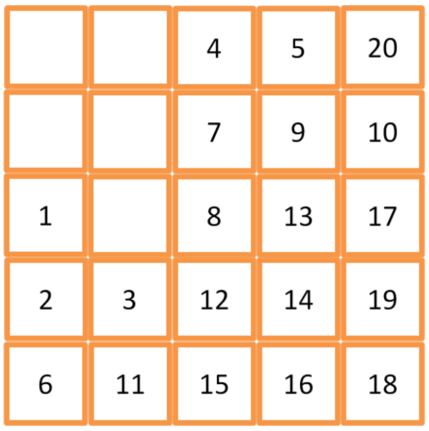
8.1 Problem

In this part, containers are not only removed, but also stacked in a way that is hopefully working well. For now, all containers arrive before the first container leaves. This means a quay is created and emptied afterwards. This stacking and removing is done by the same three simple heuristics as in part 7.

8.1.1 Creation of a quay

Because now the quay is not created randomly, the containers are not necessarily stacked in lanes of equal height. An example of a stack, created with heuristic two (explained in 8.2.2) is below:

Example 5: container 1 has to be removed first etc. empty blocks are spaces available



		4	5	20
		7	9	10
1		8	13	17
2	3	12	14	19
6	11	15	16	18

Fig 8. A quay created by placing containers more cleverly

As one can see, the containers are still not all placed in such a way that not a single reshuffle is necessarily. The number of reshuffles however has decreased dramatically, as will be seen later on. For the simulation of the order in which the containers arrived, the seed: rand('state',str2double('100')) was used.

8.1.2 Upper and lower bound.

Once the quay is created, it is easy to calculate a lower and upper bound for the number of moves made. Also, when the order in which the containers arrive is known, it is possible to calculate an upper bound by placing and reshuffling the containers as far to the left as possible. Calculating a good lower bound is not that easy. One can take as lower bound the amount of containers arriving, but this is not always a good lower bound.

8.2 Method

Just like in chapter 7, four heuristics are used. These heuristics look a lot like the heuristics in chapter 7. There are only small differences.

8.2.1 Heuristic one: the always left heuristic

This heuristic is a very simple one. Just like only the removal of containers, now also with the placement of containers always the most "left" possible place is chosen to drop a container. This means a new container is placed on top of the containers in the lane that is the most left, not maximally filled lane.

8.2.1.1 Quality of heuristic one

In section 8.1.2 this heuristic is referred to as an upper limit of the solution. This can be explained, by the fact that if place left is optimal, one should place the reshuffle container left. If it is not optimal, one should place the container somewhere else. Heuristic number three is expected to be worse than all the other heuristics.

8.2.2 Heuristic two: compare departure time to that of entire piles

This heuristic checks the departure time of an arriving container and compares this to the departure time of container in entire lanes. If there exists at least one lane for which all containers depart after the arriving container, a second check is made. This step, step 1, remarkably looks like step 2 in heuristic 7.2.2.

The lanes gotten in step 1 just cannot be filled maximally, because than it is impossible to place another container on top of it. If there is at least one of these lanes not filled maximally, the arriving container is placed on the lane for which the top container leaves the soonest after it. This is step 2, and looks very much like step 3 in 7.2.2

If it is not possible to place a container on top of a lane with all containers leaving after the arriving container, there is checked whether there is at least one empty lane. If that is true, the arriving container is place on the ground floor. Step 3 (this one) and Step 4 of 7.2.2 very much look alike.

If there is also no empty space, the container has to be place on top of a stack that contains a container leaving before this container. That means the newly arrived container will have to be reshuffled in the future. To minimize this reshuffling, the container is preferably placed on top of a container that leaves before the now arriving container. This because when reshuffle is necessarily, the container that leaves the latest will be placed down. The check if a top container leaves before the arrival container is step 4. This step is the same as step 5 in the heuristic of 7.2.2

An extra in this heuristic is that when the heuristic has come this far, it first checks whether the top container is the container that leaves the soonest in the lane. Preferably, we do not put a container on top of it now. If the top container does not leave the soonest, it is much less bad to put a container on top of it, since it already needed to be reshuffled.

Therefore, if possible, the newly arrived container is placed on the latest leaving top container, which is not the first container in the lane to leave. If not possible, we still try to place the container on a container that leaves earlier, even if this container is the first container to leave in the lane. When there is choice between two or more different lanes, choose the lane for which the top container leaves the closest to the newly arrived container.

If the now arriving container is still not placed after checking all previous points, we place the container on top of the container which leaves after it. The lane in which the newly arrived container is place is chosen randomly. Again, the maximum amount of containers stacked on top of each other considered. This is step number 6 in this heuristic.

8.2.2.1 Quality of heuristic one

For the quality of this heuristic the same holds true as for the quality of the heuristic in 7.2.2. This heuristic is also not expected to give the best results at all times.

8.2.3 Heuristic three: compare departure time to that of the top containers

This heuristic, which is the leveling with departure times (LDT) heuristic discussed by Borgman, van Asperen and Dekker in [2], [3] and [4], pretty much looks like heuristic 7.2.3. The top containers are checked. If there exists at least one not maximally filled lane for which the top container leaves later than the newly arriving container, this new container is placed on top of it. The new container is placed on the container that leaves the soonest after the new one. See step 5 of heuristic in section 7.2.2.

If that is not possible, than the heuristic checks whether there is an empty spot. If so, the container is placed on one of these spots. If not, the heuristic goes to step 3.

Step 3 is quite simple. Since the first two steps were not possible, this step is always possible (due to the limitations of the number of containers. The new arrived container is now placed on the container that leaves the closest before the new arrived container.

