Master's thesis Economics & Informatics

Evaluation of online container stacking strategies using simulation

Author:
Bram Borgman
289581

Supervisor:
Drs. Eelco van Asperen

Co-reader:
Prof.Dr.Ir. Rommert Dekker

Erasmus School of Economics, Erasmus University Rotterdam
May 8, 2009

Abstract

With the current increase of container traffic volumes around the globe, it becomes ever more important to have efficiently running container terminals. In this thesis, we investigate two concepts with which to increase efficiency, and compare them to several benchmark algorithms, using a Java-based discrete-event simulation tool. The first concept is to use any knowledge known about container departure times, in order to limit the number of reshuffles. We stack containers leaving shortly before the previous on top of each other. The second is the evaluation of the tradeoff between stacking further away in the terminal vs. stacking close by the exit points and accepting more reshuffles. We have formulated several strategies based on these concepts, some of which mix the two. We compare the strategies derived from these concepts primarily by looking at the exiting time of containers, and secondarily by looking at reshuffle occurrences, ground position usage, entry time and crane workload. It is concluded that even the use of imperfect or imprecise departure time information leads to significant improvements in efficiency. Minimizing the difference in departure times proved to be very important. It was also found that the tradeoff between stacking further away in the terminal vs. stacking close by the exit points and accepting more reshuffles leads to improvements over the benchmark.

Contents

List of Figures

List of Tables

Chapter 1

Introduction, objective and methodology

1.1 Introduction

Since the dawn of containerization in the 1950s, this technique of easy transportation in identical lotsizes (viz. containers of fixed size) has revolutionized the shipping world [30]. The cost reductions achieved by containerization have led to huge increases in the volume of international trade.

Containers require specialized infrastructure at the ports they are shipped between. Unloading a 40 feet long container is not an easy task, so several methods are used to do this. For small ports, it is often not economical to have special container handling equipment, so these ports can only serve special ships that have cranes for loading and unloading on deck.

However, placing this kind of equipment on a ship is very expensive, so most major ports nowadays have their own handling equipment, possibly at a dedicated container terminal. These terminals have giant cranes at their quays (quay cranes), which load and unload containers from a ship. At the quay, containers are stored at the terminal, waiting for trucks, trains and other ships to collect them. An example of a container terminal (in this case ECT in Rotterdam) is in figure 3.2.

1.2 Problem definition

With the continuing growth of intercontinental freight transport, which (besides bulk goods) mostly consists of containers shipped across the oceans, the container terminals of the major ports across the globe are getting busier every month. This effect is strengthened by the fact that ever larger container ships sail the seas. Larger ships are more expensive and henceforth their (unproductive) port time is also more expensive. To minimize these costs, shipping lines make less frequent port calls and unload more containers at every port. This puts a large amount of stress on the container terminal, which often must handle thousands of containers in a short amount of time.

Seeing that the increasing strain on the container terminals will eventually lead to serious problems, it is imperative that container terminals be designed and operated as efficiently as possible. One aspect of this is the way containers are stacked after arriving at the terminal, because it can save a lot of time when, if it is needed, a container can be picked up immediately, instead of several containers being stacked on top of it, which first have to be moved elsewhere.

Of course, in order to be able to make the stacking more efficient, it is necessary some of preknowledge of the containers is known. Unfortunately, the order in which containers will be picked up is usually not known in advance. Still, a rough estimate of residence time can usually be made, which could be used to identify which containers should (not) be put on top of each other. Other knowledge of potential use (though we do not use it in this thesis) is that some containers depart with the same ship, meaning that they are interchangeable, because location in the ship is not very important.

Unfortunately, we cannot just stack any container anywhere. There are many different container types (e.g. 20 ft, 40 ft, 45 ft, reefer) and usually only containers of the same type may be stacked on top of each other. Furthermore, special containers such as reefers (refrigerated containers), may only be placed at special locations in the terminal, since they require a power supply. Also, it is important to note that a stack has a finite size and a limited height.

It is possible to use many different heuristics when stacking containers. For instance, when a new container arrives at a stack, we have to decide whether to place it in an empty slot, or put it on top of other containers. Using empty slots makes picking up easier: we can always reach the container, that is, until we put another on top. However, container terminal yard space is limited and eventually this will occur. We thus want to use some more advanced algorithms to determine stacking locations.

1.3 Objective

In this research project, the objective is to discover and analyse the effects of various container stacking policies, and to find better ones.

1.4 Research questions

The main research question for this thesis is as follows: "Which container stacking strategies lead to performance improvements at single ASC-based container terminals?" With the current expansion of global container shipping, more terminals are also being constructed around the world, making this question very relevant. We investigate this particular type of container terminal, because we have some data available on this type of terminals, namely from the ECT Delta Terminal in Rotterdam.

To support the answering of the main question, we also try to answer the following subquestions:

1. "How can the performance of a container stack be measured?"

2. "Which strategies for container stacking can be formulated, using only basic information, given the performance measured used?"

3. "Which strategies for container stacking can be formulated, using residence time information, given the performance measured used?"

4. "How do these container stacking strategies compare?"

1.5 Methodology

In order to answer the questions posed in section 1.4, we use the methodology discussed here.

First of all, a literature review is performed, in order to gain insights in the working of container terminals in general and container stacking specifically and to find out what research has already been done. This is also useful for obtaining ideas for own research. See chapter 2.

Secondly, a simulation model is needed, so experiments can be carried out. The program used in previous research was written in Delphi (Object Pascal) and is not easily extendable. Moreover, parts of the source code are lost. For the simulation model, we use the framework of simulation studies, as proposed by [2] and shown in figure 1.1.

After we defined a conceptual model (step 3, see chapter 3), an implementing program was programmed in Java, with the use of a discrete-event simulation library. Because it is likely that further research is needed in this area, the program and model had to be flexible and well documented, so they can be extended in the future. To assist the programming, preliminary modelling using UML was done first. The program also needs to be working correctly (i.e. verification: does the program do what it is supposed to?), for which testing is needed. For this, a visualization is useful. Also, unit tests and precalculated test scenarios were used to verify the program. See chapter 4 for a description of the simulation program.

In addition to the main simulation model, we also need a (sub)program to generate any containers entering the model. Data from ECT used in previous research ([5]) is still available [33], however, a new program has to be written from scratch. The old data uses variable interarrival times of ships, as well as variable ship sizes (up to 8000 TEU, with a call size of about 3000 containers of various sizes) and a fixed quay crane throughput. The program also has to give all containers a scheduled departure time. The generator program is described in chapter 5.

Thirdly, several experiments with the model are carried out, and their results analysed, so that we may answer the research questions. These experiments should have different setups in such a way, that the influence of each parameter becomes clear. For this, a simple stack is important, because complex stacks, while perhaps more realistic, have a higher risk of influencing the outcomes and hence distort the effects of other parameters. We want to compare the results of the experiments with each other, so we need some performance measures, several of which are suggested by Dekker et al. [5] (see section 3.6). The main experimental setup and the benchmark algorithms are introduced in chapter 7 and the experiments themselves, including results and discussions, are in chapter 8.

Finally, when all experiments are done, we can answer our research questions and draw our

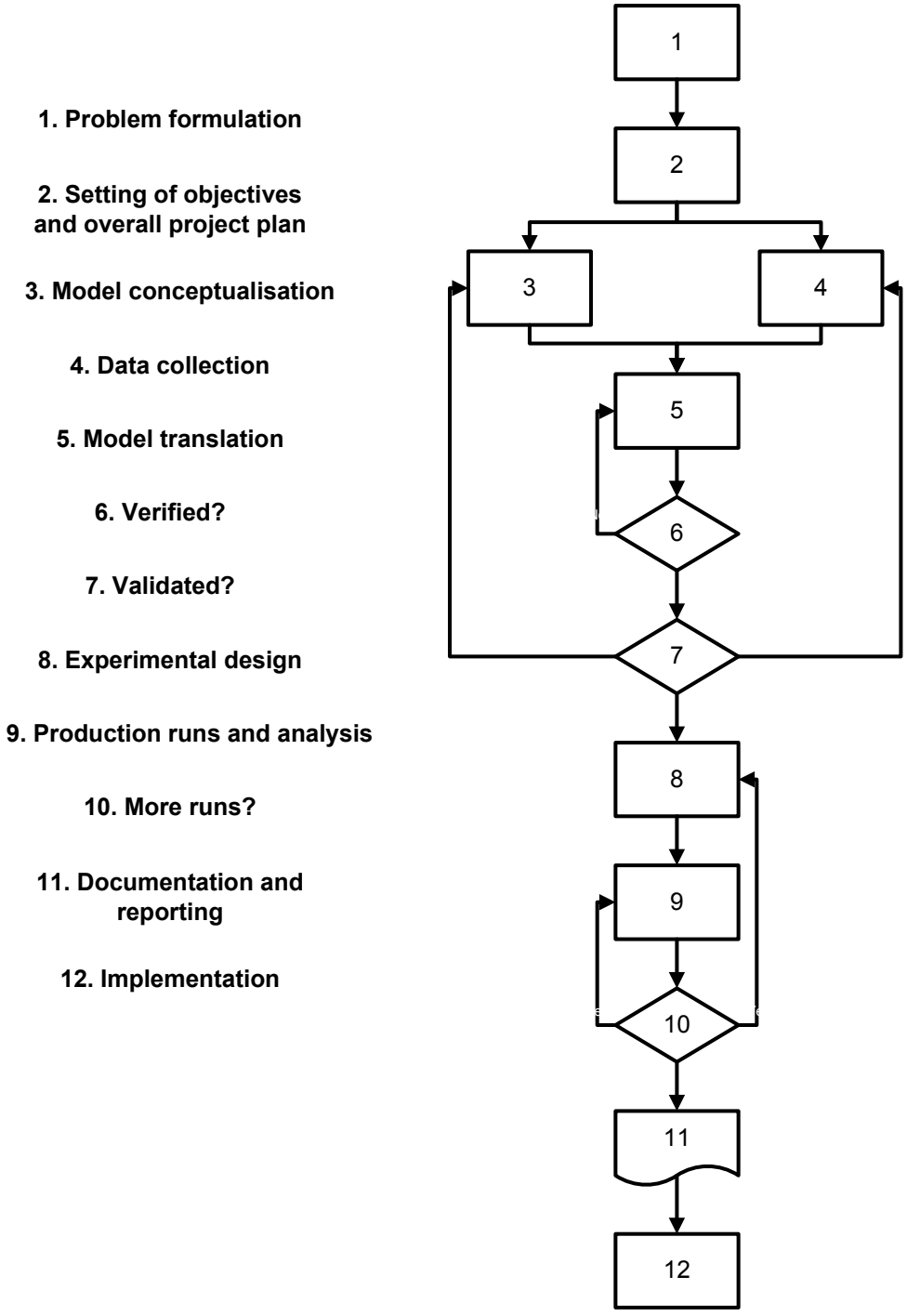

Figure 1.1: The 12-step framework of simulation studies, used in this thesis. Source: [2]

conclusions. This is done in chapter 9, where we also give some suggestions for future avenues of research.

Chapter 2

Literature review

In this chapter, we conduct a review of scientific literature to find out how stacking is usually done at container terminals (section 2.1), as well as to explore the research already performed in this area (section 2.2). This enables us to determine the features the simulation model should have and what the setups of the experiments should be.

2.1 Background on container stacking

This section is mostly based on the article by Dekker et al. [5].

In stacking containers, various decision horizons can be identified. An often used classification has four temporal categories: long term (years, decades), medium-term (months), short-term (days) and real-time (minutes, seconds).

Long term decisions are strategic decisions, e.g. decisions dealing with the location and design of the container terminal. Decisions include the size and locations of a terminal, the type and numer of cranes, the type of stacking (automated/manual, big cranes or small vehicles, stacking height), etc. Higher stacks are more efficient with space than low stacks, but they do require bigger cranes, as well as potentially leading to more reshuffles in a stacks, if the bottom container is needed. It is very important the decisions made for the long-term are as "good" as possible, since mistakes cannot easily be corrected.

Medium-term (or tactical) decisions concern capacity decisions, such as stack layout, number of vehicles used to move containers about and whether or not (and how) to do remarshalling in the yard when no ships are being served.

Short term (or operational) decisions involve the reservation of berthing space for ships, the decision to store a particular container on a particular spot and the allocation of the equipment to the various jobs scheduled in the coming days.

Real-time decisions are decisions made when actually executing whatever part of the stacking process. It includes the speed and direction control of all vehicles, as well as the cranes, and is hence mostly of importance to automated equipment. This thesis focuses primarily on several strategic and operational decisions. In particular, the operational decisions of the stacking

strategy are elaborated on. The main objectives of any stacking strategy are (in no particular order):

1. Efficient use of storage space
2. Efficient and timely transportations from quay to stack and further, and vice versa
3. Avoidance of unproductive moves.

Of course, it differs per case what the importance of every criterion is: ports like Rotterdam and Singapore have limited land space, so they are likely to have much more need of efficiently used storage space. Note also that these objectives are contradictory: you cannot maximize them all. For example, goal #3 would be optimized by having stacks of only one container high; however, this would lead to massively inefficient use of storage space.

Dekker et al. [5] mention two types of stacking strategies: category stacking and residence time stacking. The former strategy assumes that containers of the same category (e.g. having the same destination, weight, etc.) are interchangeable, and can thus be stacked on top of each other without the risk a lower container in a stack is needed before the ones on top of it have been removed. The latter strategy does not use categories, but instead looks at the departure times of the containers: only if a new container is to leave earlier than the containers it is put on top of, it may be stacked at a position.

Steenken et al. [29] distinguish between storage planning and scattered stacking. Storage planning means that before a ship arrives, a plan is made of where to stack all containers that will be (un)loaded. A specific part of the yard is allocated for this. A downside of this, is that the container cranes of specific part of the stack are put under a heavy strain, because much containers need to be moved by them. In scattered stacking, on the other hand, the yard area to be stacked in is only chosen after berthing, and positions for unloaded containers are determined in real-time. Unloaded containers of the same category are stacked in the same area. Of course, some practical limitations still apply when stacking: only containers of the same size and type may be stacked on top of each other. Some containers need dedicated space in the yards, such as reefers (refrigerated containers), which need an electricity supply.

2.2 Prior research

Literature on stacking problems is not very common yet, perhaps because the problem does not easily lend itself for analytical solutions [5]. However, in recent years, the subject seems to get more attention, because its importance is recognized [29]. In a recent overview paper on operations research at container terminals, Stahlbock and Voss looked at a number of aspects of container terminal operations [28]. Among the topics surveyed were stowage planning, berth allocation, crane optimization, terminal transport optimization and, most relevant for this thesis, storage and stacking logistics. Their work is an extension of an earlier survey [29], which also contains a paragraph on how stacking is done in practice.

Various methods are used to tackle the stacking problem, but two main approaches can be distinguished [5]:

Simplified analytical calculations Relatively easy to do (computationally at least), but their usefulness is limited, as they only give insight on a more abstract level into the relationships between parameters.

Detailed simulation studies These studies take more time to perform, but can also give much more detail.

Recent examples of the analytical approach include Kim and Hong [15], who used branch-and-bound to find an optimal solution to a stacking problem and then propose several heuristics to try to come close to the optimum, and Kang et al. [13], who use simulated annealing to find optimal solutions reasonably fast. However, the problem most of these optimization approaches have, is that they assume perfect knowledge of the order of picking up the containers. However, this information is not known in advance usually. Still, finding a theoretical optimum is useful for comparisons. Other methods previously used for this include Q-Learning [11] and critical-shaking neighborhood search [18]. Han et al. [10] use integer programming with tabu search to generate an entire yard template, which should minimise remarshalling moves. This approach is out of the scope of this thesis however, also because it is expensive to use it in existing container terminals (as they might need to be extensively remodelled). Froyland et al. [9] also use integer programming, optimizing the entire terminal in an effort to maximize quay crane performance.

More detailed simulations were performed by Dekker et al. [5]. They simulated different stacking policies for containers in automated terminals. In particular, several variants of category stacking (with up to 90 different categories) were examined and compared with a base case in which containers are stacked randomly. The simulated terminal has a theoretical yard area of 19440 TEU, with 27 lanes of 6 containers wide and 40 containers long. Containers can be stacked up to 3 high. The performance of every strategy is measured using 4 groups of statistics: reshuffles, no positions available, workload of stacking cranes and occupation.

Earlier research by Duinkerken et al. [7] also used simulation and category stacking, albeit with only a limited number of categories. Several (reactive) remarshalling rules were tested. Reactive remarshalling means the remarshalling done when a container on which other containers are stacked is demanded for retrieval, leading to a number of remarshalling operations. This in contrast to "proactive" remarshalling, which is done when stacking cranes are idle. They tested the following remarshalling rules:

Random Select a random possible new location.

Level Fill the stack level by level, starting at every (quayside) transfer point and then on until all ground positions are full, after which level 2 is filled.

Closest position Select the position closest to the current position, which is not full and has no containers in it scheduled to leave earlier (a form of residence time stacking).

Minimising RSC (Remaining Stack Capacity) reduction RSC is a measure of the capacity of a pile. The RSC of a stacking position (i.e. pile with zero or more containers) ij (row/lane i, position j) is given as follows:

$$RSC_{ij} = (H - h_{ij}) * c_{ij}$$

where
H = maximum stack height
h_{ij} = actual pile height
c_{ij} = category of top container of this pile or C if empty
C = total number of categories.

Because more RSC keeps more options open, and every new container reduces it, the reduction should be minimised. Therefore, a new container of category c_2 is added, the following problem has to be solved:

$$\min[(H - h_{ij}) * (c_{ij} - c_2) + c_2] \text{ for all } (i, j) \text{ with } h_{ij} < H$$

The RSC formula is only used on those piles where a container with a higher or equal category is on top (or empty piles). If no such pile can be found, the container is stacked upon another pile, "in such a way that the increase in RSC remains minimal".

Duinkerken et al. compared the use of categories with a model that required specific containers and found that the categories lead to much better performance. Also, it was shown that the RSC strategy lead to big improvements when compared to the other three. However, in this model, performance was chiefly measured by quay crane utilisation, which in our model is not included. Duinkerken et al. also tested two "normal" stacking strategies (i.e. for when a container has just arrived), namely random and with dedicated lanes for a quay crane. However, using dedicated lanes is hard to do in practice, as load plans are not known in advance. Also, this strategy did not yield much improvement over random stacking.