8.2.3.1 Quality of heuristic three

Just like in the previous chapter, this heuristic is expected to be little worse than the previous one, this because there is only looked at the top container instead of to the entire lane. This means more reshuffles can be necessarily.

8.2.4 Heuristic four

The fourth heuristic is the so-called penalty based heuristic. Not only are the containers reshuffled based on penalties, but they are also placed based on penalties. The penalties are just as large as they were in chapter seven (beside the penalty for placing the container at the same spot, because now there is no same spot) and the same criteria are “awarded” with a penalty; placing a container in a lane containing at least one container that leaves before. If that is true, another penalty on top of it if the top container departs after the newly placed container.

The two other penalties are the penalties for blocking spaces and the absolute value of the time between departure of the top container of a lane and the newly arrived container.

8.2.4.1 Quality of heuristic four

Just as in the previous chapter, this heuristic is expected to be somewhat better than heuristic one, where departure times of new containers are compared to departure times of containers in entire lanes, but the difference will not be significant.

8.3 Results

Again 1000 cases were created. In these cases the order in which the containers arrived differed. In this section, we check which heuristic works the best. It is split into several parts. First, the mean of the number of moves made by the four heuristics and their standard deviations will be discussed. Secondly, all 1000 different amounts of moves are compared to each other in order to find out which heuristic is the best. Then we change the size of the container block and the number of containers placed and finally also the shape of the container block is changed.

8.3.1 Mean and standard deviations

In the table below are the means and standard deviations of the three heuristics. The standard deviations are the standard deviations of 10 means over times 100 trials. The results are remarkable: the number of moves have dropped by 10.143 for heuristic two, 11.427 for heuristic three and 10.999 for heuristic four. If we remember that 20 of the moves are necessarily for removing containers, we can calculate that for heuristic two 10.143/14.193=71.46%, for heuristic three 11.427/15.358=74.40% and for heuristic four 10.099/14.186=71.19% of the reshuffles are not necessarily anymore because of the better placing. These results can be seen in figure 9:

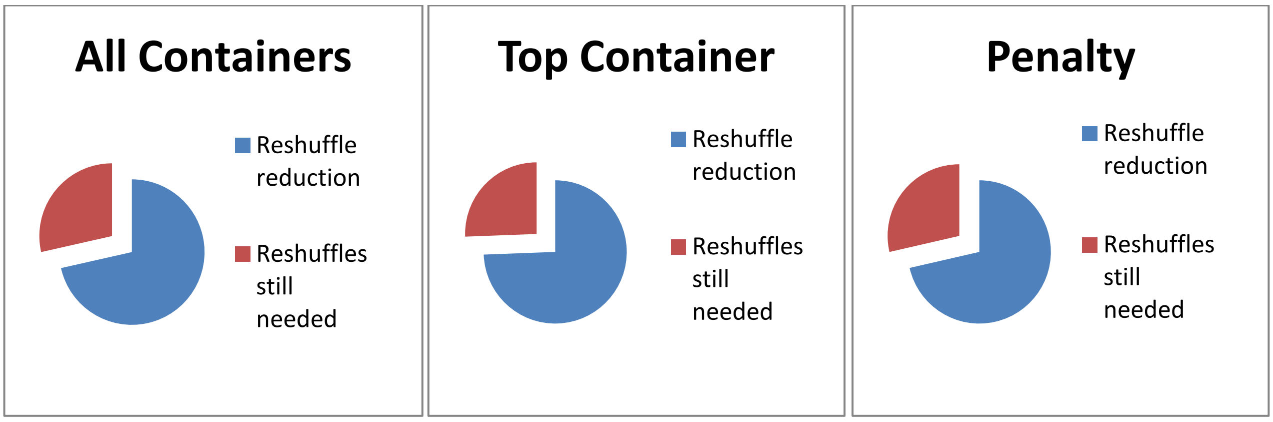


Fig 9: The reduction in reshuffles after placing according to the heuristics

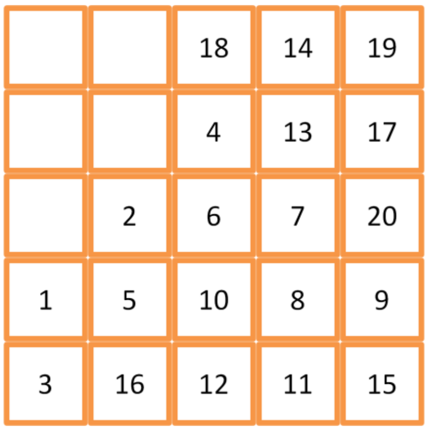
The always-left strategy, on the other hand, seems even to be worse than the random placing strategy, because the mean of moves has even increased by 1.
As one can see in the table below, the first heuristic, where both container that arrives and a reshuffle container is placed left as much as possible, is much worse than the other heuristics. There is no test needed to show that this heuristic is significantly worse than the other three.

Heuristics	Mean	Standard deviation	Runtime (sec.)
One / left	42.964	0.6227	1.016
Two / all	24.038	0.3481	1.750
Three / top container	23.896	0.3391	1.418
Four/ penalty	24.087	0.3440	0.962

Table 7: The results for stacking and removing containers, using three heuristics.

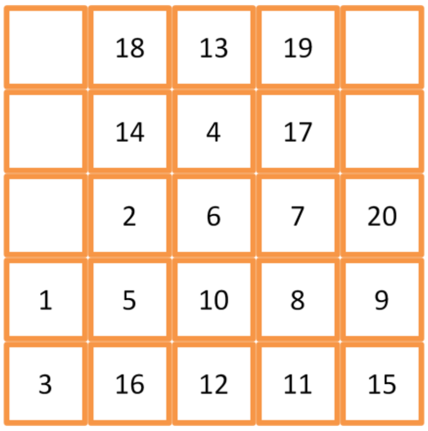
Remarkable is that the third heuristic (only compare to the top container) performs better than the second (compare with all containers in a pile), and performs better than the fourth one (based on penalties), even though it is just a fraction. This is counter-intuitive, because for the removing of containers, the second and fourth heuristic performed significantly better than the third one. With a test one can calculate what already can be seen. The third heuristic is better than the second and fourth, but only a fraction, something like half the standard deviation of heuristic three. It can therefore be said that there is no significant difference between the three heuristics. An example of a stack created by the second and third heuristic, when the second heuristic performs better, is given below:

Example 6. Stacks created by heuristics two and three, when heuristic two is better



		18	14	19
		4	13	17
	2	6	7	20
1	5	10	8	9
3	16	12	11	15

Fig 10. A stack created by heuristic three



	18	13	19	
	14	4	17	
	2	6	7	20
1	5	10	8	9
3	16	12	11	15

Fig 11. A stack created by heuristic two

Above are two stacks; the left one created by heuristic three and the right one by heuristic two. For both stacks the containers arrived in the exact same order, so the placement was really the difference. Heuristic three used only 28 moves to clear its stack, while heuristic two needed 35 moves to do so. The main reason why the stack created by heuristic two is worse is that once a container cannot be placed on top of a pile where every container departs later, it is placed on top of a pile with an earlier leaving container on top. So in this case, there will never be placed a container on top of container number 20, even though that would be better.

Again, just like in chapter 7, the penalty based heuristic works a lot faster than the other heuristics (the always-left heuristic not taken into account). In this case, it does not matter that much, but once the problems become much larger, it might be better to use this heuristic. What also can be seen is that the third heuristic now also is somewhat faster than the second, which was the reason it was compared in this thesis.

8.3.2 Comparison of heuristics every case

The stacking and retrieval of containers is done with 1000 different cases of 20 containers arriving. For the different heuristics, the same seed is used so that the results again can be compared. This seed can be found in 8.1.1.

If we compare the second (compare departure time of newly arrived container to the departure time of all containers in the lanes) and third heuristic (compare departure time of newly arrived container to the departure time of the top container in the lanes), we see that in 46.2% of the cases the second heuristic is better than the third, while the third is better in only 43.2% of the cases. In the remaining 10.6% of the cases both heuristics perform the same. Due to this result one could say that heuristic two performs little better than the third one, but again this is highly disputable. Again the two heuristics seem to perform quite the same.

If we compare the second and first heuristic (always place as far to the left as possible), it is very clear that the second heuristic is better than the first one. In 99.5% of the cases, the second heuristic actually performs better. In 0.1% the performance is the same and in 0.4% the first heuristic is better. However, these two numbers are negligible.

Also the third heuristic is much better than the first. In 99.9% of the cases, the third heuristic gives a better result than the first heuristic. In 0.0% of the cases they perform the same and in 0.1% of the cases the first heuristic is better. As one can see, the third heuristic performs even better compared to the first than the second heuristic.

For the fourth heuristic (place based on penalties), one can see it is very much comparable to heuristic two. They are most of the times equal and also compared to heuristic one and three they give the same results.

Comparison	Heuristic one (left) better	Heuristic two (all) better	Heuristic three (top) better	Heuristic four better (penalties)	Heuristics equal
Heuristic one vs. one	995	Inapplicable	4	Inapplicable	1
Heuristic one vs. three	Inapplicable	999	1	Inapplicable	0
Heuristic one vs. four	Inapplicable	Inapplicable	4	995	1
Heuristic two vs. three	462	432	Inapplicable	Inapplicable	106
Heuristic two vs. four	47	Inapplicable	Inapplicable	21	932
Heuristic three vs. four	Inapplicable	436	Inapplicable	461	103

Table 8: comparison of heuristics for stacking and removing

Beside simple one-on-one comparisons, where only two heuristics are compared, also all three heuristics are compared in one time. The results are in the table below:

As one can see, the second heuristic was the best one in 56.3% of the cases. For heuristic three this is now also more than 50%, 53.5%, heuristic four is best in 55.3% and of heuristic one it became 0%. This result thus again shows that the second, third and fourth heuristics are much better than the first, but there is not one heuristic that is clearly better or worse than the others.

The best heuristics	Number of times the best
Heuristic one (left)	
Heuristic two (all)	13
Heuristic three (top)	431
Heuristic four (penalty)	6
Heuristics one and two	
Heuristics one and three	
Heuristics one and four	
Heuristics two and three	3
Heuristics two and four	446
Heuristics three and four	
Heuristics one, two and three	
Heuristics one, two and four	
Heuristics one, three and four	
Heuristics two, three and four	101
All heuristics	
Total	1000

Table 9: number of times a heuristic was best stacking and removing

8.3.3 Comparison of larger blocks

To check if the results depend on the size of the block, the same research is done for blocks of 6-by-6 and 30 containers, 7-by-7 and 42 containers, 8-by-8 and 56 and 9-by-9 and 72 containers (so that still the top tier could stay entirely empty). Also the same seed was used as in the previous parts. We now only compare the second heuristic (check entire lane) with the third heuristic (check top container).

Heuristic two vs. three	Heuristic two (all) better	Heuristic three (top) better	Heuristics equal
5 by 5	462	432	106
6 by 6	181	288	531
7 by 7	264	427	309
8 by 8	274	548	178
9 by 9	243	690	67

Table 10: the comparison of heuristic one and two for larger blocks

The results looked quite like the results of the 5-by-5 block, even though the differences became somewhat larger (as did the standard deviations). The number of reshuffles necessarily seemed to double for every increase of the block with 1.