Weng [36] used a simpler simulation model to test a few stacking rules using residence times. A neural network is used to estimate the residence time, which proved to be a useful thing to know.

Saanen and Dekker [23, 24] went into great detail in simulating a (transshipment) container terminal with rubber-tired gantry cranes (RTG), carefully simulating all movements of the trucks and cranes. They made a "comparison between a refined, but still traditional, strategy for operating a transshipment RTG terminal with a simple random stacking strategy for this type of terminal", and measured the differences in quay crane productivity (in lifts per hour), as this is considered the most important indicator for terminal efficiency. They found that the differences between the strategies was very small (about 0.7 lifts per hour). However, it was found that the number of gantry movements is a major factor in limiting the quay crane productivity.

Park et al. [20] used simulation to determine the best combination of an AGV dispatching rule with a reactive remarshalling rule, for various amounts of containers and AGVs in an automated container terminal. Their goal is to minimise the number of remarshalling operations.

Park et al. used the *random*, *closest position* (but with residence time taken into account) and *minimal RSC reduction* remarshalling rules from Duinkerken et al. [7]. In most cases, the minimal RSC reduction rule, combined with the *Container Crane Balancing (CCB)* dispatching rule (the AGV is sent to the crane which has the most containers waiting for it), lead to the least remarshalling operations.

2.3 Final remarks

In this chapter, we attempted to give a comprehensive overview of the techniques used for container stacking research and the research done with them. While it is clear that the field of container stacking is actively researched, and hence is interesting, there are not many studies done with detailed simulation models, because of their complexity. However, because the detail of simulation models could lead to more realistic models, it is very useful to use them, so that we can gain more insight in useful stacking strategies.

Chapter 3

Conceptual model

In this chapter, we present a conceptual model for the simulation. The conceptual model is a part (step 3) of the simulation studies framework used in this thesis (see figure 1.1). Parts of this chapter are based on previous research by the same author [3].

3.1 Introduction

Robinson [21] gives the following definition of a conceptual model:

> *the conceptual model is a non-software specific description of the simulation model that is to be developed, describing the objectives, inputs, outputs, content, assumptions and simplifications of the model.*

Conceptual modelling has seen little research outside military applications, according to Robinson [22], who also states that "it is generally recognized that conceptual modeling is one of the most vital parts of a simulation study. At the same time, it also seems to be one of the least understood", so it is clear there is a need to examine some ways of conceptual modelling.

Wang and Brooks [34] investigated conceptual modelling in practice, and found that experts and novices in simulation use significantly different approaches. Hence, there probably is something to work on here, as not every approach is good. This makes it even more important we investigate good ways to do conceptual modelling.

Robinson gives a number of sources on how a conceptual model should be created, but few of them are very clear and explicit about it. Two of the few that do give an explicit step-wise methodology of creating a conceptual model are Shannon [27] and Pace [19]. We try to follow their four step methodologies in creating our own conceptual model. However, as we shall see, these methodologies do not cover everything one would like to see in a conceptual model.

Shannon's methodology [27] dates back from 1975 and comprises four steps:

1. specification of the model's purpose

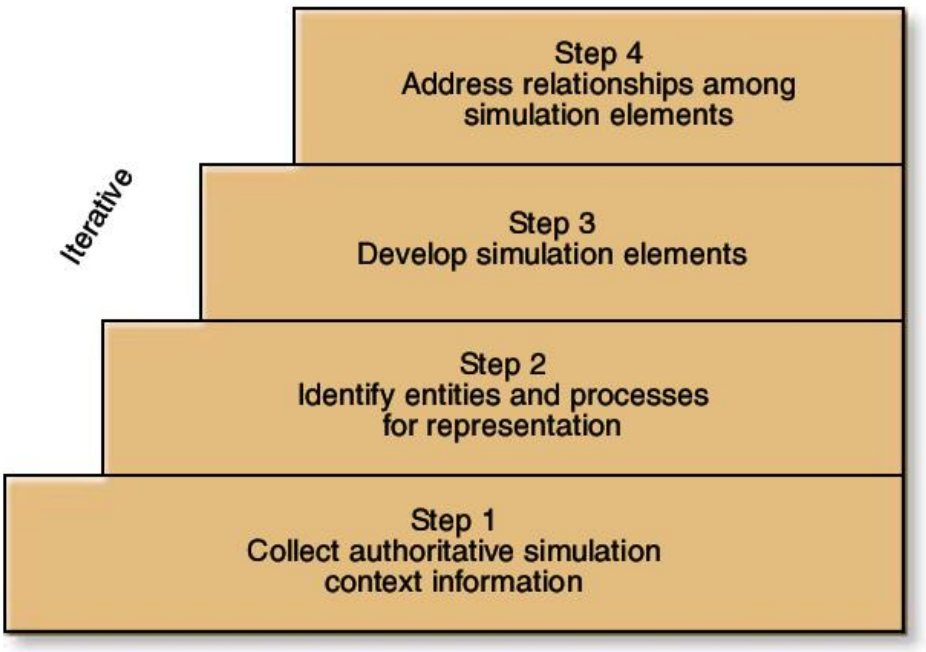

Figure 3.1: Four-step methodology proposed by Pace to make a conceptual model for a simulation. Source: [19]

2. specification of the model's components
3. specification of parameters and variables associated with the components
4. specification of the relations between the components, parameters and variables.

Pace's methodology [19] (see also figure 3.1) is based on Shannon's. Since it was originally intended for use in military applications, some of his methodology is not applicable in our field. We therefore only show the main four-step methodology he uses, and not the more detailed work:

1. collect authoritative information on the problem domain
2. identify entities and processes that need to be represented
3. identify simulation elements (for each entity/process)
4. identify relationships between the simulation elements

These four steps will usually be iterated often.

Furthermore, on conceptual model design, Kelton et al. say:

> *What level of detail is appropriate? What needs to be modeled carefully and what can be dealt with in a fairly crude, high-level manner? Get buy-ins to the modeling assumptions from management and those in decision-making positions.* (Page 42 of [14].)

While Pace's and Shannon's models are very similar, Pace's is less abstract, which is why we primarily use this method. However, Kelton [14] and Robinson [21] have pointed out that assumptions must also be made in the conceptual model, which is why we also include these, despite neither Pace nor Shannon mentioning them. We also look at input and output modelling, the latter in terms of which statistics are needed, as Robinson mentioned and Kelton also says "Specificity about what is to be observed, manipulated, changed and altered is essential" [14]. Note that we thus do not only perform step 3 of the simulation studies framework of figure 1.1, but also (a part of) step 2, namely model purpose.

The chapter is therefore divided into five sections: model purpose (see section 3.2), assumptions (see section 3.3), entities, resources and processes (see section 3.4), input requirements (see section 3.5) and model output (see section 3.6).

3.2 Purpose

As stated in section 1.3, we want our simulation model to allow us to discover and analyse the effects of various container stacking policies, and to find better ones. Hence, we need to have a model that allows for different stacking rules.

As we want the model to be scalable and expandable, it also needs to be able to simulate a number of lanes for containers to be stacked in, as well as their cranes. Quay cranes, however, and other terminal operations not directly in contact with the stacks are not needed. The model should also allow for stochasticity, e.g. in the interarrival times of container ships.

To gain any insights at all, it is imperative that some statistics and other data are generated by the tool.

3.3 Assumptions

As in any model, we have to make some assumptions in order for things to be workable. A list of made assumptions is listed below:

1. Terminal design: since we use data from ECT (terminal shown in figure 3.2), it makes sense to model the terminal to generally look like theirs. This also is useful for comparisons with the earlier work of Dekker et al.[5]. The terminal layout used is shown in figure 3.3. Hence, we use Automatic Guided Vehicles (AGV) to transport containers from quay to stack, and Automatic Stacking Cranes (ASC) to stack containers in a stack (one ASC per stack is used). Note that containers are stacked *along* the lane; i.e. a 45ft container is next to 45 feet of lane and not 8 feet (which is the width of all containers).

2. Quay cranes: quay cranes are not explicitly modelled. Their throughput rate is kept constant whenever they are servicing a ship. The number of quay cranes used per ship may be varied.

3. Stack layout: also in figure 3.3 is the layout of a normal stack. We use this as a reference and starting point, however, it would be interesting to try out different stack layouts.

4. We only simulate "standard" container types of 20, 40 and 45 feet, as well as reefers of different sizes.
5. Twenty-foot containers occupy one TEU location (Ground position) in the stack, 40-foot containers occupy two and 45-foot containers occupy three.
6. Reefer containers need to be stacked in positions adjacent to a reefer platform. This platform is one TEU long.
7. Differently sized containers may not be stacked on top of one another.
8. Containers are stacked precisely on top of the other.
9. Reefer containers are not mixed with normal containers within a pile.
10. In case a container that has not been stacked is demanded, the container is stacked anyway and scheduled to leave later. This should be a very rare case.
11. There are no delays and operational failures in the model.
12. Stochasticity: the only stochasticity in the program exists in the stacking strategies and in the arrivals and departures of containers. No stochasticity is included in the main model.
13. Because containers are easiest to measure in imperial feet, all distances in the simulation are in feet. Speeds are in feet per hour.
14. Containers, which have not been stacked at their location and are to be remarshalled already, are instead moved directly to their new position, unless the operation of lowering them onto the pile has already started, in which case they will be moved later.
15. Containers are always remarshalled to a position in their own lane. In case no such position is available, the containers are temporarily moved to the Transfer Point Landside, until the underlying container(s) is(are) removed, after which it is immediately (i.e. with priority over all other tasks) put back in its original pile.

3.4 Entities, resources and processes

In this section, we present the entities, resources and processes to be simulated in the model of our container terminal stack, when we look at it from the viewpoint of the purpose as defined in section 3.2. We also explain what is precisely meant by these terms. After this, we describe the parameters and variables (called *elements* by Pace and *attributes* by Kelton) associated with the entities and resources, and after that we discuss how these are related.

3.4.1 Entities

According to Kelton et al. [14], an *entity* is a "player" in the simulation, that moves around, changes its status, affects the output, interacts with other entities, etc. Entities are *dynamic*, meaning they are created at one time, then move around for a while and are then disposed of.

Figure 3.2: Aerial overview of the ECT terminal in the port of Rotterdam. Source: [5]

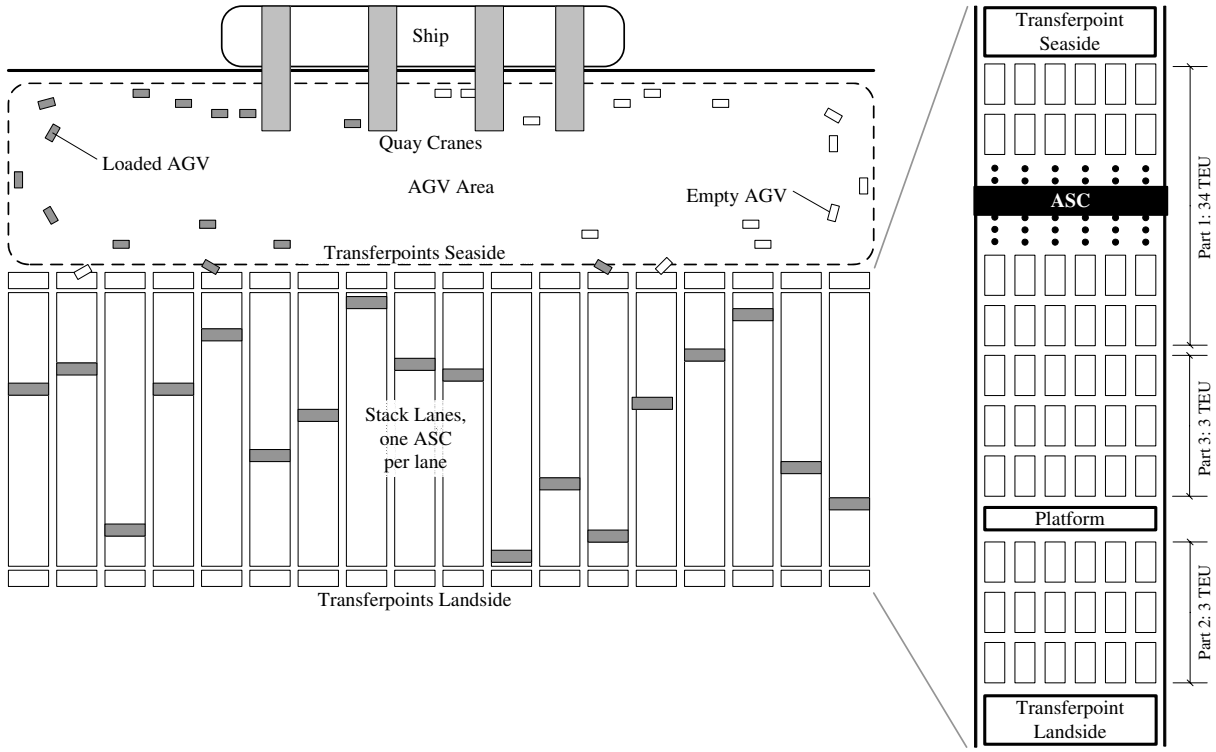

Figure 3.3: Schematic overview of a container terminal and a stack. Source: [5]

In our model, there is only one type of entity:

- `Container` A container to be stacked, comes in various sizes: 20 feet, 40 feet and 45 feet, as well as reefers of these sizes.

3.4.2 Resources

By *resources*, we mean the other "objects" in the simulation that do things, but that are not entities. Resources are *static*, unlike entities. Resources, therefore, are usually present at the beginning of a simulation and the same ones are usually still present at the end.

The following types of resources exist in the model:

- `Automatic Guided Vehicle (AGV)` A vehicle capable of moving a single container between quay and stacking lane. AGVs need to be serviced by a crane and cannot load or unload themselves.
- `Straddle Carrier` A vehicle capable of moving a single container between stacking lane and truck loading point. Straddle carriers can unload and load containers by themselves and do not need a crane for this.
- `Automated Stacking Crane (ASC)` A crane capable of lifting containers on/off a lane and put them on an AGV. There is one ASC per stacking lane.
- `Ground position` Places where containers may be stored. Each location can store one Pile of 20 feet containers (or other types, see assumption no. 5 in section 3.3).
- `Pile` A pile of some containers stacked on top of each other. Every Pile takes up 1 or more Ground positions (see assumption no. 5 in section 3.3 for the number of used Ground positions). Piles only exist when they have containers in them, so their existence is variable.

3.4.3 Processes

By a *process*, we mean the actions undertaken by the resources, affecting the entities. A process takes some time to complete, and in the mean time the resources the process affects cannot do something else. Processes can occur at both high and low levels. On a low level, the precise actions of the individual resources are studied, while high-level processes are done by a combination of multiple resources and low-level processes.

Low-level processes Our stack model contains the following low-level processes:

- Quayside loading/unloading: putting containers on AGVs and taking them off them by a Quay Crane.
- Travelling: moving of AGVs and Straddle Carriers (with or without containers on them) through the terminal, e.g. from quay crane to stacking lane.

- Transferpoint seaside loading/unloading: putting containers on AGV's and taking them off them by an ASC at the "transferpoint seaside" (see figure 3.3).
- ASC: movement along its lane.
- Picking up/setting down: ASCs moving containers within a lane.
- Waiting: sometimes a resource might be idle, not having anything to do.

High-level processes Our stack model contains the following high-level processes:

- Moving a new container into the yard
- Moving a container out of the yard
- Remarshalling

Because high-level processes are more complicated than low-level processes, we describe each of these high-level processes in detail.

Moving a new container into the yard In this process, a container first arrives at an arrival location, i.e. either at the quay or at the truck loading point. In case the container arrived at the truck loading point, a straddle carrier is given a task to come pick it up. If no straddle carrier is available, the container is kept at the truck loading point until a straddle carrier is free to pick it up. The container is then carried to the lane the container was assigned to and the straddle carrier leaves the container at the transfer point landside. After setting down the container, the straddle carrier is free to start another task. At the transfer point, the lane's ASC picks up the container when the associated task has been started and the ASC has moved to the transfer point. The ASC then moves to the position in the lane where the container was assigned to and puts the container on top of the pile there.

In case of a container arriving at the quay, the container is put on top of an AGV. As with straddle carriers, at times it may be possible that no AGV is available and the container has to be kept at the quay for a while. When the AGV is loaded it moves to the assigned lane, where it waits for the lane's ASC to pickup the container. Because the ASC can have other tasks, this might take some time. When the container has been picked up, the AGV is freed and can start a new task. The ASC moves to the correct lane position and puts the container on top of the pile there.

Moving a container out of the yard This process largely works opposite to moving a container into the yard. When a container has to move out and is on top of its pile, the lane's ASC moves to the container's pile to pick it up. (If a container is not on top, remarshalling must occur first, see below.) After the pickup, the ASC moves to a transfer point. In case the container is demanded by a truck, the ASC heads for the transfer point landside, where it put down the container. The ASC is then free to do a new task. Meanwhile, a straddle carrier is sent to pickup the container and move it to the truck loading point, where it leaves the model.

The straddle carrier is then freed to perform a new task. If no straddle carriers are available, containers may have to wait at the transfer point landside.

In case a container is demanded at the quay, the ASC moves to the transfer point quayside, where it waits for an AGV to place the container on. The AGV may be waiting for the crane already. Only after putting the container on the AGV is the ASC freed to perform its next task. After being loaded, the AGV moves to the quay where the container leaves the model and the AGV is freed.

Remarshalling Remarshalling is the process of moving containers between piles. In the model, this is only done within the same lane. This is done to simplify matters and because the additional work of having to use a straddle carrier and a second ASC would likely not often be beneficial.

Remarshalling is needed when a container is scheduled to leave, but is not at the top of its pile. Additionally, it may be decided by an algorithm to do remarshalling when the ASC has no other tasks, in order to improve the stack's efficiency.

When the remarshalling process occurs, the ASC moves to the location of the container and picks it up. The ASC then moves to the container's new location and sets it down on the pile there. In case no piles can be found to stack the container, it is temporarily moved to the transfer point landside. When the container scheduled for departure has been removed, the containers temporarily stacked at the transfer point are put back with the highest priority.