Heuristic three seems to become somewhat better once the block becomes larger. An explanation for this is that the situation described in example 6 (page 23) is happening a lot more often once the number of containers increases while the block does not get longer. But these results do not mean that the first heuristic is much worse, because as a matter of fact, there will never be container terminals that stack more than 6 containers on top of each other. Therefore, also the next test is done.

8.3.4 Comparison of different shaped blocks

To see which heuristic works better in a more realistic environment, a block with a higher amount of lanes than tiers is created. Remember, when there are more stacks, this can be converted into one stack with more lanes. The blocks created have 10 lanes and 5 tiers, with 40 containers, and 20 lanes and 5 tiers, with 80 containers. The amount of containers is chosen so that the same proportion of containers/space for containers is used as in the stack with 5 lanes and 5 tiers. We again use the same seed as before.

Heuristic two vs. three	Heuristic two (all) better	Heuristic three (top) better	Heuristics equal
5 by 5	462	432	106
10 by 5	20	16	974
20 by 5	0	0	1000

Table 11: the comparison of heuristic one and two for different shaped blocks, part 1.

As one can see, the longer the block becomes, the less difference is there is between the two heuristics. This because there will be more places available to place a container and therefore less times a container has to be placed on top of (or on a pile in which there is) another container that departs before this new container. What is also striking is that the number of reshuffles decreases by a factor three every time the number of lanes doubles. For the 20 by 5 block, only 0.077 reshuffles are on average necessarily.

Just to check whether the heuristics really look like each other when the quay becomes longer, we finally compare them in a block of 120 lanes, 6 tiers and 683 containers. This 683 is 1 less than the 95% of the total spaces. Once the 95% is reached, there will be too many containers and they will not be placed.

Heuristic two vs. three	Heuristic two (all) better	Heuristic three (top) better	Heuristics equal
120 by 6	0	0	1000

Table 12: the comparison of heuristic one and two for different shaped blocks, part 2.

In this case, it is for both heuristics in all 1000 cases place the containers in such a way that reshuffling is most of the times not necessarily. The average reshuffle is 0.012 for both heuristics. The heuristics thus perform quite well on a larger scale, and are quite fast as well. 1000 times placing and removing 683 containers in the best way (the reshuffling included) took 62 seconds for heuristic two and 57 for heuristic three.

8.4 The departure time of not-yet arrived containers is already known

In the previous section the departure time of the containers that had not arrived container were not known. Now, we see if we can place the containers more cleverly and make some reshuffles before all containers have arrived.

The main idea is this: if a container arrives the first check is made. Once the heuristic reaches the check where a container would be placed on the ground, various other things are checked before.

First there is checked how many empty spaces are still available. If there are only one or two empty spaces available, we will place only the latest or second latest departing containers (of the containers that still have to arrive) on an empty spot.

If another container arrives we place this container on a pile with a container that will leave earlier. Preferably, the container is placed on top of non-maximally filled lane with on top a container that leaves before the newly arrived container (and of them, the latest leaving container). Otherwise on top of a container that leaves after the new arrived container (and then the earliest leaving container).If a number of containers equal to the maximum stacking height have arrived, but still not one of the latest arriving containers, a entire lane is filled with these containers.

If one of the latest container arrives, it will be placed on the ground floor. This container is than saved in the container block and all containers that were placed on top of an earlier leaving container are placed on top of this late leaving container. Of course, that placing happens in such a way, no further reshuffles are necessarily.

In this section we only check whatever happens with the second heuristic if this extra information is used, the other heuristic are not considered.

8.5 Results

he results are split in two parts. Just like in section 8.3 first the means, standard deviations and runtimes are compared and after the 1000 separate cases are compared. To get these results the seed rand('state',str2double('100')) was used.

8.5.1 Mean and standard deviation

Below is the table with the mean and standard deviations of the number of moves made:

Kind of information of departure time	Mean moves	Standard deviation moves	Runtime (sec.)
Not all known	24.038	0.3481	1.750
All containers known	23.164	0.2668	2.998

Table 13: the mean moves and standard deviations with different kinds of information.

If one takes a look at this, it seems quite profitable to have the information about the departure time of the not yet arrived containers. The number of moves (and reshuffles) have decreased by almost one, which is almost 25% of the number of reshuffles needed. This reduction in number of moves is a significant one, so it is safe to say that there is significant reason to assume information decreases the number of reshuffles. On the other hand, the gains are not that high and the runtime of this program is much longer (71%). We do not know if the gains outweigh the extra costs for more computation time. Therefore, we compare the 1000 different cases in the section below.

8.5.2 Different cases

To take a look at the 1000 different cases, the next table is entered.

Comparison	Information better	Information worse	Does not matter
Information vs. not	518	160	322

Table 14: the comparison of 1000 cases

As one can see there are differences between knowing the information about jet to arrive container and not knowing. Knowing the departure times is profitable in 518 cases, which is more than 50%. In two third of the rest of the cases, there is no difference. Only in 160 cases the information gives a worse result. Again, this gain because of less reshuffles has to be weighed against the costs of extra information (extra time in computation, but also money to be paid in order to get the information of when a container leaves). If it is better to have the information also depends on the preferences of the management of container terminal operators.

9 Stacking and removing containers II: online information

9.1 Problem

Now that the heuristics are compared to each other when precise information is available and all containers arrive before they leave, it is time to make the scenario somewhat more realistic.