3.4.4 Parameters and variables

Next, we must determine the parameters and variables associated with these entities and resources. A container has the following variables:

- Category it belongs to
- Size (20 ft, 40 ft, 45 ft, reefer)
- Expected and real residence/departure time

AGVs and straddle carriers have the following variables:

- Moving speed (in ft/h)

An ASC has the following variables:

- Moving speed (in ft/h) (lengthwise / widthwise)
- Lifting speed of containers (empty / full)

A Stacking location has the following variables:

- Capable of storing reefers?
- Maximum stacking height

3.4.5 Relations between entities, resources, processes and variables

The next step is to define the relations between all entities, resources, processes and variables (step 4 in figure 3.1).

AGVs and Straddle Carriers are related to a Container, namely the one they are carrying at one time (if any).

An ASC is related to a Container, namely the one it is carrying at one time (if any). It is also linked to some Ground positions, namely the ones in its stacking lane.

A Ground position is related to the Pile stacked on it (if any).

A Pile is related to its contents: a number of containers.

3.5 Input requirements

Any output generated by the simulation will only be useful if correct data is given as inputs (the "garbage in, garbage out" principle [14]). The inputs of the model can be given in a plain text file. The following data must be provided in order for the simulation to run:

- Stack layout:
  - Number, widths and lengths of stacking lanes (also: special reefer piles)
  - Maximum stacking height
  - Distances between the transfer points (i.e. how long do AGVs and ASCs need to travel?)
- Container flows:
  - Used container categories and corresponding frequencies of occurrence
  - Ship interarrival times
  - Quay crane production speed
  - Residence times (real and "expected" - i.e. what is known to the simulation)
- Number and speed of AGVs and Straddle Carriers
- Speeds of ASCs
- Used stacking policy

Some of these inputs require a look at real data, or otherwise a reasonable assumption.

3.6 Output modelling: statistics

Our simulation model should be able to calculate some useful statistics about every experiment, so that they may be compared to others and we can make an assessment of the degree in which the three criteria mentioned in section 2.1 (originally by [5]) are met. It would be useful to generate statistics also used in previous research, so as to perhaps make a comparison. For starters, [5] use the following performance measures:

Reshuffles

- Number of occasions of reshuffling.
- Number of containers reshuffled.

No positions available

- Number of times no suitable (i.e. a stack was assigned, but there was no room) position was found for a container.

Workload of automated stacking cranes

- Average workload per ASC (in percentage of time the ASC is busy).

Occupation

- Number of used ground locations (i.e. at least one container stacked).
- Number of used stack positions (i.e. percentage a lane is full (of the maximum number of TEUs)).

Some other measures we could use include:

Waiting times

- Time ASCs had to wait for AGVs to service.
- Time ASCs had to wait for straddle carriers to deliver their container.
- Time AGVs had to wait for ASCs to service them.
- Time straddle carriers had to wait for ASCs to deliver their container.

Container dwell times

- Time containers are in the system before they are stacked (entering time)
- Time containers are in the system after they should have left (exiting time)

For all these statistics, we are of course interested in their means (average), but also in indications of their reliability, e.g. maximum, minimum, standard deviation and confidence intervals.

Chapter 4

Description of the simulation tool

In this chapter, we describe the implementation of the tool (called "SSTACk" (short for "Stacking Simulation Tool for Automated Container terminals") in internal development), which uses Java and the SSJ library. We refer to appendix A for UML class diagrams, which are useful for a better understanding of the relations between and functions of the classes in the object-oriented model, as these diagrams are too large to be included in this chapter. The tool was developed using Java and the SSJ (Stochastic Simulation in Java) library, because the author has used this library before and it is easy to use. In this previous application SSJ performed well [4]. Some alternatives were not recommended for this kind of topic, such as Arena [12]. For details on SSJ see [17] and [16]. SSJ is free software and is released under the GNU General Public License. Parts of the code were taken from earlier projects of the author, in particular the seminar Simulation 2007 [4] and a bachelor's thesis [3]. This section is a part (step 5) of the simulation studies framework used in this thesis (see figure 1.1).

The simulation's main class is called `Experiment`, which contains references to all used objects and starts the simulation. It is in turn created by a Client class.

4.1 Entities and resources

In section 3.4, we discussed which entities and resources the model should contain. Therefore, the simulation tool uses the following entities and resources to operate (to show that some resources are contained in others, they were tabbed extra where applicable).

- `Container` A 20, 40 or 45 foot container, can be a reefer; stacked in and moved through the terminal.
- `Terminal` Main class of the terminal. Contains one or more Quays, Truck Loading Points and Stacks.
  - `Quay extends`[1] `ArrivalLocation` Represents the quay where containers are taken from ships or put onto them.

[1] `Extends` is the Java term to describe inheritance, thus we mean here that `Quay` and `TruckLoadingPoint` are different, but share some common characteristics (defined in the class `ArrivalLocation`).

  - `TruckLoadingPoint extends`[1] `ArrivalLocation` Represents the facility where Straddle Carriers load or unload trucks.
  - `TStack` A collection of Stacking Lanes. Served by a number of AGVs and Straddle Carriers.
    * `StraddleCarrier` A straddle carrier; transports Containers from the Truck Loading Point to the Stacking Lanes and vice versa.
    * `AGV` An Automatically Guided Vehicle; transports Containers between the Quay and the Stacking Lanes and vice versa. At the lane, it needs to be served by the lane's ASC.
    * `StackingLane` A number of rows with Ground Positions, served by a single ASC.
      · `ASC` An Automated Stacking Crane. Moves Containers through a Stacking Lane.
      · `TPLandside` Transfer Point Landside. Straddle Carriers leave and pickup containers here.
      · `TPQuayside` Transfer Point Quayside. AGVs are served here by the ASC.
      · `GroundPosition` A 20 feet long (and one container wide) area of flat land in a Stacking Lane. Ground positions can be occupied by a Pile with Containers or be empty.

- `Pile` A pile of containers. Always contains at least one PileSlot. Contains only containers of the same size and type (i.e. reefer or not reefer).
  - `PileSlot` A slot (i.e. level) of a pile. Contains a container and information about any stack locks (see section 4.1.1) present.

The following subsections provide more details on how these entities and resources are modelled.

4.1.1 Ground positions, reefer platforms, piles and stack locks

A stacking lane is divided into several ground positions. Ground positions have the size of 1 TEU container: 20 feet long and 8 feet wide. Because lanes are usually much longer than they are wide and containers are stacked along the lane, a stacking lane normally has only a few ground positions next to each other (with their long side), and much more with their short side.

Besides ground positions for stacking, a position can also be occupied by a reefer platform. These platforms are (in the model) also one TEU in size. The locations of reefer platforms can be set as a parameter of the simulation. The program then automatically sets the `hasReeferPowerSupply` variable ground positions have. This variable, if set to `true`, indicates a power supply is available, so that reefers may be stacked here.

A ground position may be empty or be part of a Pile. A pile is a stack of at least 1 pile slot and at most whatever number the specified maximum height is. Piles contain only one size of containers, and can thus use a number of ground positions. As stated previously in the assumptions (see section 3.3), a pile with 20-feet containers uses one ground position, a pile with 40-feet containers uses two adjacent ground positions and a pile with 45-feet containers uses three ground positions. Piles never overlap.

Piles consist of pile slots, which contain information on whatever is present in that slot. In particular, the pile slot contains a container (which may or may not be already stacked - see below), the type of stack lock present (if any) and the (expected) unlock time (-1 if no lock is present).

There are two (or actually, three, see below) types of stack locks: incoming and outgoing. These are very different. An incoming stack lock means that the container has not arrived yet, but is underway, so the slot must be kept free. The unlock time is the expected time the container will be stacked. An outgoing stack lock means the container is present, but will be removed shortly. The unlock time is the expected time the container (and with it, the slot) will be removed from the pile.

It is possible for any lock to be both incoming and outgoing (see assumption 10 in section 3.3). This is made clear with a special *through*-lock, which is changed to an outgoing lock when the container is stacked. These cases should be very rare, but the program must be able to deal with them. Another case in which through locks are used, is when a container is temporarily remarshalled to the transfer point landside. It's old position is filled with a through-lock until the crane has returned it.

Stack locks are important for the model, since they provide the stacking algorithm with information about what will happen to piles and containers in the stack. For example, a stacking algorithm would want to know if a container is about to be stacked at a certain position (incoming lock), because this position cannot be used by another container. Conversely, when an outgoing lock is present, it might not be a good idea to stack additional containers on top of it, as these would likely have to be remarshalled, leading to extra work.

The simulation program enforces the locks w.r.t. stacking on top of them as follows:

- If a pile contains any *outgoing* or *through* lock, no container may be stacked on it, except when an incoming lock for it already exists. This means no new incoming locks may be put on the pile until the outgoing containers have all gone.
- If a pile has an *incoming* lock on top, new incoming locks may be put on top of it. This presents no problem, since containers arrive in the lane on a first come, first serve basis.
- Because remarshalling has a priority above normal tasks, containers that are to be remarshalled may not be put on top of existing incoming locks.
- These rules are superseded by normal stacking rules, such as maximum height and uniform size and type.

4.1.2 ASCs

ASCs are the most complex resources in the simulation. This is mainly because they have a big influence on the results of the simulations, which means it is important to model them in detail, more so than AGVs for example. However, piles, for example, are also of great influence. Piles, however, do not move, i.e. they are not very active. These activities of an ASC make it

ID: 4	finished = `false`	startTime = 4.5
Task	Target location	Target container
`MoveSubtask`	`TPQuayside`	
`HoistSubtask`	Pile `null`	Container #2
`MoveSubtask`	`NormalLanePosition`	
`HoistSubtask`	Pile #4	Container #2

Table 4.1: Example of an ASCTask: a task moving a container #2 from the transfer point quayside to pile #4.

very complex, the ASC has many interactions with other entities and many dependencies in its relations exist.

ASCs pick up containers from the Transfer Point Landside (where a straddle carrier put them previously) or from a waiting AGV at the Transfer Point Quayside. The ASC then moves these containers to their specified position in the stacking lane. In case of outbound containers (i.e. containers to be picked up by ships or trucks), the ASC picks up containers from the stack and takes them to the Transfer Point Quayside or to a waiting AGV at the Transfer Point Landside. Alternatively, the ASC may move containers between locations in the lane, a process known as remarshalling. This is particulary necessary when a container, which is not at the top of its pile, has to be moved to a Transfer Point.

Every ASC keeps a task list, which is modelled as a Queue. The tasks in the queue are executed in the order of arrival. Every `ASCTask` is made up of several `ASCSubtasks`. There are two types of subtasks: move-subtasks and hoist-subtasks. Move-subtasks tell the crane to move to a certain position in the lane, while hoist-subtasks tell the crane to lift a specific container (which they have a reference to). ASCs are not interrupted in their tasks: when a task is started it has to be finished before a new one is started. It is, however, possible to change an `ASCTask` while it is being executed. This is done when a container, which has not been stacked yet, receives a new location. For an example of an `ASCTask`, see table 4.1.

An ASC can be either in `RAISED` or `LOWERED` position. Raising and lowering takes a fixed amount of time. An ASC can only move in raised state, and has to be lowered to set down or pick up a container.

4.1.3 AGVs and Straddle carriers

AGVs and Straddle carriers serve the purpose of transporting containers from the transfer points at every end of every stacking lane to the quays and truck loading points (and vice versa), respectively. Straddle carriers are straightforward to model, since they just pick up and drop off containers without any involvement needed by the ASC. If a container to be picked up is not yet at the landside transfer point, the straddle carrier waits at the transfer point until it has been delivered by the ASC.

AGVs are a little more complex to model, since they require the ASC of a lane to pick up the container they are carrying, or alternatively to put a container on them. If an AGV arrives at the quayside transfer point, it thus waits for the ASC, unless it is already there as well. Likewise,

the ASC might have to wait at the transfer point for the AGV.

To coordinate the various tasks for the AGVs and straddle carriers, two task managing classes exist: `AGVTaskManager` and `SCTaskManager`. These classes keep a list of tasks to be done and assign them to vehicles whenever they have finished a task, or immediately to the first waiting vehicle if there were any idle vehicles.

4.2 Event-driven aspects

4.2.1 Theoretical background to discrete-event simulation

Although we decided to use discrete-event simulation for our research fairly early on, we only now explain the meaning of this term, because only now does this thesis commit itself to discrete-event simulation. We give only a very short introduction here, for a more in-depth explanation we refer to chapters 1 and 2 of [14].

(Computer) simulation refers to methods designed to imitate a real world system's behaviour, in order to determine how the real system behaves under different circumstances. Simulations can be either discrete or continuous, discrete meaning changes can only occur at certain moments in time, whereas continuous simulations change all the time. We use discrete simulation, because containers arrive (and are demanded) at fixed points in time, and other parts of the model should only then act accordingly. These marked points in time could be represented as events, leading us to discrete-event simulation. In discrete-event simulation, the model only changes when an event is encountered. In the mean time, nothing happens, except for time progression. Note that although the movent of (for example) cranes is continuous when it's happening, we model it using discrete events: an event is scheduled to occur at the calculated time the crane (or AGV or Straddle Carrier) should arrive at its destination.

4.3 Time, schedules and delays

4.3.1 Simulation time

We have chosen to have the hour as the base time unit in the simulation. Hence, if a container is to be picked up 10 hours after its arrival in real life, it takes 10 time units in the simulation. Likewise, if a crane movement takes 36 seconds in real life, it takes 0.01 time units in the simulation. The simulation runs for a predetermined amount of time, but collection of statistics may be turned off until a sort of "steady state" is reached; the period before this is called the warming-up time, and is used to make sure the initial absence of containers does not affect the statistics. A special Event is scheduled in the simulation to facilitate this. This `WarmingUpFinishedEvent` re-initialises all Tallies and Accumulates and also resets all other counters.

4.3.2 Schedules and tasks

As previously mentioned in section 3.4, ASCs, as well as AGVs and Straddle Carriers, have tasks assigned to them. Fixed schedules as such do not exist in the main simulation model, but may be created for arriving ships and trucks by the user.

Tasks are normally executed in the order in which they were received by the ASC or the `AGVTaskManager/SCTaskManager`. (...)

4.3.3 Delays

As stated previously in the assumptions (see section 3.3), no delays are modelled, because the purpose of this model is to find the best stacking rules in theory. Delays are more suited for operational models, where the impact of a delay is very important and may be major.

4.4 Input data

In section 3.5 we described which inputs are needed to conduct experiments with our simulation tool. These inputs can be roughly divided into three groups:

- Container arrivals and departures
- Stack layout
- Simulation parameters

We discuss these three groups in more detail below.

4.4.1 Container arrivals and departures

It is not trivial to come up with realistic artificial arrival schedules for containers in a terminal. Containers do not arrive at random, but usually in groups, when a ship has docked. Ships are usually sailing according to a fixed schedule, which means the same ships visit the terminal on regular intervals. Also, ships are often loaded with the same type of containers in the same holds.

We thus might say that container flows are rather predictable. However, truck arrivals are far more random and less easy to predict.

Because of these difficulties, it was decided to separate the container arrivals from the main program. A separate program has to make a list (a plain text file), either based on available data or a complete fantasy setup, which can then be used as an input for the simulation. The simulation reads lists with a number of rows, where every row represents an arrival. Rows/lines starting with a '#' are ignored and can be used for comments. All other rows have to contain eleven values, separated by a semicolon (';'). These eleven values are:

1. **Arrival time.** A floating point real number (Java: `double`), stating the time of this arrival.
2. **Arrival mode.** Choose from: *jumbo*, *ds*, *ss*, *barge*, *rail* and *truck*, depending on the arrival mode of the container.
3. **Size.** Integer number, length (in feet) of this container. Should be 20, 40 or 45.
4. **Category.** Integer number representing the category of this container.
5. **isReefer?** Either "true" or "false", depending on whether or not the arriving container is a reefer.
6. **Actual departure time.** A floating point real number (Java: `double`), stating the time this container will be called for at the departure location.
7. **Departure mode.** Choose from: *jumbo*, *ds*, *ss*, *barge*, *rail* and *truck*, depending on the departure mode of the container.
8. **Departure batch number.** Integer number, stating the ship or train this container departs with.
9. **Expected departure time.** A floating point real number (Java: `double`), stating the time this container is expected to be called for at the departure location. This information may be used by the simulation to improve stacking strategies.
10. **Expected arrival time.** A floating point real number (Java: `double`), stating the time this container is expected to arrive at the quay or truck loading point. This information may be used by the simulation to improve stacking strategies.
11. **Arrival batch number.** Integer number, stating the ship or train this container arrives with.

The arrivals text file is processed by the class `ArrivalReader`, which takes out any comments and puts all lines in a `LinkedList`. Every time the simulation encounters a `ListContainerArrivalEvent`, it removes the first line from the list and creates a container based on the information in it. After this, a new `ListContainerArrivalEvent` is scheduled (to occur at the time of the next arrival time), as long as there are more lines to process. It also makes sure the container enters the simulation, as well as creating an event so the container will be picked up. For the experiments done in this thesis, a generator program was written, which generates files containing arrivals. This generator is described and discussed in chapter 5.

4.4.2 Stack layout

In this category, everything related to transporters and layout of the stack is included. For AGVs and Straddle Carriers, the following parameters have to be set:

- Number of AGVs/SCs

- Fixed travel time
- Fixed quay/tlp loading time
- Fixed quay/tlp unloading time

ASCs have these parameters:

- Lengthwise speed (in feet per hour)
- Widthwise speed
- Lifting speed (both empty and loaded)
- Acceleration for every movement type
- Fixed lane pickup time
- Fixed lane setdown time
- Fixed AGV service time (pickup)
- Fixed AGV service time (setdown)

Parameters related to the sizes and number of lanes are as follows:

- Number of lanes
- Lane length
- Lane width
- Maximum stacking height
- Locations of reefer platforms
- Start and end of the dedicated reefer section in every lane

The dedicated reefer section may be used by algorithms wishing to differentiate between positions that can be used by all containers (i.e. near reefer platforms) and ones that cannot (i.e. away from reefer platforms, where reefer containers may not be placed). This could give more options to stack reefer containers, since normal containers are not in the way. The dedicated section is independent from the locations of the platforms and not used by the simulation model itself, besides the collection of statistics for both the reefer and normal sections.

4.4.3 Simulation parameters

The following other parameters can be manipulated:

- Used stacking/remarshalling algorithm
- Probability to include a container from the arrivals list in the simulation
- Simulation length
- Warming-up length
- Number of replications
- Whether or not to use a live message trace
- Whether or not to use a visualisation
- Visualisation speed

All these parameters can be changed with ease in a Microsoft Excel (.xls) file. This file is read using the Apache POI library [1].