Therefore, we do now simulate the arrival and departure of containers, still both arriving and leaving one by one. This can be explained as follows: the containers can only be removed one by one out of a ship that stays at a dock. Also, on average, the same amount of containers has to leave as arrive. If not, the container terminal might either become empty (more leaving than arriving) or flow over (more arriving than leaving). A container might leave between two arrivals of containers, unloaded of the same ship. All this is the same as it has been in the previous chapter. The difference between this chapter and chapter 8 is that we do not know the departure time of the containers, beside the fact that now containers can still arrive after the first container leaves.

9.2 Method

We do not know anything about the containers that still have to arrive, and now we still do know something about the departure times of the containers that have arrived. Unfortunately, we cannot say for sure what the departure time is. All we know is the distribution of the time the containers will stay at the terminal.

This section is split into three sub-sections: the arrival and placement of containers, the departure of a container and reshuffles and the rest of the simulation.

9.2.1 The arrival of containers

Compared to when we do not know everything about the departure time of containers, a newly arrived container when we only know the distribution of the storage time of the containers is much harder to place. However, the containers are placed in exactly the same way as in chapter 8. We now only cannot use the real departure time of the containers (because we do not know this), but we use the expected departure time of the container.

All the containers have the same distribution. Two different distributions are used:

- The storage time is randomly distributed between 0 and 2 * the number of containers at the start of the simulation. This means the average storage time is the number of containers at the start of the simulation.

- The storage time is exponentially distributed with mean lambda. This lambda is equal to the number of containers at the start of the simulation. This also means the average storage time is equal to the number of containers at the beginning of the simulation.

The containers that are already at the container terminal at the beginning of the simulation will have a different distribution. This is explained in 9.2.3.
For both distributions it holds true that the average amount of containers at the container terminal will be close to the number of containers at the beginning of the simulation (since on average the same amount of containers arrive at- as leave the terminal). A next arrival is generated in the same way as in the previous section.

If we have an expected value of the time containers leave, the placement can be done with five kinds of heuristics: the first one is the second heuristic of 8.2. This heuristic looks at the entire piles and compares the departure time of the first leaving container of that pile to the newly arrived container. The second one is the third heuristic of section 8.2. This heuristic only compared the top container of each pile to the new container. Then we have the penalty based heuristic of section 8.2 and the stacking of containers by using leveling (where containers are always placed in the lane with the least containers in it). The last heuristic is the random one, where a newly arrived container is placed randomly.

9.2.2 The departure of containers

If it is time for a container to leave the container terminal, one has to check where this container is and if there are any containers on top of it. If there are no containers on top of the departing container, we are very happy; now the container can be placed on a truck and leave the terminal without any reshuffle. If the container, however, is not on top of a lane, other containers have to be reshuffled. This reshuffling is done in the same way a new container would be placed. Of course, the lane they were in are not considered.

The reshuffling happens with one of the heuristics of chapter 7. The combination in which the heuristics are used will be explained below.

If we have an expected value of the time containers leave, the reshuffling can be done with four kinds of heuristics: the first one is the second heuristic of 7.2. This heuristic looks at the entire piles and compares the departure time of the first leaving container of that pile to the reshuffle container. The second one is the third heuristic of section 7.2. This heuristic only compared the top container of each pile to the reshuffle container. The third heuristic is the last heuristic of 7.2, where we make reshuffles based on penalties for all lanes. Then there is the leveling heuristic, where a reshuffle container is placed in the lane with the least containers stacked in it. The last heuristic is the random one, where a reshuffle container is reshuffled randomly.

The heuristics are used in combination with the heuristics used for stacking. We use the second heuristic of section 7.2 with the second heuristic of section 8.2. We also use the third heuristics of section 7.2 and 8.2 together, the fourth heuristic of section 7.2 with the fourth heuristic of section 8.2 and the leveling heuristics as well. Finally, both the random heuristics are used together.

But because heuristic two of 7.2 seemed significantly better than heuristic three and together with heuristic two in section 8.2 not better than both heuristics three, one might say heuristic three stacks the containers in a better way than heuristic two. Therefore we also use a sixth combination. This combination places the containers according to heuristic three of section 8.2 (check only the top containers) and reshuffles them according to heuristic two of 7.2 (check entire piles). This might give even better results for the simulation.

9.2.3 The rest of the simulation

The rest of the simulation consists of the input, the initialization, the scheduling of a next event, the 'main' simulation and the replications.

To start the input: the user of the simulation can set various variables as input. The times the simulation is repeated, the length of one simulation replication (in time units), the seed of the random generator and limit value of the percentage the terminal may be filled. Also one can enter the number of containers already at the terminal at the beginning of the simulation, the number of lanes a terminal has and the maximum height containers can be stacked at the terminal.

Now let us continue with the initialization: the containers that are at the terminal at the beginning of the terminal must all have a place. These containers are chosen in a random order. If we know the distribution of the storage time of the containers, we compose the time they will leave the terminal and their expected departure time. We then place the containers based on their expected departure time.
Furthermore, the starting time, the first arrival and the number of containers at the terminal are determined.

After the initialization there is the scheduling of a next event. This is nothing more than the minimum of the list of all events that can happen (the arrival or the departure of a container). This function will indicate the time the event happens and whether it is a departure or an arrival.

We also have the replicate function. This function makes sure that the main function of the simulation will be executed several times. Every time the results will be stored and after all replications means will be taken.

The most important function is the main function. This main function will perform one replication. Every replication an initialization is made first. Afterwards, we will continue the simulation until the stopping criterion is reached (time is equal to the length of the simulation inputted). First, the next event is scheduled, than either an arrival or departure (with reshuffles) is simulated. Once the stopping criterion is reached, the time of a new arrival is set on infinity. The rest of the containers at the container terminal are then removed in order of departure time. Again reshuffles are made if necessarily. Finally the results for this replication are stored at the replication function.