4.5 Reporting and data generation

For every batch run of simulations, a subdirectory is created within the directory "results", which is located wherever the `sstack.jar` file is located. The subdirectory is given a date/timestamp as its name, to make sure it is unique. In the subdirectory, the following files are written:

- Trace log of events for every run
- Log files for every run
- Overall (aggregated and summarized) report

All files are plain text files, and can be opened with any ASCII text editor, such as Microsoft WordPad, to view their contents. Additionally, log files are made to be easily machine readable for data processing and analyzing tools, by adhering to a comma separated value (csv) format. Tools like Matlab can be used to analyse and visualise the data in these logs. The following log files are included for every run:

- **Crane activity** - per ASC a log containing time stamps combined with a value, which is valid from that time. Values can be either 0.0 (not busy) or 1.0 (busy).
- **Lane ground position usage** - per stacking lane a log containing time stamps combined with a value, which is valid from that time. Value is the number of occupied ground positions. There are separate logs for reefer and non-reefer sections.

- **Stacking lane usage** - per stacking lane a log containing time stamps combined with a value, which is valid from that time. Value is the number of TEU stacked in the lane. There are separate logs for reefer and non-reefer sections.

In the report files, statistics are reported. The report contains all statistics discussed in section 3.6, as well as the setup used, so it can be checked which inputs led to which outputs. There is a report file, in normal ASCII text format, for every simulation configuration file executed, so even if 10 replications were chosen, one report file, which averages the statistics over the 10 runs, is generated. Figure 4.2 contains an example of a report file. The trace logs are not particularly interesting usually, but are essential for debugging and verification work. They contain messages of events happening in the simulation, combined with a time stamp of when they occurred. Figure 4.1 contains an example of a trace log.

4.6 Results compilation utilities

When a lot of experiments are done, this will lead to very many report files, which makes interpreting the results nigh impossible. Also, creating many experiments one by one is a very tedious task and prone to mistakes. For this reason, we also created a few tools to assist creating experiments and compiling their results.

The first of these tools is the `ExperimentsGenerator`, which can be used to generate complete experiments of multiple input files. The `ExperimentsGenerator` is run from the command line, and takes several arguments, including: arrivals file, number of lanes, pile height, lane length and algorithm parameters. The `ExperimentsGenerator` then generates input files for every combination of the parameters provided. The following is an example call to the `ExperimentsGenerator`. The program will generate 2 (number of lanes) x 2 (heights) x 6 (algorithm parameters) = 24 input files in the subfolder "Experiment1". The experiments will use the algorithm "RommertsID1". Note that different algorithm parameters are separated by an underscore, and different values for these parameters are separated by a ">". The other simulation parameters are done individually and their values are separated by underscores.

```
java -jar ExperimentsGenerator.jar "Experiment1" "configMain.xls" "RommertsID1"
"arrivals18weeksLow20onlyV1.txt" "lanes_6_8" "height_3_4" "length_34"
"params_1_0_0_-0.03>-0.01>0.0>0.003>0.01>0.03"
```

The `ExperimentsGenerator` also creates a .bat file, which runs all the generated input files in sequence.

Another tool we created is called `Results2Excel`, which compiles the text report files of all the runs in a certain directory and puts them in a single Microsoft Excel file, where they can be easily compared.

```
48.028      ASC #6 has reached target lane position: Lane Position of GPs #1392
48.030      Straddle carrier #43 has reached stacking lane #3.
48.032      ASC #3 is now raised
48.033      ASC #6 is now lowered
48.033      Container #1618 was stacked on pile #118, where a lock for it was already
present.
48.033      ASC #6 has finished setdown
48.033      ASCTask #1618 (TPLandside to Pile task: container #1618 to Pile #118, with
position:
Lane Position of GPs #1392 and height 2 of max. 4) is now finished
48.036      ASC #3 has reached target lane position: Lane Position of GPs #746
48.036      Straddle carrier #45 loaded at TLP
48.037      ASC #6 is now raised
48.037      New task for ASC #6: SSTACk.main.asc.CarryLandsideToGPTask@15c2843
48.040      Straddle carrier #44 has reached stacking lane #3.
48.041      ASC #6 has reached the TP landside
48.042      ASC #3 is now lowered
48.042      Container #1622 was stacked on pile #247, where a lock for it was already
present.
48.042      ASC #3 has finished setdown
48.042      ASCTask #1622 (TPLandside to Pile task: container #1622 to Pile #247, with
position:
Lane Position of GPs #746 and height 2 of max. 4) is now finished
48.046      ASC #3 is now raised
48.046      New task for ASC #3: SSTACk.main.asc.CarryLandsideToGPTask@
15780d9
48.046      ASC #6 is now lowered
48.047      ASC #6 has finished pickup of container #1620
48.049      ASC #3 has reached the TP landside
48.049      ASC #3 is now waiting at TP Landside for container #Container #1623, size:
20 ft., normal, assigned pile: Pile #247, with position:
Lane Position of GPs #746 and height 4 of max. 4
48.050      Straddle carrier #43 unloaded cargo at Lane #3
48.050      Straddle Carrier #43 has been added to IDLE SCS.
48.050      ASC #3 has reached the TP landside
48.052      ASC #6 is now raised
48.055      ASC #3 is now lowered
48.056      ASC #3 has finished pickup of container #1623
48.056      ASC #6 has reached target lane position: Lane Position of GPs #1392
48.059      Straddle carrier #45 has reached stacking lane #3.
48.060      ASC #6 is now lowered
48.060      Container #1620 was stacked on pile #118, where a lock for it was already
present.
48.060      ASC #6 has finished setdown
48.060      ASCTask #1620 (TPLandside to Pile task: container #1620 to Pile #118, with
position:
Lane Position of GPs #1392 and height 3 of max. 4) is now finished
48.060      Straddle carrier #44 unloaded cargo at Lane #3
48.060      Straddle Carrier #44 has been added to IDLE SCS.
48.061      ASC #3 is now raised
48.063      ASC #6 is now raised
48.063      New task for ASC #6: SSTACk.main.asc.CarryLandsideToGPTask@1cc5461
48.065      ASC #3 has reached target lane position: Lane Position of GPs #746
48.067      ASC #6 has reached the TP landside
48.069      ASC #3 is now lowered
48.069      Container #1623 was stacked on pile #247, where a lock for it was already
present.
48.069      ASC #3 has finished setdown
48.069      ASCTask #1623 (TPLandside to Pile task: container #1623 to Pile #247, with
position:
Lane Position of GPs #746 and height 3 of max. 4) is now finished
48.072      ASC #6 is now lowered
48.072      ASC #3 is now raised
```

Figure 4.1: Example of a trace log produced by the simulation.

```
Overview simulation report for input file: configExp0.xls
Simulation length;2520.0
Warming up;504.0
Arrivals file: arrivals18weeksLow20onlyV1.txt
numArrNormQuay;51966
numArrReefQuay;0
numArrNormTLP;8580
numArrReefTLP;0
Number of lanes;8
Lane height;4
Lane width;6
Normal section length (excl platforms);34
Reefer section length (excl platforms);0
Number of AGVs;75
Number of SCs;75
Replications;10
Algorithm;TvrDtcMd
Algorithm parameters;1  0     0     0.03
      68.1977:79.4249:92.4209:111.5949:192.2115 0.003
Filter;DummyFilter
Filter parameters;
Execution time (seconds);11640.389
### meaning of stat probe lines below: name, average, minimum,
maximum, standard deviation*, CI left*, CI right*.  * = if applicable
###
Total number of remarshalling occasions
(normal);215.0;204.0;232.0;8.219218670625299;209.12032519306038;220.8
7967480693962
Total number of emergency remarshalling occasions
(normal);0.0;0.0;0.0;0.0;0.0;0.0
Total number of remarshalled containers
(normal);215.1;204.0;232.0;8.211780156174017;209.2256463857154;220.97
435361428458
Total number of emergency remarshalled containers
(normal);0.0;0.0;0.0;0.0;0.0;0.0
Total number of exiting remarshalling occasions
(normal);0.7;0.0;3.0;1.0593499054713802;-
0.057813266751798875;1.4578132667517987
Total number of exiting remarshalled containers
(normal);0.7;0.0;3.0;1.0593499054713802;-
0.057813266751798875;1.4578132667517987
Total number of remarshalling occasions
(reefer);0.0;0.0;0.0;0.0;0.0;0.0
Total number of emergency remarshalling occasions
(reefer);0.0;0.0;0.0;0.0;0.0;0.0
Total number of remarshalled containers
(reefer);0.0;0.0;0.0;0.0;0.0;0.0
Total number of emergency remarshalled containers
(reefer);0.0;0.0;0.0;0.0;0.0;0.0
Total number of exiting remarshalling occasions
(reefer);0.0;0.0;0.0;0.0;0.0;0.0
Total number of exiting remarshalled containers
(reefer);0.0;0.0;0.0;0.0;0.0;0.0
Total number of normal containers not stacked
(quay);0.0;0.0;0.0;0.0;0.0;0.0
Total number of reefer containers not stacked
(quay);0.0;0.0;0.0;0.0;0.0;0.0
Total number of normal containers not stacked
(tlp);0.0;0.0;0.0;0.0;0.0;0.0
Total number of reefer containers not stacked
(tlp);0.0;0.0;0.0;0.0;0.0;0.0
Average time ASC waiting for
AGV;0.03389073548932495;0.033696604896576915;0.03403242756113507;9.87
9122668491618E-5;0.033820064503436484;0.03396140647521341
Average time ASC waiting for
SC;0.05232126097532896;0.051938786146372755;0.052534427576820815;1.84
```

Figure 4.2: Example of a report file.

4.7 Visualisation

To get a better understanding of what the program is doing whilst simulating (which also turned out to be very useful for debugging and verification), a visualisation using Java's Swing library was created. This visualisation (which can be turned off, if so desired, e.g. in case of a batch-run of 100 simulations), pops up in two frames after the simulation is started. The first frame contains an animation: besides plotting the current simulation time, it draws an abstract representation of all stacking lanes on the screen, including their contents and the position of their ASCs.

For every lane, the different levels are plotted next to each other. This way, one can easily see which ground positions are filled with piles and how high the piles are. It is also shown whether any stack locks are present on the individual pile slots. Finally, reefer containers are marked with an asterisk (*) in their middle.

A screen shot of the animated visualisation of the program can be seen in figure 4.3. We can see containers of various sizes in the stack (some of which, namely the green dashed boxes, have not yet been stacked), as well as ASC's (the purple bars) moving around the lanes. Neither of the two ASCs shown in the picture is currently carrying a container.

The second frame contains data about the current state of the simulation. In various tabs it can be seen what every transporter is doing, what the task queues of the ASCs contain and which containers are waiting at the transfer points. An example is in figure 4.4, where we see that lane #0 has 23 containers stacked, which amounts to 2.81% of capacity. At the transfer point quayside, there is one AGV waiting to be serviced. The ASCTask for this can be seen in the ASC's task list above the central line. Besides the current task, at the moment there are two more tasks scheduled.

4.8 Final remarks

In this chapter, we discussed SSTACk, a simulation program for ECT-like container terminals, which was programmed in Java using the SSJ discrete-event simulation library. The program can simulate container terminals based on lengthwise stacking lanes, with a single ASC per lane. The program was written to be extendable, so with some effort it could be extended to be able to simulate other container terminal types, e.g. with more than one ASC per lane. A basic system for AGVs and straddle carriers is also implemented, but it has not been developed into too much detail yet, because this is outside the scope of this thesis.

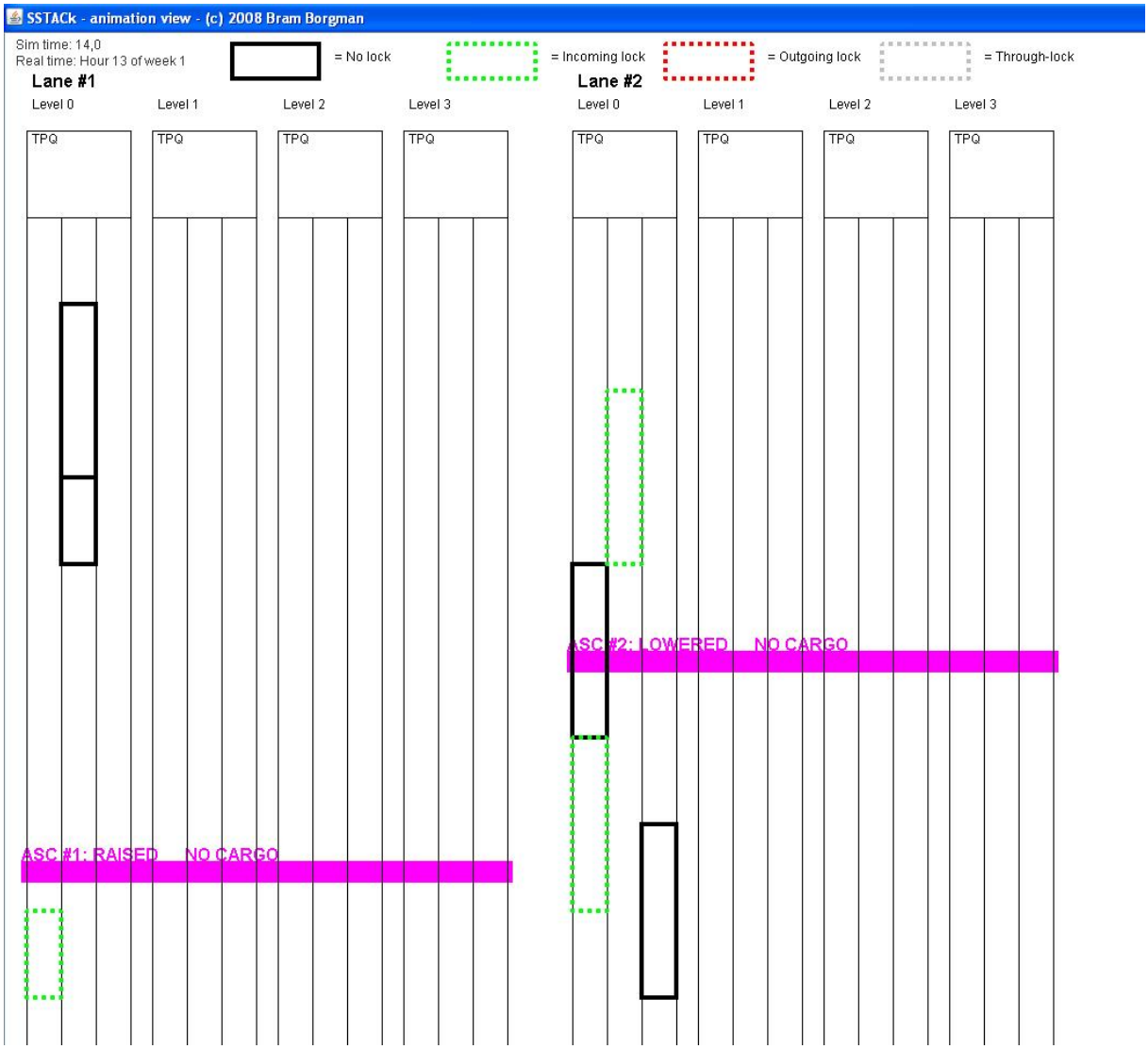

Figure 4.3: Screen shot of the visualisation of SSTACk.

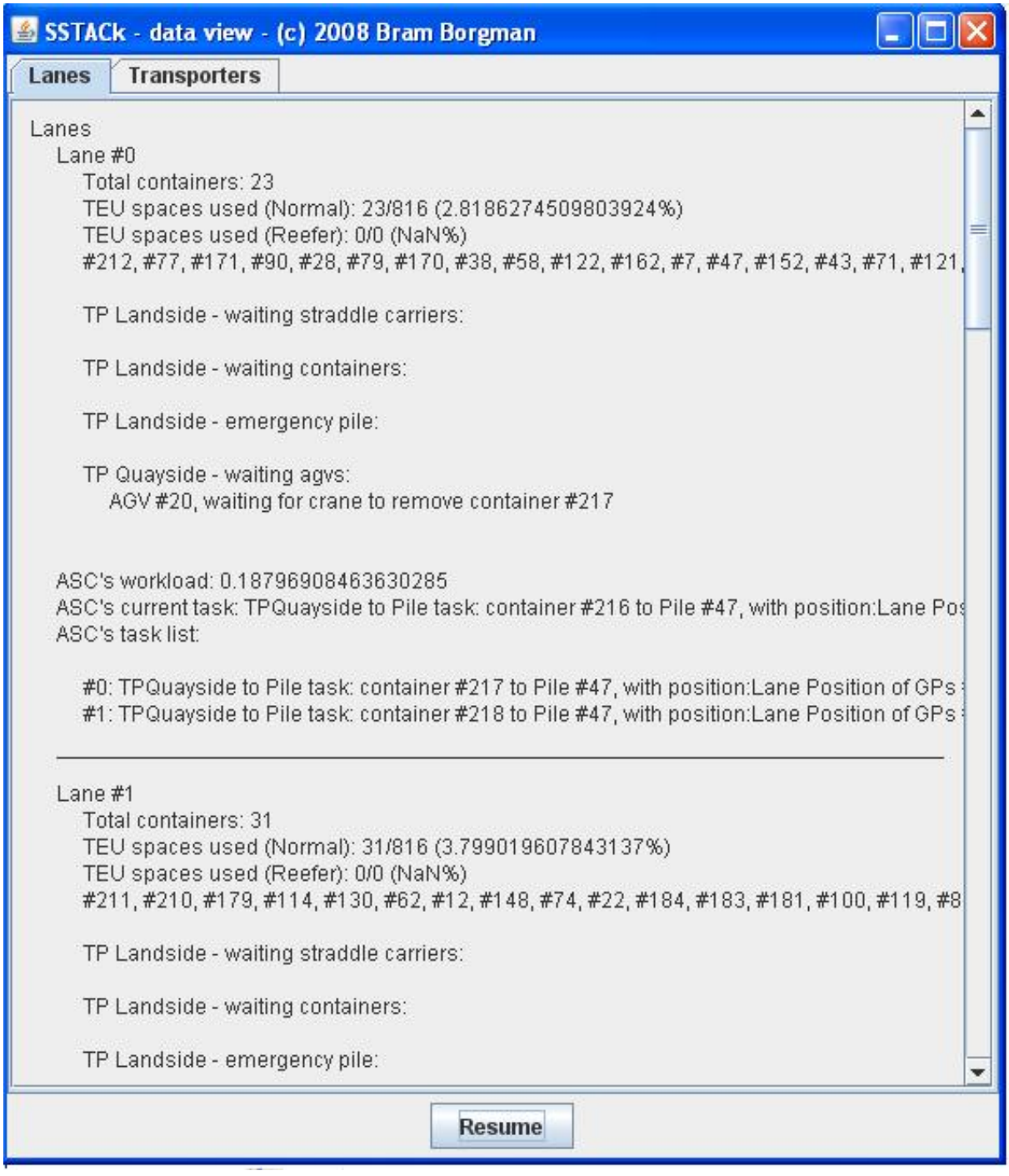

Figure 4.4: Screen shot of the data mode runtime visualisation of SSTACk.

Chapter 5

Generator program

In section 4.4 we said that the simulation program requires a list of arrivals of containers as one of its inputs. Because there are a lot of containers in one simulation, this cannot be done by hand. Therefore, we created a generator program for creating these files, which is described in this chapter. This chapter is organised as follows: in section 5.1 we describe the data gathered for the generator program, in section 5.5 we discuss the output of the generator program and in section 5.6, we describe how the generator program works. Also of interest may be appendix B, which contains pseudo code of the algorithms described in this chapter.

5.1 Data for the generator program

Since, in the interests of validation, we would like to have as realistic arrivals as possible, we cannot just pull data out of thin air. However, getting all the data needed would be an enormous task, and is outside the scope of this thesis. Fortunately though, there is some data available from previous research [33, 35], which we can use. Unfortunately, the data is not in a format which can be used directly, and hence some processing has to take place.

In the data, containers can arrive and leave by the following modes:

- Jumbo cargo ship (8000 TEU or more)
- Deep sea cargo ship (less than 8000 TEU)
- Short sea/feeder cargo ship
- Barge
- Rail
- Truck

For each of these modes, a list of arrivals has to be created, and then for every (ship/rail/truck) arrival, individual (un)load times per container have to be generated. Then, these times have

to be linked to an actual container with size, category and type. Also, incoming and outgoing arrivals have to be matched, so that there is a departure time for every arrival time.

From the available data, only categories, types and sizes for containers to and from jumbos and deep sea ships are known. Using this data, we can match containers to arrive and depart via those ships with arrivals from other modes. However, we also need this data for short sea ships, as there are also containers bypassing jumbos and deep sea ships completely, as can be seen in table 5.1 and 5.2. In section 5.2.3, we discuss how this problem was solved.

There are two scenarios available: one with 186000 containers per 3 weeks ("high") and one with 37200 per 3 weeks ("low"), which is exactly one fifth of the high scenario's number. Both scenarios are discussed in the following sections.

5.2 Arrival times per ship/train

This section discusses how the arrival times of ships and trains at the terminal are calculated.

5.2.1 Jumbo and deep sea ships

Scenario "high"

In the previous research, jumbo ships arrive 15 times in every period of three weeks and deep sea ships arrive 20 times per three weeks. If we then assume that the first ship arrives at Monday 0:00 hours and that interarrival times are equal, we get the following schedule:

Jumbo:	week 1	Mon 0:00, Tue 9:36, Wed 19:12, Fri 4:48 and Sat 14:24
	week 2	the same
	week 3	the same
Deep sea:	week 1	Mon 0:00, Tue 1:12, Wed 2:24, Thu 3:36, Fri 4:48, Sat 6:00 and Sun 7:12
	week 2	Mon 8:24, Tue 9:36, Wed 10:48, Thu 12:00, Fri 13:12, Sat 14:24 and Sun 15:36
	week 3	Mon 16:48, Tue 18:00, Wed 19:12, Thu 20:24, Fri 21:36 and Sat 22:48

A random number between -12 and +12 hours (uniformly distributed) is then added to these times to determine the actual arrival time of the ships.

Scenario "low"

In the low traffic scenario, jumbo ships arrive only once a week, according to the following schedule:

Week 1: Thursday 0:00, Week 2: Wednesday 0:00 and Week 3: Tuesday 0:00.

	To jumbo	Deepsea	Shorts./feed.	Truck	Rail	Barge	Total
From Jumbo	0	11660	18150	2035	4825	7025	43695
Deepsea	13455	6720	7330	1350	1945	2840	33640
Shorts./feed.	19350	10000	0	4835	9380	13675	57240
Truck	2190	1840	4835	0	0	0	8865
Rail	5235	2700	9385	0	0	0	17320
Barge	7620	3940	13680	0	0	0	25240
Total	47850	36860	53380	8220	16150	23540	186000

Table 5.1: Number of containers by import and export modality within the terminal every 3-week period in the "high" scenario. Source: [33].

	To Jumbo	Deepsea	Shorts./feed.	Truck	Rail	Barge	Total
From Jumbo	0	2332	3630	407	965	1405	8739
Deepsea	2691	1344	1466	270	389	568	6728
Shorts./feed.	3870	2000	0	967	1876	2735	11448
Truck	438	368	967	0	0	0	1773
Rail	1047	540	1877	0	0	0	3464
Barge	1524	788	2736	0	0	0	5048
Total	9570	7372	10676	1644	3230	4708	37200

Table 5.2: Number of containers by import and export modality within the terminal every 3-week period in the "low" scenario. Source: [33].

Note that while interarrival times are 144 hours, there is no arrival on Monday at 0:00. For deep sea ships, a similarly thinned schedule is used:

Week 1: Mon 0:00, Sun 0:00 Week 2: Sat 0:00 and Week 3: Fri 0:00.

In this way, a ship (jumbo or deep sea) arrives every three days.

5.2.2 Trucks

Trucks arrive in a particular pattern during a week. During weekends and nights, almost no trucks would arrive, whereas the majority of trucks arrived during the afternoon and early evening. The pattern of arrivals over a week can be seen in figure 5.1. We assume that every truck carries only one container, regardless of its size.

Scenario "high"

In the "high" scenario, the weekly number of containers arriving and leaving by truck is known to be 2955 and 2740, respectively. We can then calculate the number of containers arriving every hour by multiplying these values with the percentage corresponding to that hour of the week. In case that, due to rounding, too many or too few arrivals were generated, extra ones are

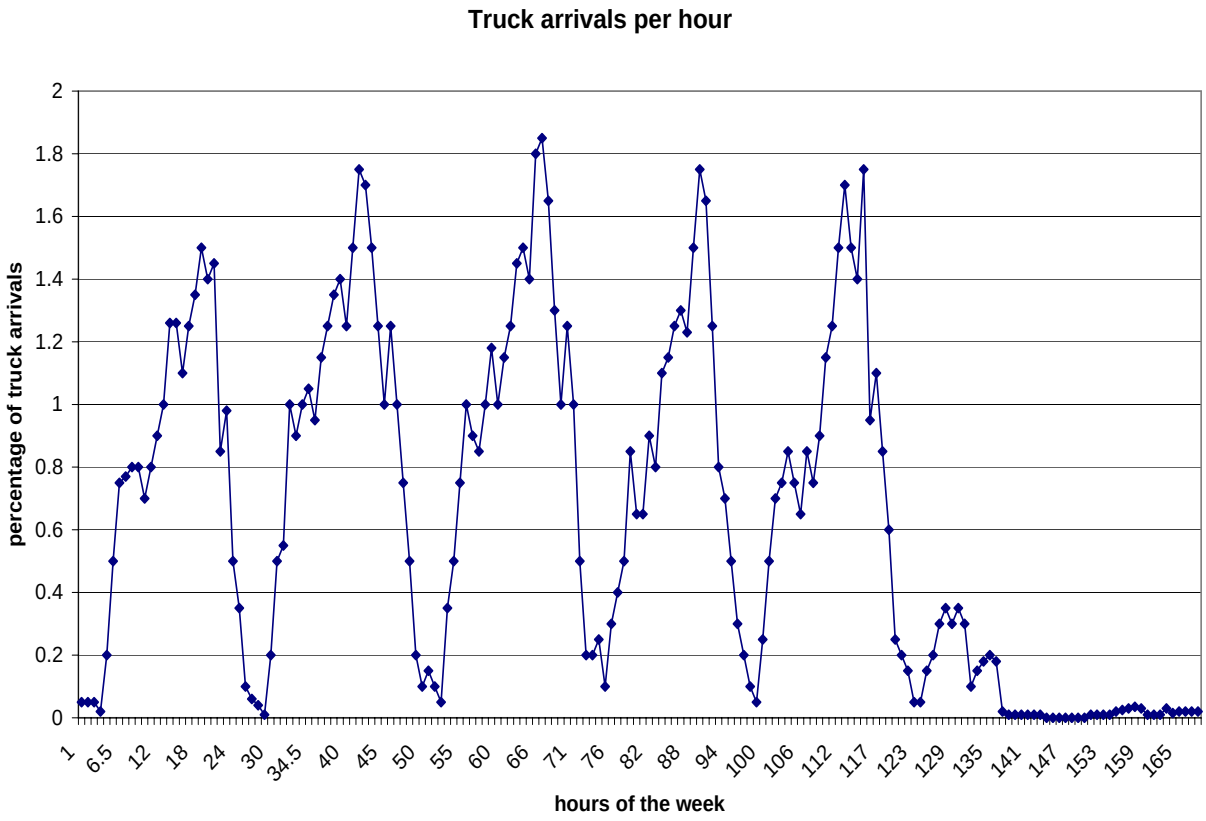

Figure 5.1: Arrival pattern of trucks during one week. Source: [33, 35]

added and removed. For each of these arrivals, a random time within that hour is then drawn from a uniform distribution.

Because the available data models the truck arrivals in a single full week, the generator program can generate one full week of arrival times at a time. Several weeks may be generated by applying the generator a number of times in succession. In case less than 168 hours of the last week are needed, the excess arrivals are removed.

Scenario "low"

In the "low" scenario, the weekly number of containers arriving and leaving by truck is known to be 591 and 548, respectively. The real arrivals are then determined in the same way as in the "high" scenario.

5.2.3 Other modes

For the other modes, i.e. rail, barge and short sea, no such pattern as with trucks or big ships was found by the previous researchers. However, for rail at least, this contradicts [31], which mentions "a peak load in the night" for rail arrivals. However, since there is no directly available data on the exact size of this peak, and because other modes (except truck) also ignore it, we do our research with arrivals spread evenly over 24-hour periods.

Voogd et al. use more or less constant interarrival times for train, barge and short sea containers. This method ignores the fact that whenever a ship arrives, a peak in demand on the terminal appears. Therefore, we model the arrivals of trains, barges and short sea ships just like jumbos and deep sea ships. However, since no schedule is known, we have to construct this first.

Scenario "high"

We do this by determining how many ships are needed to load and unload the number of containers the terminal processes according to table 5.1. We assume that short sea ships load and unload 400 containers, that barges load and unload 100 containers and trains load and unload 80 containers. We then get the following amounts of arrivals per 3 weeks:

Short sea: $((53380 + 57240)/2)/400 = 138.275$ arrivals
Barge: $((23540 + 25240)/2)/100 = 243.9$ arrivals
Rail: $((16150 + 17320)/2)/80 = 209.1875$ arrivals

Since three weeks equals 504 hours, we get the following interarrival times:

Short sea: $504/138.275 \approx 3.645$ hours
Barge: $504/243.9 \approx 2.066$ hours
Rail: $504/209.1875 \approx 2.409$ hours

Now, since table 5.1 shows that the number of containers loaded and unloaded for these modes in not equal, we have to calculate how many containers are to be unloaded:

Short sea: 57240 containers unloaded/138.275 arrivals $\approx$ 414 containers per ship
Barge: $25240/243.9 \approx 103$ containers per ship
Rail: $17320/209.1875 \approx 83$ containers per train

And how many are to be loaded:

Short sea: 53380 containers loaded/138.275 arrivals $\approx$ 386 containers per ship
Barge: $23540/243.9 \approx 97$ containers per ship
Rail: $16150/209.1875 \approx 77$ containers per train

These numbers have been rounded and are unlikely to sum to the total numbers specified in table 5.1. Therefore, we randomly add extra containers to the arrivals, until the number is reached. In case too many containers were already generated, they are randomly removed. Adding and removing is done with every ship/train arrival having an equal probability to lose a container.

Just like with the other modes, we apply a uniformly distributed perturbation for each mode, albeit a much smaller than with deep sea ships and jumbos. We draw a random number between -1 and +1 and add this to their scheduled arrival time. The first arrival of each mode is assumed to be at 0.0 hours.

Crane #	Holds
Crane 1	24, 23, 22, 20, 19, 21
Crane 2	17, 16, 15, 14, 13, 18
Crane 3	12, 8, 9, 10, 11
Crane 4	1, 2, 3, 4, 5, 6, 7

Table 5.3: Deep sea ship loading/unloading order. Quay cranes first unload a hold and then load it, before moving onto the next one. Source: [33]

Scenario "low"

For the "low" scenario, the same calculations are made, only ships and trains arrive 5 times less often, i.e. we multiply their interarrival times by 5. The number of containers each transporter delivers and takes is kept the same.

5.3 Arrival and departure times for individual containers

Because the SSTACk simulation program does not model the arrivals of entire ships, we have to calculate the individual times of containers as they are passed on by the quay cranes. In this section, we describe how these times are calculated for each arrival/departure mode.

5.3.1 Jumbo and deep sea

For both jumbo and smaller deep sea cargo ships, the exact loading and unloading sequence is given. It is assumed the order is the same every time the ship type is handled. This allows the use of categories for these two modes.

Jumbos and deep sea ships carry their cargo in *holds*. One hold per crane at a time is unloaded. If a hold has been fully unloaded, the loading of the same hold is started. Once this is finished, the quay crane moves on to the next hold in its schedule, a move which takes 2.5 minutes (i.e. 0.04167 hours). There are 4 quay cranes used to unload a deep sea ship and 6 to unload a jumbo. The order in which holds are loaded and unloaded is shown in tables 5.3 and 5.4. Note that a deep sea ship has 24 holds and a jumbo has 25. Unloading or loading a single container is assumed to take 1/35 of an hour, as quay cranes reach productivity levels of about 35 cranes per hour (see [32], page 19).

5.3.2 Trucks

Because it is assumed a single trucks only carries a single container, the individual arrival and departure times of containers moved by trucks are exactly the same as the trucks' arrival times.

Crane #	Holds
Crane 1	17, 16, 15
Crane 2	14, 13, 12, 10, 11
Crane 3	8, 9, 7, 6
Crane 4	5, 4, 3, 2, 1
Crane 5	21, 20, 19, 18
Crane 6	25, 24, 23, 22

Table 5.4: Jumbo ship loading/unloading order. Quay cranes first unload a hold and then load it, before moving onto the next one. Source: [33]

5.3.3 Other modes

For the containers arriving and leaving by train, barge or short sea ship, we assume they are first all unloaded and then loaded. In reality, this may be mixed, but since no data is available, and since it is unlikely to seriously alter the results, we disregard this possibility.

Because of the small sizes of the transporters in these modes, we assume each is served by only a single crane, which has a productivity of 35 containers per hour. Hence, unloading or loading a single container again takes 1/35 of an hour.

5.3.4 Expected arrival and departure times

The expected arrival and departure times for containers departing via ship or train are the same as their actual arrival or departure times, only instead of the actual arrival time of the ship (which, in the case of jumbos and deep sea ships, can be as much as twelve hours too late or early), the scheduled arrival time is used as basis for the individual containers' times.

For trucks, a different method is used. We start with the truck's arrival time and then add some time to it as follows:

- 5% probability of being 24 to 6 hours early (uniformly distributed)
- 90% probability of being 6 hours early to 6 hours late (uniformly distributed)
- 5% probability of being 24 to 6 hours late (uniformly distributed)

This models any delays the trucks may have, but makes sure most trucks are not that much too late or early.

5.4 Matching arrivals and departures

In this section, we describe how individual arrivals and departures are matched, so that an input file for the simulation can be created. Recall (see section 4.4.1) that every row in the file should contain the following fields: arrival time and mode, size, category, type, departure time and departure mode.

From Jumbo	#To Deep sea		From Deep sea	#To Jumbo	#To Deep sea
J1	50		DS1	45	34
J2	83		DS2	135	34
J3	225		DS3	179	34
J4	225		DS4	269	34
			DS5	269	100
			DS6		100

Table 5.5: Flows from jumbos to deep sea ships and from deep sea ships to jumbos and other deep sea ships in the "high" scenario. J1 and DS1 are the ships that arrived closest before the current ship. Source: [33]

5.4.1 Containers both arriving and leaving via jumbo or deep sea ship

Scenario "high"

When all arrival and departure times of all containers have been generated, we first match the arrivals of containers which both arrive and leave by jumbo or deep sea ship. For these combinations of modes, exact quantities are available from the data. The quantities for the "high" are in table 5.5. Jumbo ships receive containers from the 5 deep sea ships which arrived closest before they did. Deep sea ships on the other hand, receive containers from the 4 jumbos and 6 deep sea ships which arrived closest before their arrival time. Because the arrival times per ship are stochastic (but still based on an underlying schedule), the last of these ships has to have a scheduled arrival time of at least 48 hours before the scheduled arrival time of the receiving jumbo or deep sea ship. Since both the ships for arrival and departure can be 12 hours late or early, this leaves at least 24 hours for loading, unloading and marshalling in the yard.

For each departure, a random matching (i.e. same size and type) arrival from the corresponding arrival ship is chosen. This is possible because size and type of both the arrivals and the departures is known for deep sea and jumbo ships. The chosen departure and arrival are then removed from the lists they were in, so they can not be chosen again.

Scenario "low"

For the "low" scenario, flows are a bit different. In particular, because there are fewer ships, every ship is in time for the next departure. This means, however, that for one of the 3 jumbo ships, 2 deep sea ships directly precede its arrival. The arrivals and departures for these have to be divided among the 2 deep sea ships. In table 5.6, the exact amounts from every jumbo/deep sea ship to every deep sea ship (and vice versa), are shown. When the amount of containers moved between the two ships has been determined, the arrivals are matched just like in the "high" scenario.