In appendix B one can find the simulation function, where the storage times are known to be randomly distributed. Comments will tell what happens at every part of the simulation.

9.3 Results

Before we will discuss the results, the inputs have to be stated.
To start with the seed of the random generator: we again used the same seed as in chapter 7, which was rand('state',str2double('12345')).
Furthermore we use a starting number of 300 containers and the number of time units used is 300 as well. This means there will be 600 containers at the container terminal. 600 moves therefore are made for sure. All the extra number of moves is made because of the reshuffles.

Of course we also need the size of the container terminal and a limit value of the stocking rate. These are inputs as well: the maximum stacking height is equal to 6 (so 6 containers on top of each other) and the number of lanes is equal to 60. This means a maximum of 360 containers can be placed at the container terminal. Of course we do not like 360 containers at the terminal, because now we cannot reshuffle a container if necessarily. We therefore set the maximum filling rate at 95%. Now a maximum of 342 containers can stay at the container terminal at the same time.

We now know the input variables, so let's go to the results of the simulation. As results there is the number of moves made using different heuristics and different kinds of distribution/knowledge for the storage time of a container.

9.3.1 The number of moves for different kinds of information

Below is the table in which the number of moves is. These moves are both removal (600) and reshuffle moves (the rest).
As a reminder: for the first line the second heuristics of 7.2 and 8.2 are used. These heuristics compare the departure time of the reshuffle/new-to-place container to the departure times of all containers in the lanes. For the second line the third heuristics of 7.2 and 8.2 are used. These heuristics compare the departure time of the reshuffle/new-to-place container to the departure times of the top containers of the lanes. For the third line the second heuristic of 7.2 and the third heuristic of 8.2 are used. Below, line 4, we used the penalty based heuristics (fourth) of 7.2 and 8.2. Finally, for the last two lanes we used the leveling heuristic and random heuristics respectively.

Heuristics used	Storage times randomly distributed	Storage times exponentially distributed
2 stacking & reshuffling	1441.32	1531.20
3 stacking & reshuffling	1448.12	1532.64
3 stacking, 2 reshuffling	1439.05	1532.21
Penalty stacking & reshuffling	1443.47	1544.50
Leveling	2135.99	2153.15
Random stacking	2109.73	2074.98

Table 15. The number of moves made for different kinds of departure times and use of heuristics.

As one can see are the results quite comparable. It does not seem to make much difference which heuristics are used to stack new containers or reshuffle, only the random stacking and the leveling work a lot worse (they have an extra number of moves of 50%), just like Borgman (2009), [4], already found out before. Because now the exact departure time of the containers is not known, the number of reshuffles increases tremendously.

The reason for the increased number of reshuffles is quite simple: containers are stacked based on their *expected* departure time. The order in which containers leave based on their *real* departure time however, is different. Therefore the wrong assumptions about the order in which the container depart are made at several occasions. So even though we think we placed the containers in an optimal way (based on expected departure times) we did not and need to reshuffle some containers. The simulations also are quite fast. One hundred times the arrival of 300 extra containers and the removal of 600 containers takes between 5 and 7 seconds for all heuristics.

10 Conclusion

In this thesis various heuristics were compared. This comparison was done in several ways: the removal of an already existing container stack, the creation of a container stack and then the removal of this container stack and the placement of new containers and removal of already placed containers at the same time. By doing so, we saw that the tested heuristics worked better than random stacking and reshuffling and also better than leveling and the always-place-as-far-to-the-left-as-possible heuristic.

There were three heuristics tested:

- The first heuristic compares the departure time of a container that arrives or has to be relocated to the departure times of all containers. If there exists a lane in which all container leave after this “new” container, this container is placed there. If not, the container is (whenever possible) placed on the ground floor. Only then the heuristics takes a closer look at the top containers.
- The second one is the so-called leveling with departure times (LDT) heuristic, which is already used in previous research ([2],[3] and [4]). This heuristic compares the departure time of a container that arrives or needs to be relocated to the departure times of the top containers at the stack. If there leaves no container later than this container a container is (if possible) placed on the ground floor.
- The third heuristic is a heuristic based on penalties. For the placement of a new seq. reshuffle container this heuristic gives every lane a penalty. The container is then placed at the lane which has the lowest penalty.

Which of the heuristic used did not seem to matter that much, only when an already existing stack is removed it seemed better not to use the heuristic where the departure time of containers that need to be relocated is compared to the departure time of the top containers only.
The other two heuristics, which are the heuristic where placement and reshuffling is based on penalties and the heuristic where the departure time of new arrived (and reshuffle) containers is compared to the departure times of all containers in the lanes, seem to give almost exactly the same results for every comparison.

Still it might be better to use the penalty based heuristic rather than another, because the computation time of this heuristic is a lot smaller than the computation time of the other two heuristics.

Knowing information about the departure time of not yet arrived containers did seem to improve (so decrease) the number of moves made. However, it did cost extra time to run the programs in the mathematical program matlab. Therefore, and because in this world nothing (so information also not) is for free, it still might be better not to try to get information about the departure times of containers that still have to arrive.