	To Jumbo 1	J2	J3	Deep sea 1	DS2	DS3	DS4
From Jumbo 1	0	0	0	0	583	0	0
J2	0	0	0	0	0	583	0
J3	0	0	0	583	0	0	583
Deep sea 1	449	0	0	0	336	0	0
DS2	0	897	0	0	0	336	0
DS3	0	0	897	0	0	0	336
DS4	448	0	0	336	0	0	0

Table 5.6: Flows from jumbos to deep sea ships and from deep sea ships to jumbos and other deep sea ships in the "low" scenario. Source: [33]

5.4.2 Other containers

When all deep sea-jumbo combinations have been processed, all other departures are matched with arrivals. The method of matching is the same for both scenarios. There are two types left: containers which enter or leave using a jumbo or deep sea ship, and containers which do not use either at all. The difference is that the category, size and type is known for the former type, whereas it is not for the latter. How this was solved is explained in section 5.4.3.

To match an arrival and a departure, first a random departure is chosen. Every individual departure (which has not been matched yet) has an equal probability of being chosen. After this, an arrival mode is chosen. This is done randomly, but probabilities are not equal for all modes: the probability an arrival mode is chosen is dependent on the number of containers that should flow between this mode and the departure mode, minus the number of containers between these modes that has already been assigned.

After selecting the arrival mode, we choose a random aimed dwell time. This is the time we would want the container to stay in the stack. This time is randomly drawn from a triangular distribution with minimum 2 days (48 hours), mode 3 days (72 hours) and maximum 6 days (144 hours). The generator then searches for the arrival that would lead to the dwell time closest to the time aimed for. This time should be at least 2 days (the minimum of the distribution) before departure. If the aimed arrival time is negative (e.g. on the first few ships and trains, no arrivals are available, as they would be in negative time), the departure is discarded. If still no matching arrival could be found, we just use the aimed time as departure time, so that the container is not lost. We only do this in this part, because the probability of not being able to find a match is the greatest here.

We chose the aforementioned triangular distribution, because in previous research and in the available data, the mean time in the stack for a container was 3.5 days, and only very few containers stayed for over a week. The chosen parameter values of the distribution yield one with an expected mean of 3.5 days. The triangular distribution has a low probability density at its edges, so it accurately models the fact that few containers stay either very long or very short. The values of the distribution were tweaked to ensure the average dwell time was not too great (which could lead to an overfull stack).

Load	#reefer	#normal	#total		Unload	#reefer	#normal	#total
20 ft.	10	771	781		20 ft.	8	609	617
40 ft.	118	795	913		40 ft.	117	813	930
45 ft.	11	138	149		45 ft.	11	124	135
total	139	1704	1843		total	136	1546	1682

	%reefer	%normal	%total			%reefer	%normal	%total
20 ft.	0.5426%	41.8340%	42.3766%		20 ft.	0.4756%	36.2069%	36.6825%
40 ft.	6.4026%	43.1362%	49.5388%		40 ft.	6.9560%	48.3353%	55.2913%
45 ft.	0.5969%	7.4878%	8.0846%		45 ft.	0.6540%	7.3722%	8.0262%
total	7.5421%	92.4579%	100.000%		total	8.0856%	91.9144%	100.000%

Table 5.7: Distribution per size/type-combination of deep sea vessels. Source: [33, 35]

5.4.3 Determining the category of containers neither arriving nor leaving via jumbo or deep sea ship

As we can see from tables 5.1 and 5.2, not all containers are in a jumbo or deep sea ship when they enter or leave the terminal, and hence we cannot determine their category, size and type this way. All these containers are, however, at one time in a short sea ship. Unfortunately, there is no data available on the contents of these short sea ships, so we have to make the assumption that short sea ships carry the same types of containers as deep sea ships, in the same (relative) quantities. We look at deep sea ships, rather than jumbos, because their sizes are more alike.

To determine the size and type of a container, we look at the distribution of container types of deep sea ships, and calculate every combination's probability. The distribution of incoming (Unload) and outgoing (Load) containers for deep sea ships is in table 5.7. We can use these percentages as probabilities for containers in short sea ships. For example, there would be a 43.1362% probability of a loaded container being 40 ft. long, and not a reefer. There is a 0.6540% probability an unloaded container is a 45 ft. long reefer.

5.5 Generated data

We have run the generator program for the "low" scenario, and found that the average dwell time is about 90.5 hours, or 3.77 days. This is very close to the time of 3.73 days that was generated for the previous research [33]. The minimum dwell time observed is 28.9 hours and the maximum is 192.2 hours. These are not absolute values, however, since every new run (i.e. with different random seeds) gives different results due to randomness. The number of containers generated per three week period is on average slightly below the originally specified value of 37200. This is due to randomness and to the fact that sometimes a departure can not be matched with an arrival. However, because this occurrence is so rare, we ignore it and do not look for a way to include also the unmatched arrivals and departures in the final arrivals list.

5.6 Description of the generator program

The generator program was written in Java. Development of the generator did not go as smooth as hoped, mainly due to difficulties in obtaining the right data from the original source. Because there were some inconsistencies and omissions, some assumptions had to be made, as well as a lot of trial and error.

The program itself consists of several static methods, because this type of program is not suitable for true Object Oriented Programming (OOP), which is the paradigm on which Java is based. Almost all code is thus in a single class, with some auxiliary and wrapper classes to make things a bit easier. For this reason, we do not include UML diagrams for the generator.

Since we found OOP of little use for the generator, it is also possible to use languages and tools, which are better suited for purely algorithmic programming, such as Matlab.

5.7 Conclusion

The generator program described in this chapter is able to generate arrivals files for the simulation tool described in chapter 4. The program is based on earlier work, and was made to be able to use the available data as good as possible.

There are 6 modes of transport for the generator: jumbo ship, deep sea ship, short sea ship, barge, train and truck. For every mode, a schedule is generated, and for every ship or train arrival, unloading and loading times are generated. For trucks, times are generated according to a pattern of the distribution of arrivals over the day. The resulting loading and unloading times are then matched, according to an OD-matrix. The exact times matched are chosen to be matched as close as possible to a value drawn from a (triangular) distribution.

Chapter 6

Verification

Model verification is, according to Schlesinger [26]: *"ensuring that the computer program of the computerized model and its implementation are correct"*, and thus basically means checking whether the program does what it is documented to do, according to the conceptual model (see chapter 3). Verification should not be confused with *validation*, which is, according to [26]: *"substantiation that a computerized model within its domain of applicability possesses a satisfactory range of accuracy consistent with the intended application of the model"*, so simply put: making sure that what the program/model is doing, is realistic w.r.t. the real world. While validation is important, it is very hard to do, and since this thesis project is not about making a program to be used in practice, we largely skip this step. Verification is step 6 in the simulation studies framework used in this thesis (see figure 1.1).

According to [25], when verifying a computerized model (such as the one discussed here), the most important factor is what programming language has been used; i.e whether a specialized simulation language (e.g. SIMAN), or a general-purpose language (e.g. Java, FORTRAN, C++) is used. The latter is true in our case, and it is pointed out that general-purpose languages tend to have more errors in simulation programs developed with them, because they give the programmer more freedom - but also more freedom to make mistakes [25]. The use of special libraries can only partially offset this.

This chapter is organised as follows: in section 6.1, we discuss which verification techniques are available and in section 6.2, we discuss the verification of our simulation program.

6.1 Verification techniques

Note: parts of this section were taken from earlier work by the same author [3].

According to Whitner and Balci [37], there are six techniques to verify a model (choosing just one of them will not suffice, though):

1. Informal analysis
2. Static analysis

3. Dynamic analysis
4. Symbolic analysis
5. Constraint analysis
6. Formal analysis

Informal analysis Informal analysis includes such things as code checking and desk checking (i.e. running the program whilst programming to quickly check for bugs etc.). These two forms of analysis are usually not documented, although they are used frequently, which is why we do not go more in-depth into these techniques, although they have been used in the development process.

Static analysis Static analysis includes techniques such as syntax analysis (checking whether the program's code adheres to the specifications of the used language). This is done by the Java compiler and the Eclipse IDE (Integrated Development Environment) used, so checking this was easy. Other static analysis techniques include semantic analysis and structure analysis, but these are harder to implement, and as [37] says, "these techniques are limited in what they can actually verify". Eclipse has some functions implementing semantic analysis and structure analysis.

Dynamic analysis Dynamic analysis is done by executing the simulation program. By examining the outputs the program gives, the program can be verified. We use some of the dynamic analysis techniques available, and therefore refer to section 6.2.

Symbolic analysis This type of analysis is similar to dynamic analysis in that it too examines the behaviour of the program during execution. *Symbolic execution* is the most used symbolic analysis technique. It works by giving the simulation "symbolic" (i.e. fictional, but legal) inputs and then examining the outputs. We also use this technique, so we refer to section 6.2.

Constraint analysis Constraint analysis checks whether assumptions made in the conceptual model are correctly reflected in the program. Techniques include *assertion checking* (adding statements to the code that produce errors when an assumption is violated) and *boundary analysis* (which says that when creating test cases, always use data along boundaries of very different expected results, e.g. enter months 0, 1, 12 and 13 in a program - 0 and 13 should result in very different output).

Formal analysis Formal analysis is based on obtaining formal mathematical proof of correctness. This is very hard, if not impossible, to do [37], however, and probably suited best for simulations depending on only some equations, rather than discrete-event simulation (especially when stochasticity is involved), so we do not use it in this thesis.

6.2 Verification of the program

For the verification of the simulation program we are using a combination of dynamic, symbolic and constraint analysis, besides the use of informal and static analysis already discussed in section 6.1.

Assertion checking (a form of constraint analysis) is done by adding numerous assertions to the code, which make the program terminate whenever a situation, which should be impossible to reach, is encountered. Examples include: attempted stacking on a pile where this is impossible, attempted pickup when a container is not at the top of the pile and when an ASC has moved to the wrong location. Many bugs and other errors were fixed during development this way (see: informal analysis), something which was made easier by using the (animated) visualisation to discover the causes of bugs.

The related techniques of dynamic analysis, symbolic analysis and boundary analysis are combined by creating some test cases, of which the outputs are then examined for errors. These tests start simple, to check whether basic functionality is working correctly, after which gradually more complex scenarios are tested. While no errors were discovered, it should be noted that while testing is useful is boosting confidence in the program, not finding any errors does not mean the program is error-free. As Dijkstra once said:

> *"Program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence."* [6]

6.2.1 Test case 1

In this test case, we start simple. There is one stacking lane of width 2, length 8 and maximum height 3. There are only 20 ft and 40 ft containers and none of them are reefers. Categories are ignored: the used stacking algorithm is *levelling*, because its behaviour is easiest to predict. All containers enter and leave through the truck loading point, so AGVs are not included. Straddle carriers take a fixed amount of time to carry out their tasks: 1 time unit for moving, as well as 0.2 time units for picking up and setting down. We use one straddle carrier. The lane's ASC takes 1 time unit to raise or lower itself, in addition to 0.05 time units for picking up and setting down. The ASC's movement time is dependent on the distance it travels. The ASC is given a speed of 200 ft per time unit. We only stack a limited number of containers, so that the straddle carrier and the ASC never run behind schedule. For a list of containers, see table 6.1.

We run this setup for 400 time units ("hours"), so that there is plenty of time for all containers to exit the simulation.

In the simulation's output, we expect no remarshallings, as containers exit the simulation in reverse order. Likewise, there should be a maximum of 1 container waiting at the transfer point landside at any time. From the list in table 6.1, we can calculate that the average time a container should be standing in the stack is at least 170 time units. However, because there are pickup delays and travel times, this time is a bit longer: ASC pickup and setdown of 3.1 time units, straddle carrier pickup/set down of 0.4 time units and straddle carrier travel of 1 time

#	Arrival	Size	Departure
1	0.0	20	320.0
2	10.0	40	310.0
3	20.0	20	300.0
4	30.0	40	290.0
5	40.0	20	280.0
6	50.0	40	270.0
7	60.0	20	260.0
8	70.0	40	250.0
9	80.0	20	240.0
10	90.0	40	230.0
11	100.0	20	220.0
12	110.0	40	210.0
13	120.0	20	200.0
14	130.0	40	190.0
15	140.0	20	180.0
16	150.0	40	170.0

Table 6.1: Containers in test cases 1 and 2.

unit, plus any time the ASC has to move, which is variable. This makes for a total value of 4.5 time units.

The time the ASC will have to wait for the straddle carrier to deliver a container is also variable, but has to be at most 2.4 time units, for the straddle carrier to move from the lane to the truck loading point (1 time units), pickup a container (0.2 time units), moving to the lane (1 hour) and setting down the container (0.2 time units). The minimal value is 0.4, which occurs only for the very first container: the ASC starts at the transfer point quayside and has to move 200 feet, which takes one time unit. The straddle carrier starts at the truck loading point and also has to travel 1 time unit. This leaves only the pickup and set down by the straddle carrier for 0.4 time units.

The time between call for pickup (departure time) and actual departure depends on the ASC, which has to move (variable time), pick up the container (2.05 time units for 3 actions), move again (variable time) and set down the container (another 1.05 time units). The container then has to be picked up by a straddle carrier (0.2 time units), move (1 time unit) and set down (0.2 time units). This gives a minimum value of 4.5 time units. Any ASC movement comes on top of this.

The stack should at its peak contain 24 TEUs: 8 containers of 20 feet (1 TEU each) and 8 containers of 40 feet (2 TEU each).

When we examine the results, we can conclude that all expectations were met. Also, the trace and animation show no apparent irregularities. Hence, we assume all functionality in this test works correctly, and move on to test 2.

#	Arrival	Size	Departure
1	0.0	20	170.0
2	10.0	40	180.0
3	20.0	20	190.0
4	30.0	40	200.0
5	40.0	20	210.0
6	50.0	40	220.0
7	60.0	20	230.0
8	70.0	40	240.0
9	80.0	20	250.0
10	90.0	40	260.0
11	100.0	20	270.0
12	110.0	40	280.0
13	120.0	20	290.0
14	130.0	40	300.0
15	140.0	20	310.0
16	150.0	40	320.0

Table 6.2: Containers in test case 3.

6.2.2 Test case 2

In test case 1, we tested straddle carriers, but no AGVs. In test case 2, we let all containers originate and leave at the quay, and henceforth use an AGV instead of a straddle carrier. The AGV has a fixed travel time of 1.0 time units. Loading or unloading at the quay takes 0.5 time units, servicing by an ASC takes 0.1. We keep everything else the same. While there is still plenty of time to remove all containers, the exact times will differ due to the different functionality of AGVs: they need to be serviced by the ASC.

With AGVs, the average time of containers in the stack should be at least 4.65 above nominal scheduled time: ASC pickup and setdown of 3.15 time units, AGV movement of 1 time unit and AGV unloading 0.5 time units. Again, the real value should be higher, due to ASC movement.

In the results, we find that the minimal time a container spent in the stack is 24.850, 4.850 time units above the lower bound. This and the other results indicate that the tested AGV functionality also works properly.

6.2.3 Test case 3

In test case 3, we change the list of container departures to test remarshalling. The new times are in table 6.2.

Because containers now exit the terminal in the same order as they came in, some containers will need to be remarshalled. In this case, all containers stacked on the second layer of the lane need to be remarshalled. As 6 containers (4x 20 ft, 2x 40 ft) are placed on the ground at the left positions of the lane and 5 containers (2x 20 ft, 3x 40 ft) are placed on the right positions, this leaves 5 containers on layer 2 which definitely need to be remarshalled. However, for the first 2

remarshalling occasions (a 20 ft and a 40 ft container), the only positions available are on top of other containers, which will depart before they do. Therefore, these containers will have to be remarshalled again, to bring the total to 7 remarshalling occasions.

The results of this test confirmed that the program has correctly determined which containers had to be remarshalled, as indeed 7 containers were remarshalled.

6.2.4 Test case 4

In test case 4, we test emergency remarshalling to the transfer point landside. We use the same arrivals as in test case 3 (see table 6.2), but use a smaller stack. We use one lane of length 8 TEU, width 1 container and maximum height 3 containers. This stack is so small, that the last 2 40 ft. long containers cannot be stacked (nos. 14 and 16), as there should be only 2 piles of 40 ft. containers, containing 6 containers at most. When the first of these containers have to leave, there will thus also be no room to remarshall the containers on top to. These containers should be remarshalled to the transfer point landside.

The number of remarshallings to the transfer point landside should be at least 3, because after one 40 ft. container has been removed from the stack, there is room for one new container, but since there are 2 containers stacked upon the one wanted, the second time one of these still has to be moved to the transfer point landside. The third time, this is also the case, because although there is space available, the simulation prohibits planning containers to stack on top of incoming stack locks, which has just been created for the first container that had to be moved. However, since there are only 10 time units available per container to do remarshalling, moving the container to the transfer point quayside and moving any containers temporarily put at the transfer point landside back to their original pile, in some cases, the next container will be called for already before this is all completed. This will cause additional instances of emergency remarshalling, because containers may not be remarshalled to any pile which contains an outgoing lock. The simulation confirms this, as 4 emergency remarshalling occasions are reported.

In order to see whether the value of 3 is obtained if more time is available, we conduct another test; this time with 30 time units between the pickup times of the containers (the first container is still called for at $t = 170$, but the second at $t = 200$, etc.). When we examine the results, we find this is indeed the case here.

6.2.5 Test case 5

This test case is different from the previous ones, in that we do not use a similarly controlled setup and henceforth cannot predict the output. Instead, we do some bigger experiments, like it was done in [5], and compare the results with theirs. While is is very unlikely to have exactly matching results, if we can only show that the same kinds of relationships between the chosen algorithms and results exist, we can have greater confidence in the program. We match the experiments A0 and A from Dekker et al. [5]. The used setup is the same as for the other experiments done later in this thesis, and can be found in section 7.1.

Experiment	Random stacking Dekker et al.	SSTACk 1	SSTACk 2	SSTACk 3
Reshuffle occasions (%)	60.9	63.25	63.07	62.99
Reshuffled containers (%)	89.3	94.07	93.63	93.47
No stacking position (per 100,000)	0	0	0	0
No reshuffle position (per 100,000)	74	324	301	325
Avg. gr. pos. usage (normal) (%)	83.5	84.51	84.37	84.40
Avg. gr. pos. usage (reefer) (%)	42.9	50.82	51.03	50.93

Table 6.3: Test case 5. Comparison of the results for random stacking, obtained by Dekker et al. [5] and the SSTACk simulation program.

Experiment	Category stacking Dekker et al.	SSTACk 1	SSTACk 2	SSTACk 3
Reshuffle occasions (%)	31	43.03	42.71	42.99
Reshuffled containers (%)	46.1	70.81	70.44	70.92
No stacking position (per 100K)	0	0	0	0
No reshuffle position (per 100K)	40	51	31	43
Avg. gr. pos. usage (normal) (%)	69.6	67.76	67.19	67.28
Avg. gr. pos. usage (reefer) (%)	37.45	36.83	36.83	36.73

Table 6.4: Test case 5. Comparison of the results for category stacking, obtained by Dekker et al. [5] and the SSTACk simulation program.

Tables 6.3 and 6.4 show the results of this test case. The results of the experiments done with our SSTACk program are based on an average over 4 runs. This number was chosen to achieve a balance between taking out any effects from stochasticity, whilst maintaining acceptable run times. Some tests were done with 10 replications, but it was found there was little variability over the runs. Because stochasticity exists not only in the main program, but also in the generator, we compare the results of three arrival files, generated with the same parameters, but with different random seeds.

We can see from these tables that whereas some of the statistics closely match the previous research, some do not. The used number of ground positions is virtually equal in both programs, for both tested algorithms. The numbers of reshuffles and reshuffle occasions computed by SSTACk are also somewhat similar to those of Dekker et al., particularly for random stacking. However, SSTACk's random stacking has a much higher number of emergency reshuffle occasions ("No reshuffle position" in the table).

What also becomes clear, is that it matters little which arrivals file is chosen. All results are pretty much the same for all files. This means that further experiments can be done using only the same arrivals file, which reduces the number of parameters needed to investigate.

Assuming that both the implementation of Dekker and ours are correct, these differences have to be caused by differences between the two models. One major difference, which could explain the high values of "no position found", might be the handling of emergency reshuffles. While our model moves these containers to the transfer point landside, the model of Dekker et al. simply discards these containers entirely. This means they no longer take up any stack space, as well

Experiment	Random stacking		Category stacking	
	Dekker et al.	SSTACk 1	Dekker et al.	SSTACk 1
Reshuffle occasions (%)	60.9	63.18	31	42.99
Reshuffled containers (%)	89.3	93.91	46.1	70.77
No stacking position (per 100K)	0	0	0	0
No reshuffle position (per 100K)	74	336	40	31
Avg. gr. pos. usage (normal) (%)	83.5	84.45	69.6	67.59
Avg. gr. pos. usage (reefer) (%)	42.9	50.92	37.45	36.62

Table 6.5: Test case 5. Comparison of Dekker et al. [5] and the SSTACk simulation program, modified to discard containers which cannot be reshuffled.

as that no moves are "wasted" in moving them back and forth to the transfer point landside.

To test whether this is indeed a big factor in the difference between Dekker's results and ours, we modified the program to also discard any containers where no reshuffle could be found for. The results of these experiments are in table 6.5. The arrivals file "SSTACk 1" was used.

The results show that the effect of the different handling of emergency remarshalling does not matter. All statistics are almost the same as before. including the number of cases where no reshuffle position was available. Henceforth, it appears that the difference in handling the emergency remarshalling occasions is not a factor in explaining the difference in the results of both programs.

A second major difference between the two models is that our model contains moving ASCs, whereas the model of Dekker et al. did not: containers simply appeared in the stack, whether a crane would have been available or not. This results in ASC workloads (which were calculated at the end of each run) being sometimes over 100%, meaning that whatever was simulated, could not have been done on time.

In order to get some idea of what might be a good setup, we tried increasing the number of lanes to 29, 30 and 31. This indeed produced results similar to the ones obtained by Dekker et al., with 30 lanes giving the best match, as can be seen in table 6.6. This could indicate that the stack capacity is simply too low for the number of containers. This was found by Dekker et al., too, and they ran later experiments with 29 instead of 27 lanes. It may be the case that our arrivals list is significantly different from the one used by Dekker et al. Although the list was made to resemble it, we can never be sure how much they are alike, since that particular list is lost and had to be reverse engineered (see chapter 5). We do know that our list has a slightly higher average dwell time, which may be a cause of the higher pressure on the stack.

6.3 Conclusion

In this section, we tried to verify the simulation tool, by running some simple tests and checking the outcomes by hand, as far as possible. We also compared a more complex setup with previous research. In both cases, all tests passed. Furthermore, throughout development, informal verification methods were employed and many bugs were found and fixed. While this all does

Experiment	Dekker - 27 lanes	SSTACk - 29	SSTACk - 30	SSTACk - 31
Reshuffle occasions (%)	60,9	61,4	60,33	59,5
Reshuffled containers (%)	89,3	90,2	88	86,3
No stacking position (per 100K)	0	0	0	0
No reshuffle position (per 100K)	74	128	72	47
Avg. gr. pos. usage (normal) (%)	83,5	88,68918	81,67	92,70153
Avg. gr. pos. usage (reefer) (%)	42,9	52,88066	48,67	54,52675

Table 6.6: Test case 5. Comparison of random stacking performance in Dekker et al. [5] (27 lanes) and the SSTACk simulation program, variously with 29, 30 and 31 lanes.

not mean the program is error free, at least we now have substantial confidence in it, so that we can use it for research.

Chapter 7

Introduction to the experiments

In this chapter, we introduce the experiments which we conducted in the following chapters. First, we give the basic terminal setup used in all experiments (except where stated), in section 7.1. We then explain some very basic stacking strategies, which we will use for comparisons in further experiments. See section 7.3.

7.1 Basic terminal setup

The basic parameters are the same for all experiments. While most of these are calculated or directly taken from various sources, some have to be determined by some experiments. These parameters are discussed in section 7.2. We used the following values for the input variables described section 4.4:

Simulation parameters (following [5]):

- Simulation time: 2520 hours (i.e. 15 weeks)
- Warming up time: 504 hours (i.e. 3 weeks)

AGV/SC variables:

- Number of AGVs: 75. This value was chosen to take the AGV variable out of the equation; there are now always more than enough AGVs. This makes interpreting the results easier.
- Number of SCs: 75. This value was chosen to take the Straddle Carrier variable out of the equation; there are now always more than enough Straddle Carriers. This makes interpreting the results easier.
- AGV quay load/unload time: 0.02 hours
- SC tlp load/unload time: 0.02
- AGV travel time: 0.03 ([32], page 16/17)
- SC travel time: 0.02 ([32], page 20)

ASC variables (according to [32], page 18):

- ASC lengthwise speed: 47244,10 ft/hour (i.e. 4 m/sec)
- ASC lengthwise acceleration: 12755905,51 ft/hour^2 (0.3 m/sec^2)
- ASC widthwise speed: 9448,82 ft/hour (0.8 m/s)
- ASC widthwise acceleration: 12755905,51 ft/hour^2
- ASC lifting speed (empty): 11811,025 ft/hour (1.0 m/s)
- ASC lifting acceleration (empty): 12755905,51 ft/hour^2
- ASC lifting speed (full): 7086,615 ft/hour (0.6 m/s)
- ASC lifting acceleration (full): 12755905,51 ft/hour^2
- ASC lane pickup/setdown time: 0.001 (i.e. 3.6 sec)
- AGV-ASC pickup/setdown time: 0.001

Stack layout, following [5], except the number of lanes, which was put at 30 instead of 27 or 29, according to the verification experiments done in section 6.2.5.

- Number of lanes: 30
- Lane width: 6 containers
- Lane length: 41 TEU (including 1 TEU long reefer platform at position 37)
- Maximum height: 3 containers
- Reefer section start: position 34, end: 41

For the container arrivals, we used the file generated by the generator program using the "low" scenario (see chapter 5).

7.2 Determining the simulation parameters

Several required parameters of the simulation cannot be determined without some trial-and-error experimenting. In this section, we present the experiments to find their values. It concerns the following parameters:

- Number of replications. More replications cost more time to compute, but give a more accurate result.

- Number of lanes. Although the base case is with 30 lanes, we would like to decrease this number, since this has two advantages: 1) it decreases the time necessary to run the simulation, and 2) it makes interpreting differences caused by alterations in the algorithm easier. Of course, using a smaller stack also means using less containers. Therefore, the number of containers is reduced proportionally to the reduction in the number of lanes. This is done by drawing a random number every time a container arrives. If it is under a a specified value, the container is included; if not, it is discarded and no statistics are kept of it.

7.3 Basic algorithms

In this section we introduce the basic algorithms used for comparison in the experiments.

7.3.1 Random stacking

Random stacking is a straightforward way of determining a stacking position for a new container. Basically, the new container is placed on a randomly chosen allowed location, with every allowed location having an equal chance of being chosen. In our model, this gives us the following (general) algorithm:

1. Obtain a list (actually, we need a set which contains no duplicates) of all non-full piles of this container type/size, as well as possible empty Ground positions.
   (a) Iterate over all Ground positions in all stacking lanes.
   (b) If a Ground position contains a Pile that is not full and of the same type as the container to be stacked, add it to the set and check the next Ground position.
   (c) If the Ground position is empty, check the adjacent Ground positions (if needed to fit a container larger than 20 ft.). If stacking is possible, create a Prospective Pile and add it to the set.
2. Choose a random pile from the set and stack there, with every pile having an equal chance of being chosen (i.e. uniform distributed - this is why we only want unique piles: showing up twice would double their chances of being chosen).

While the algorithm described above would work, it would not be very fast. Especially when the stack is not very full, it will take a lot of time to obtain the list of possible locations. Therefore, we implemented another version of this algorithm, which is much faster:

1. Look at a random lane
2. Look at a random position in the lane
3. Check whether we could stack at this position
4. If so: stack here

5. If not: start again in a different lane

While this algorithm is not guaranteed to find an available location, given enough tries it probably will do so.

This algorithm can also be applied for remarshalling, with the difference that we then only want to look in the lane the container is in.

7.3.2 Levelling

This strategy is taken from the earlier work of Duinkerken et al. [7]. It is an intuitive strategy, but it does not use most of the available information. The idea is to fill lanes in layers, so that all empty ground positions are filled with containers first, before containers are stacked upon others. The stacking lane is filled from the transfer point landside on. We thus get the following steps:

1. Choose a random lane with at least one available position.
2. Search for the first empty location, from the transfer point quayside towards the transfer point landside, row for row (i.e. widthwise).
3. If found: stack there
4. If not found: search all existing piles (of the same size and type), from the transfer point landside towards the transfer point quayside, row for row, for the lowest (i.e. search lowest piles first) stack location and stack on the location found first.

This algorithm can also be applied for remarshalling, with the difference that we then only want to look in the lane the container is in.

The original algorithm mixes containers of different sizes within lanes. This is problematic, however, because due to the preference of stacking on the ground, gradually all positions get filled with 20 ft. containers. To counter this, we modified the algorithm to dedicate lanes to a certain size, and we also tried dedicating lane segments (a "row" within a lane) to a specific size, which allows for a better distribution of dedicated space per size. The problem described here does not apply to reefer containers in the current setup, since only one pile of any size can occupy a reefer section spot, and henceforth these containers are still mixed w.r.t. sizes in lanes.

7.4 Performance measures

In order to compare the different strategies, we have to define some measures of performance. The most important measure of performance is the exiting time, because if this value is too high, truck, ships and trains will have to wait longer for their containers, which could lead to significant extra costs. Although the simulation includes time spent on AGVs and straddle carriers towards the exit as exiting time, this is no problem, because of the abundance of these resources.

While the exiting time is our main performance measure, we also look at some secondary measures. These include the percentage of reshuffles, the average ASC workload, ground position usage, and entry time. For each of these measures, a lower value is better, however, a lower value for one measure could well lead to higher values for others. We look at these statistics, because they can all directly be linked to costs incurred by the terminal operator. For ground position usage, this is because more ground positions used, means more land used, which means higher costs. For the reshuffles and ASC workload, this is because extra personnel has to be hired in order to do the work. A similar argument can be made for the entry times, because lower entry times mean less work and more time for other actions.

In summary, we use the following performance measures:

1. Main performance measure: exiting time
2. Secondary performance measures:
    - Reshuffle percentage
    - ASC workload
    - Ground position usage
    - Entry time

Chapter 8

Experiments

In this chapter, we conduct some experiments with extra assumptions, which are intended to provide insight in the complex mechanisms of the stacking process, which is a bit simplified so the outcomes can be interpreted more easily. The concept of these tests is to determine what the influence on stacking performance is, in case some (normally online unavailable) data is taken into account. This is useful to know, because such information may be available in real life, although perhaps at a price. Some other assumptions include using only a single size of containers and only have containers both arriving at and departing from the quay. With these experiments we can show the effectiveness of very basic ideas.

Because we want to know what the benefit of these ideas is w.r.t. not using them, we compare the results with random stacking and levelling (with dedicated lane segments per size; note that this does not matter since we only use 20 ft. long containers.)

8.1 Benchmark tests

In this section, we test the normal Levelling and Random Stacking algorithms against the same terminal setups used in the other experiments in this chapter. We use lane length 34, 6 or 8 lanes and maximum pile height 3 or 4. We only use normal 20 ft. containers, and in some cases only sea-sea containers (i.e. containers arriving and leaving at the quay). This is summarised in table 8.1.

Parameter	Values used
Containers allowed	Only sea-sea (20 ft.) / both sea-sea and other (20 ft.)
# Lanes	6 / 8
Max. pile height	3 / 4

Table 8.1: Parameters varied in the benchmark experiments.

8.1.1 Hypotheses

Since the Levelling algorithm maximises ground position usage, we expect this value to be very high. This in turn means that the average pile height is lower, resulting in fewer reshuffles than with random stacking. The lower pile heights will also mean that there will be little (if any) difference between stacking 3 containers compared with stacking 4, because a pile would only reach that height in circumstances of extreme stacking position shortage.

While Levelling will lead to long ASC travel times and thus to long exiting times, we suspect random stacking will not do much better, because it too puts containers all over the lane. The increased number of reshuffles will probably lead to even longer exiting times.

8.1.2 Results

See table 8.2 for the results of the experiments with all normal 20 ft. containers, and table 8.3 for the results using only sea-sea containers. We can see that for every setup tested, the exiting times and reshuffle percentages are lower for levelling than for random stacking. We also see that results for the 4-high stacks are worse than the 3-high stacks, in terms of exiting times, but not in terms of reshuffle percentages.

	Reshuffle occasions	Reshuffled containers	GP usage	Stack usage	ASC workload	Avg. exiting duration (quay)	Avg. exiting duration (tlp)	90% exiting duration (quay)	90% exiting duration (tlp)
Rnd St. 6L 3H	69.44%	107.69%	89.56%	75.00%	62.70%	0.73	0.47	2.09	1.23
95% CI HW	0.0011	0.0017	0.0005	0.0000	0.0003	0.0084	0.0055	0.0477	0.0298
Rnd St. 8L 3H	65.11%	97.37%	79.12%	56.13%	49.84%	0.24	0.20	0.46	0.34
	0.0011	0.0015	0.0006	0.0000	0.0003	0.0023	0.0017	0.0060	0.0049
Rnd St. 6L 4H	69.32%	134.11%	84.68%	56.42%	69.41%	1.83	1.15	5.48	3.64
	0.0013	0.0027	0.0003	0.0000	0.0003	0.0201	0.0123	0.0634	0.0778
Rnd St. 8L 4H	65.77%	118.35%	76.14%	42.13%	54.38%	0.41	0.30	0.99	0.61
	0.0012	0.0026	0.0006	0.0000	0.0003	0.0047	0.0030	0.0166	0.0171
	Reshuffle occasions	Reshuffled containers	GP usage	Stack usage	ASC workload	Avg. exiting duration (quay)	Avg. exiting duration (tlp)	90% exiting duration (quay)	90% exiting duration (tlp)
Levelling 6L 3H	62.92%	81.10%	99.60%	74.92%	59.44%	0.48	0.33	1.30	0.76
95% CI HW	0.0009	0.0012	0.0000	0.0000	0.0003	0.0056	0.0034	0.0186	0.0110
Levelling 8L 3H	52.73%	52.74%	99.57%	56.11%	46.15%	0.17	0.16	0.28	0.25
	0.0009	0.0009	0.0000	0.0000	0.0004	0.0006	0.0007	0.0015	0.0017
Levelling 6L 4H	62.20%	80.60%	99.35%	56.26%	63.17%	0.84	0.53	2.52	1.44
	0.0008	0.0009	0.0001	0.0000	0.0003	0.0067	0.0046	0.0385	0.0222
Levelling 8L 4H	52.63%	52.64%	99.54%	42.09%	48.53%	0.21	0.19	0.38	0.31
	0.0012	0.0012	0.0000	0.0000	0.0004	0.0012	0.0013	0.0035	0.0025

Table 8.2: Results of benchmark tests with all normal 20 ft. containers.

	Reshuffle occasions	Reshuffled containers	GP usage	Stack usage	ASC workload	Avg. exiting duration (quay)	90% exiting duration (quay)
Rnd St. 6L 3H	64.68%	96.63%	78.55%	55.52%	47.57%	0.26	0.53
95% CI HW	0.0012	0.0020	0.0005	0.0000	0.0004	0.0026	0.0118
Rnd St. 8L 3H	58.73%	83.74%	68.41%	41.62%	37.47%	0.17	0.26
	0.0007	0.0011	0.0006	0.0000	0.0002	0.0005	0.0013
Rnd St. 6L 4H	65.32%	117.36%	75.68%	41.68%	51.93%	0.46	1.14
	0.0016	0.0041	0.0007	0.0000	0.0004	0.0035	0.0230
Rnd St. 8L 4H	59.11%	97.31%	66.72%	31.22%	40.38%	0.21	0.36
	0.0014	0.0028	0.0008	0.0000	0.0002	0.0011	0.0031
	Reshuffle occasions	Reshuffled containers	GP usage	Stack usage	ASC workload	Avg. exiting duration (quay)	90% exiting duration (quay)
Levelling 6L 3H	55.17%	55.47%	99.43%	55.49%	44.35%	0.18	0.31
95% CI HW	0.0010	0.0010	0.0000	0.0000	0.0003	0.0011	0.0025
Levelling 8L 3H	31.78%	31.78%	97.60%	41.61%	33.76%	0.13	0.19
	0.0007	0.0007	0.0002	0.0000	0.0002	0.0002	0.0006
Levelling 6L 4H	55.09%	55.40%	99.39%	41.63%	46.68%	0.23	0.45
	0.0013	0.0013	0.0000	0.0000	0.0003	0.0012	0.0073
Levelling 8L 4H	31.66%	31.66%	97.58%	31.21%	35.41%	0.14	0.21
	0.0009	0.0009	0.0002	0.0000	0.0002	0.0003	0.0006

Table 8.3: Results of benchmark tests with all normal 20 ft. sea-sea containers.