References

[1] Günther H-O, Kim KH, (2006), Container terminals and terminal operations. OR Spectrum 28, p.437–445

[2] Borgman, B., van Asperen, E. and Dekker, R., (2010), Evaluating container stacking rules using simulation. Proceedings of the 2010 Winter Simulation Conference

[3] Borgman, B., van Asperen, E. and Dekker, R., (2010), Online Rules for Container Stacking. OR Spectrum 32, p.687-716

[4] Borgman, B., (2009), Evaluation of Online Container Stacking Strategies Using Simulation: B, Borgman (Msc Thesis). Erasmus University Rotterdam

[5] Froyland, G., Koch, T., Megow, N., Duane, E., and Wren, H., (2008), Optimizing the landside operation of a container terminal. OR Spectrum 30, 53-75

[6] Hartmann, S., (2004) Generating scenarios for simulation and optimization of container terminal logistics. OR Spectrum 26, p.171-192

[7] Wu, K-C. and Ting, C-J., A beam search algorithm for minimizing reshuffle operations at container yards. Department of Industrial Engineering and Management, Yuan Ze University

[8] Kim, K. H. and Hong, G-P., (2006), A heuristic rule for relocating blocks. Comput. Oper. Res. 33, p.940-954

[9] http://ashasmaritimenews.blogspot.com/2011/04/container-terminal.html

Appendix

Appendix A. The code of section 8.2.1

```
function
[containerstacks]=Placecontaineronplaceway3(containerstacks,container,...
... maximumhoogte)
welofgeencontainers=(containerstacks~=1200);
containergeplaatst=0;
keuze=[];
hoogtes=sum(welofgeencontainers);
topcontainers=zeros(1,size(containerstacks,2));
for a=1:size(containerstacks,2)
    if hoogtes(a)>0
        topcontainers(a)=containerstacks(maximumhoogte-hoogtes(a)+1,a);
    else topcontainers(a)=1200;
    end
end
if any((topcontainers>container).*(topcontainers<1000))==1
    keuze=[];
    for i=1:size(containerstacks,2)
        if topcontainers(i)>container && topcontainers(i)<1000 &&
hoogtes(i)<maximumhoogte
            keuze=[keuze;topcontainers(i)];
        end
    end
    if sum(keuze)~=0
        [opdeze,~]=min(keuze);
        [rij,kolom]=find(containerstacks==opdeze);
        containerstacks(rij-1,kolom)=container;
        containergeplaatst=1;
    end
end
if containergeplaatst==0 && any(hoogtes==0)==1
    [~,plek]=min(hoogtes);
    containerstacks(maximumhoogte,plek)=container;
    containergeplaatst=1;
end
if containergeplaatst==0
   keuze=[];
   for i=1:size(containerstacks,2)
       if topcontainers(i)<container && hoogtes(i)<maximumhoogte
           keuze=[keuze;topcontainers(i)];
       end
   end
   [opdeze,~]=max(keuze);
   [rij,kolom]=find(containerstacks==opdeze);
   containerstacks(rij-1,kolom)=container;
   containergeplaatst=1;
end
end
```

Appendix B. The simulation with randomly distributed storage times

```
function SimulationofContainers
% In this function the arrival and departure of containers at a container
% terminal are simulated. because only the order of arrival matters, not
% the actual time, we say that every 1 time unit a new container arrives.
%   This function exists of a main function, a replicator function, an
%   input function, the initialization, a function to get the next even and
%   a arrival and departure function.
clc;
%get the input
inputdata;
% do the main function multiple times
[~,meanmoves,meanpercentagenotplaced,meancontainers] = replicate;
%print the results
fprintf('The mean number of moves is: %.6f\n', meanmoves);
fprintf('The average percentage not placed is:
%.6f\n',meanpercentagenotplaced);
fprintf('The average number of containers at the terminal is:
%.6f\n',meancontainers);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function inputdata
%gets the input for the simulation
global T s alfa containers maximumhoogte aantallanes
%enter the several data we require for the simulation
prompt  = {'Times repeated','Length of simulation run',...
    'Seed of random number
generator','drempelwaarde','aantalcontainersbegin',...
    'maximum hight','number of lanes'};
def     = {'100','300','12345','0.95','300','6','60'};
titel   = 'input';
lineNo  = 1;
parmss  = inputdlg(prompt,titel,lineNo,def);
s = str2double(parmss{1});
T  = str2double(parmss{2});
seed = str2double(parmss{3});
alfa = str2double(parmss{4});
containers = str2double(parmss{5});
maximumhoogte = str2double(parmss{6});
aantallanes = str2double(parmss{7});
l = RandStream.getDefaultStream;
reset(l);
%set the seed for the random generator, so we can compare the different
%heuristics.
rand('state',seed);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function runstat = main
% Perform one replication
global T moves maximumhoogte
%initialize the different variables we will use
[t, tE,containeraantal,aantalcontainers,eventlist, tDexp,containerstacks,
nietgeplaatst]= initialization;
containervar=[containeraantal,aantalcontainers];
while t<=T    % Stopping criterion
    [t,i] = schedule_next_event(eventlist); % get next event
    if i==1 % first event is arrival, so schedule a next arrival
```

```
        [tDexp,containervar,eventlist,containerstacks,nietgeplaatst] =
arrivalrealisation(tDexp,t,tE,containervar,eventlist,containerstacks,nietge
plaatst);
    else %since the next event is no arrival, it will be a departure, so
        % schedule next departure
        [containervar,containerstacks, eventlist, tDexp] =
departurerealisation ( t,tE,containervar,i,containerstacks, eventlist,
tDexp);
    end
    tE=t;
end
%if all containers have arrived, we also remove the containers that are
%still at the terminal, so the terminal ends empty
welofgeencontainer=(containerstacks~=12000);
eventlist(1)=inf;
%remove as long as there is at least one container at the terminal
while sum(sum(welofgeencontainer))>0
    %find the container that needs to be reshuffled
    [~,i] = schedule_next_event(eventlist);
    [rij,kolom]=find(containerstacks==tDexp(i-1));
    %if there is at least one container on top of the container that needs
    %to be removed
    if sum(welofgeencontainer(1:rij,kolom))>1
        %reshuffle these containers
        [ containerstacks ,moves ] = MoveWithContainersway2(
containerstacks ,moves,rij,kolom,maximumhoogte );
        %remove the container that has to be removed.
        containerstacks(rij,kolom)=12000;
        moves=moves+1;
        % no containers on top of the to remove container, remove it
    elseif welofgeencontainer(rij,kolom)==1
        containerstacks(rij,kolom)=12000;
        moves=moves+1;
    end
    %expected and real departure time of this container is set to infinity
    tDexp(i-1)=inf;
    eventlist(i)=inf;
    welofgeencontainer=(containerstacks~=12000);
end
%get the total number of moves and percentage not placed this simulation
%run and the average number of containers at the terminal
runstat(1)=moves;
runstat(2)=nietgeplaatst/(T);
runstat(3)=containervar(1)/T;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [ matrix,meanmoves,meanpercentagenotplaced,meancontainers ] =
replicate
global s
% number of replications is s. Do main function s times
matrix=zeros(s,3);
for i=1:s
    runstat = main;
    %stire results of this time
    matrix(i,:)=runstat;
end
%get the mean number of moves and the mean percentage of containers not
%placed
meanmoves=mean(matrix(:,1));
meanpercentagenotplaced=mean(matrix(:,2))*100;
meancontainers=mean(matrix(:,3));
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [t,tE,containeraantal,aantalcontainers,eventlist,tDexp,
containerstacks, nietgeplaatst]=initialization
%initalize the variables
global T containers maximumhoogte aantallanes moves
%time is zero
t=0;
% the container terminal is set empty
containerstacks=12000*ones(maximumhoogte,aantallanes);
%first container arrives at time 1
tA=1;
tD=zeros(T+containers,1);
tDexp=inf*ones(T+containers,1);
for i=1:containers
    j=12000;
    %pick a container not used, this will give the contianers already at
    %the terminal at t=0 in a random order
    while sum(sum(containerstacks==j))>=1
        j=ceil(containers*rand);
    end
    %the departure times of the containers are random distributed on the
    %interval [0,2*containernumber], therefore their expected values are
their
    %containernumber. The actual departure time is somewhere in the
    %interval.
    tDexp(i)=j;
    tD(i)=rand*2*j;
    % the containers already stacked at t=0 are placed at the terminal by
    % the heuristic based on the expected leaving time

[containerstacks]=Placecontaineronplaceway2(containerstacks,tDexp(i),maximu
mhoogte);
end
%the actual leaving time of jet to arrive containers is set on infinity
tD(containers+1:T+containers)=inf;
%all events are the arrival of a new container and the departure of the
%containers
eventlist=[tA;tD];
%total number of moves starts at 0, same as the amount of containers not
%placed because the terminal has already to much containers.
moves=0;
nietgeplaatst=0;
%time of last event and total time any container has been at the terminal
%are zero at beginning of simulation. number of containers at the terminal
%is equal to the input number of containers.
tE=0;
containeraantal=0;
aantalcontainers=containers;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [t,i]=schedule_next_event(eventlist)
%schedule the next event. This is the minimum time of the eventlist. i is
%the spot this minimum time is at.
[t,i]=min(eventlist);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [tDexp,containervar,eventlist,containerstacks,nietgeplaatst] =
arrivalrealisation(tDexp,t,tE,containervar,eventlist,containerstacks,nietge
plaatst)
```

```
%a container arrives
global containers alfa maximumhoogte aantallanes
%the total storage times increases by the time since the last event times
%the number of containers at the terminal
containervar(1)=containervar(1)+(t-tE)*containervar(2);
%find out on which spots a container is located
welofgeencontainer=(containerstacks~=12000);
%if there will be too much containers at the terminal, do not place the
%newly arrived container.
if ((sum(sum(welofgeencontainer))+1)/(maximumhoogte*aantallanes))>alfa
    eventlist(1)=eventlist(1)+1;
    nietgeplaatst=nietgeplaatst+1;
else %calculate expected time the just arrived container will leave.
    tDexp(t+containers)=t+containers;
    %update the arrival time of new container and real departure time of
    %just arrived container
    eventlist(1+t+containers)=t+rand*2*containers;
    eventlist(1)=eventlist(1)+1;
    %place the container based on the expected time the container will
leave the terminal again

[containerstacks]=Placecontaineronplaceway2(containerstacks,tDexp(t+contain
ers),maximumhoogte);
    %one more container at the terminal
    containervar(2)=containervar(2)+1;
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [containervar,containerstacks, eventlist, tDexp] =
departurerealisation (t,tE,containervar,i,containerstacks, eventlist,
tDexp)
%departure of a container
global moves maximumhoogte
%update the total time spend on the container terminal
containervar(1)=containervar(1)+containervar(2)*(t-tE);
%find the spot at the terminal were the leaving container is stacked.
[rij,kolom]=find(containerstacks==tDexp(i-1));
welofgeencontainer=(containerstacks~=12000);
%if there are containers on top of the leaving container
if sum(welofgeencontainer(1:rij,kolom))>1
    %remove these containers
    [ containerstacks ,moves ] = MoveWithContainersway2( containerstacks
,moves,rij,kolom,maximumhoogte );
    %container itself leaves the terminal
    containerstacks(rij,kolom)=12000;
    moves=moves+1;
    %no containers on top of leaving container
elseif welofgeencontainer(rij,kolom)==1
    %remove leaving container
    containerstacks(rij,kolom)=12000;
    moves=moves+1;
end
%(expected) time this container leaves set to infinity
tDexp(i-1)=inf;
eventlist(i)=inf;
%one container less at the terminal
containervar(2)=containervar(2)-1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```