8.1.3 Discussion

The levelling algorithm consistently outperforms random stacking. This is probably caused by the fact that levelling minimises pile hight, thereby reducing the number of reshuffles, which in turn leads to lower exiting times. A surprising result is the higher exiting times for higher allowed stacks. For random stacking, this is partially due to the algorithm, which can now make even worse decisions by putting 4 containers on top of each other when it is not necessary. However, levelling does not do this, so there has to be another explanation. We suspect it is due to the fact that the crane now has longer lifting times, because it is higher. This is also reflected in the higher ASC workload values. This suspicion is also confirmed by the fact that reshuffle percentages have not increased at all, for the higher stacks, using levelling.

The confidence intervals show that the results are significant, so this is an indication we have run enough replications.

8.2 Experiment 1 - LDT

The first idea is to test the influence of knowing the exact departure times of all containers. This can be exploited by only stacking containers on top of other containers, if the new container departs at an earlier time than the one below it, i.e. residence time stacking. We also differentiate between containers both arriving and leaving at the quay and other containers (i.e. containers arriving or departing at the truck loading point). The containers both arriving and leaving at the quay should be stacked close to the transfer point quayside, because every meter they move towards the transfer point landside is a waste.

To simplify things, we start with only sea-sea containers of a single size.

The dwell time, or better the departure time of a container, can be used when determining where to stack it. Containers departing before the containers below them will never lead to a reshuffle. We would like to exploit this to the maximum and stack as high as possible, because this means other positions remain free for other containers which depart later. The first priority when searching for a place to stack is thus to find the piles which have such a container on top.

Second, and again in the interests of keeping options open, we want to find that position where the difference between departure times is as small as possible, since this means that there are more possibilities for stacking other containers on top of them, using as little space as possible. We thus select from the found piles the one where the difference in departure times would be smallest.

If we can find no pile with a container on top departing after the new container, we have to stack it elsewhere. Preferably, we do this on the ground, so no reshuffles will occur. From the available positions on the ground, we want to stack it as close to the transfer point quayside as possible, so that travel times are minimised.

Should we still not be able to find a position, we have to stack the container on an existing pile, rendering a reshuffle inevitable. To minimize the number of reshuffles, we place the container on the highest pile available, so that no or few containers can be stacked on top of them, each

Parameter	Values used
Containers allowed	Only sea-sea / both sea-sea and other
# Lanes	6 / 8
Max. pile height	3 / 4
Mixed piles?	yes / no
Departure times	Real (perfect information) / expected

Table 8.4: Parameters used experimenting with Experiment 1.

of which would lead to another reshuffle. In case several of these piles are available, we use the pile as close to the transfer point as possible, in order to minimise travel time for the ASC.

In summary, we use the following algorithm, which we shall call "LDT" *(Levelling with Departure Times)* from now on:

1. Stack the new container (departing at $T = T_n$) at that pile, where the top container departs at time $T = T_o$ and $T_o - T_n$ is minimal (but positive) and on which the container may technically be stacked (i.e. pile is not full).
2. If no position was found yet: stack the container on an empty ground location. Sea-sea containers are to be stacked as close to the transfer point quayside as possible.
3. Stack at a pile of the highest height available, as close to the transfer point as possible.

Sea-land and land-sea containers do not prefer any part of the lane, since they have to traverse it in full anyway. However, because sea-sea containers prefer the sea (quay) side of the lane, sea-land containers should be stacked away from them, as close to the transfer point landside as possible, when the two types are both included in an experiment. However, because we select on departure times, the two types may become mixed. If only sea-sea containers are included, the algorithm automatically stacks the containers near the transfer point quayside, but with land containers included, it may not do this. Since this removes the advantage of stacking near the quayside, we can choose to separate the piles, so that sea-sea containers may not be stacked on land-sea containers and vice versa. However, this means there are less options for optimizing residence times, it remains to be seen which is best.

8.2.1 Experimental setup

To test this algorithm, we used the settings of section 7.1, with the exception of the stack size. We do not include reefer containers, or any other containers over 20 ft. in length. Other parameters were varied, as can be seen in table 8.4. Note that allowing mixed piles does not make much sense when only sea-sea containers are used.

In this experiment we are particularly interested in the value of the perfect information w.r.t. departure times. We therefore compare the results to random stacking. We would also like to know whether it is better to allow mixed piles or not. The difference is measured by looking at the times it takes for a container to enter and leave the stack.

8.2.2 Comparison setups

We compare the LDT algorithm with random stacking, levelling and a modified version of random stacking (which we refer to as RS-DT, for random stacking - departure times), in which the algorithm searches for a random pile with the top container's departure time being after the new container's departure time. If no such pile is found, the container is stacked randomly, according to the random stacking algorithm. We use this to see which part of the differences between ordinary random stacking and the LDT algorithm is caused by the "random" part and which part is caused by the lack of perfect information.

8.2.3 Hypotheses

Hypothesis 1.1 The LDT stacking algorithm will have a lower number of reshuffles, a lower exiting time and a lower ASC workload than the benchmarks RS and LEV.

Hypothesis 1.2 The RS-DT stacking algorithm will have a better performance than RS, but worse than LDT.

Hypothesis 1.3 Mixing piles in the LDT stacking algorithm will lead to less reshuffles for smaller stacks (compared to not mixing).

Hypothesis 1.4 Mixing piles in the LDT stacking algorithm will lead to higher exiting times (compared to not mixing).

Because of the great advantage of perfect information w.r.t. departure times, we expect to see a very big improvement for relevant statistics, compared to random stacking and levelling. In particular, we look at the reshuffle percentage (which we expect will be lower for this algorithm), time to exit (will also be lower), ASC workload (will also be lower, as it is related to the previous ones), stack usage (will be slightly lower due to containers exiting quicker) and ground position usage (will be much lower than with levelling, which maximises ground usage. We cannot say in advance how it compares with random stacking). We expect these effects to occur, whatever the stack layout. However, with a smaller stack, the number of available options is decreased and henceforth some more bad decisions have to be made, which will make the improvement smaller.

The effects of the RS-DT algorithm should be similar to those of LDT w.r.t. random stacking and levelling, because of the extra information. However, because the pile choosing process is still very basic, it probably won't perform as good as LDT.

Regarding mixed and unmixed piles, it is difficult to make a prediction of the results. The advantage of not mixing piles is that containers are stacked closer to their exiting point, which should lead to a shorter exiting time. However, it does decrease the available number of stacking options, which brings with it the risk of more imperfect piles and hence more reshuffles and a larger exiting time. Which of these effects will be dominant is probably dependent on the size of the stack. If there is a lot of space, the loss of options with unmixed piles does hardly matter. When there is little space, the extra possibilities will probably be very useful.

The experiments with expected, rather than real, departure times should still be better than random stacking and levelling, i.e. the same effects should occur as with perfect information. We do expect these effects to be somewhat weaker, since the information is less reliable and hence some bad decisions are likely to be made.

8.2.4 Results

Table 8.5 shows some results from this experiment, using all (non-reefer) 20 ft. containers and a relatively small stack of 6 lanes of size 34x6 and maximum height 3. As expected, the LDT algorithm in it various forms outperforms the random stacking and levelling benchmarks. Less reshuffles occur and exiting times and ASC workloads are lower. Stack usage is not shown, but we found it is decreased slightly with the LDT algorithms.

Ground position usage is also lower. Curiously, it is much lower in the experiments with imperfect expected departure times than with real departure times.

The relative performance of the modified random stacking algorithm is as predicted: better than the benchmark but worse than LDT. It is also of note that the modified random stacking algorithm (using real departure times), outperforms the LDT when the latter is using expected departure times. Using expected data, rather than actual, leads to a big performance drop for LDT.

The effects of mixed piles seem to be unclear. When using actual data, mixed piles lead to less reshuffles but an increased exiting time. However, using expected data, they lead to more reshuffles (and also an increased exiting time). These effects are also found in the other stack layouts, albeit somewhat weaker.

In larger stacks, there is little improvement in the results for expected times, unlike those for actual times. Moreover, the results for expected times actually deteriorate when going from 6 lanes, 3 height to 6 lanes, 4 height.

More results from this experiment can be found in appendix C.1.

8.2.5 Discussion

From the results, it becomes clear that LDT's departure times have a big impact, even when they are not exactly known. Even partially random stacking, using departure times only to a limited extent, leads to big improvements across the board. Still, there remains a big gap between the results of expected and actual departure times, especially in bigger stacks, where little improvement, if any, is seen w.r.t. smaller stacks. This is most dramatically the case when going from height 3 to 4 with 6 lanes, where performance actually drops, despite an increase of stacking options. This is probably due to any mistakes made being punished more heavily with extra reshuffles at higher stack levels.

Using mixed piles seems to be not such a good idea. In some cases, there is an improvement in the number of reshuffles (as was expected), but in others there is none. In all cases, mixed piles lead to longer exiting times, so all in all they cost more than they earn.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.92%	81.10%	99.60%	59.44%	0.48	0.33	1.30	0.76
Random stacking	69.44%	107.69%	89.56%	62.70%	0.73	0.47	2.09	1.23
Modified RS (real)	35.27%	52.10%	82.94%	54.44%	0.37	0.26	0.94	0.56
LDT (unmixed)	10.20%	15.21%	87.39%	49.93%	0.21	0.17	0.42	0.30
LDT (mixed)	9.09%	13.62%	86.64%	49.62%	0.22	0.18	0.44	0.32
Modified RS (exp)	56.10%	82.60%	77.85%	57.98%	0.41	0.31	1.05	0.71
LDT (unmixed, exp)	47.93%	70.20%	79.15%	54.67%	0.29	0.24	0.63	0.46
LDT (mixed, exp)	48.54%	71.62%	77.96%	55.83%	0.36	0.29	0.83	0.60

Table 8.5: Results of experiment 1. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 3. "exp" denotes experiments done with expected departure times, rather than actual. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

8.2.6 Conclusions

It is clear that knowing the departure times is very valuable, and using these, according to the LDT algorithm, if they are known gives great advantages for stacking. Even imperfect departure times are useful, so an estimate can also be used to improve efficiency. Sea-sea containers and other containers should not be mixed within a pile, it increases exiting times and reshuffles and thus lowers efficiency.

Hypothesis 1.1 Confirmed

Hypothesis 1.2 Confirmed

Hypothesis 1.3 Rejected

Hypothesis 1.4 Confirmed

8.3 Experiment 2 - LDT with departure time classification

In this experiment, we go back to the idea of using (expected) departure times (LDT from experiment 1, section 8.2). This time, however, we use a different method of determining the expected departure time.

In a real-world situation, it is rare for a terminal operator to know exactly when a container will arrive or depart. Even knowing it with a margin of error of one hour is usually too much to ask for. We thus use a method which takes this into account.

At any time, a container has a residence time left in the stack. We divide this residence time into 5 classes. The boundaries of these classes are calculated from the arrivals file using Matlab's `prctile` function, which calculates the percentiles of a data set (in our case, the quintiles or the 20th, 40th, 60th, 80th and 100th percentiles). This gives us 5 classes of almost equal size, for the initial residence times at least. The classes used can be seen in table 8.6.

Class	From	To
1	0	68.1977
2	68.1977	79.4249
3	79.4249	92.4209
4	92.4209	111.5949
5	111.5949	∞

Table 8.6: The 5 classes of container departure times used in experiment 4. The values represent hours of dwell time left. The maximum time any container in the used arrivals file will stay is 192.2115 hours.

We use the algorithm from experiment 1 for this experiment too, only when the time difference is calculated we do not use the actual time or the expected time from the file, but instead use the class value from table 8.6 (based on actual departure time). This means that lower classes will be stacked on top of higher classes, thereby ensuring no reshuffles occur, unless no suitable pile or ground position could be found.

Note that a container's class will change over time as its departure time comes nearer. This means that the lower classes will be more prevalent in the stack, because every high class will at one time become a low class. This is just like with ordinary times, so it is not a problem.

8.3.1 Experimental setup

To test this algorithm, we used the settings of experiment 1 (see section 7.1), because we want to compare the different method of estimating departure times with the original and with perfect knowledge.

8.3.2 Comparison setups

We compare this algorithm to the entire experiment 1 (normal LDT), including random stacking and modified random stacking.

8.3.3 Hypotheses

Hypothesis 2.1 The LDT-DTC stacking algorithm will have a lower number of reshuffles, a lower exiting time and a lower ASC workload than LDT-exp, but higher than LDT.

The reduction in efficiency when using expected times with normal LDT was rather high, because some wrong information was taken to be perfect. In this case, we take correct information and treat it as imperfect. While this means less mistakes should be made, since a difference in class means a similar difference in departure time, the algorithm may have a problem with small stacks, where space is in short supply and some suboptimal decisions may have to be made. In large terminals, we expect that the LDT with classes of departure times will perform almost as good as normal LDT with real departure times.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.92%	81.10%	99.60%	59.44%	0.48	0.33	1.30	0.76
Random stacking	69.44%	107.69%	89.56%	62.70%	0.73	0.47	2.09	1.23
Modified RS (real)	35.27%	52.10%	82.94%	54.44%	0.37	0.26	0.94	0.56
Modified RS (class)	54.07%	77.55%	86.25%	59.14%	0.46	0.32	1.22	0.74
Modified RS (exp)	56.10%	82.60%	77.85%	57.98%	0.41	0.31	1.05	0.71
LDT (unmixed)	10.20%	15.21%	87.39%	49.93%	0.21	0.17	0.42	0.30
LDT (unmixed, exp)	47.93%	70.20%	79.15%	54.67%	0.29	0.24	0.63	0.46
LDT (unmixed, class)	12.20%	18.08%	88.67%	50.42%	0.22	0.18	0.44	0.32

Table 8.7: Results of experiment 2. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 3. "exp" denotes experiments done with expected departure times, rather than actual. "Class" denotes experiments done using the LDT algorithm with departure time classification. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

8.3.4 Results

Some results are in table 8.7. For simplicity, we took out the results for mixed piles, since these performed like in the previous experiment and they only complicate the table.

These are the results for the relatively small 6 lane, 3 height terminal. Still, the LDT with classes of departure times performs much better than normal LDT with expected departure times. Exiting times and ASC workloads are lower and less reshuffles occur. The random stacking version of the departure time classes algorithm only manages small improvements in these areas, compared to the normal modified random stacking algorithm. Still these differences are out of the 95% confidence intervals (not shown here) and are statistically significant.

8.3.5 Discussion

Using the suggested 5 classes gives a very good result and a very good approximation of the results of actual departure times. In large stacks, there is barely any difference between the two. The performance is also much better than that of expected times. This is probably due to the fact that the underlying data is perfect. Still, much of this data is not used by the algorithm, due to the use of the classes.

8.3.6 Conclusion

LDT-DTC produces results which are almost as good as normal LDT. This means that the classification system is a good idea to use, if a reliable class can be assigned to every container. There is little need to improve the residence time knowledge even further.

Hypothesis 2.1 Confirmed

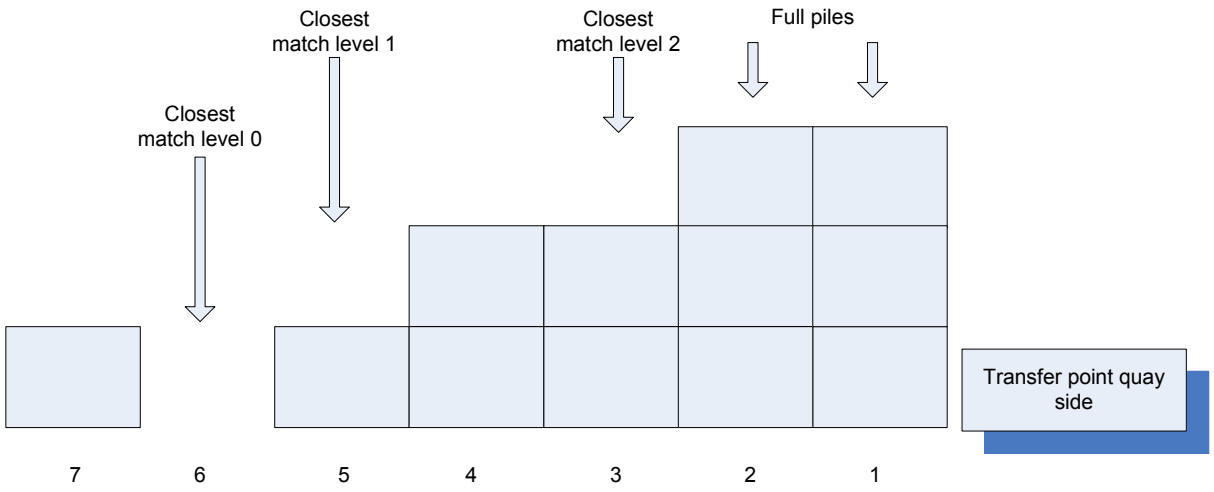

Figure 8.1: Example case for experiment 3. Maximum stacking height is 3 containers. The 2 piles to the right are closest to the transfer point, but are full. Three other options are available though.

8.4 Experiment 3 - TVR (Travelling distance vs. reshuffling)

In this experiment, we do not use residence time knowledge, but rather try to optimise the selection of a location in some other way.

Suppose we only have sea-sea containers. As said previously, these containers should be stacked as close to the transfer point quayside as possible, because they both arrive and leave the lane there. Levelling all the way down the lane would thus not be a good idea; we want to stack as high as possible at the beginning of the lane, and leave the end of the lane empty, if possible. Of course, we could just stack new containers as close to the transfer point as possible, by stacking them on top of each other, but this has the unwanted side effect of leading to reshuffles, because the order in which containers arrive and leave is mostly random.

However, since we also don't want to level too much, a compromise solution should be better. Whenever a container arrives at a lane (maximum stacking height n), we can say there are n possibilities to stack it: at every possible height layer, as close to the transfer point quayside as possible. For example, if $n = 3$, we may get the scenario from figure 8.1. The ground positions closest to the transfer point are full, but the next one is only stacked 2 out of 3 high. This is the best position from the distance viewpoint, but it has a high risk of leading to reshuffles, with two containers under it. The next position has the same risk, but it is further away, so therefore not as good, and we henceforth discard it for this decision.

We then encounter a pile with only one container. This means that the risk of reshuffling is lower, but unfortunately the pile is also further away from the transfer point. Still, it may be a good candidate to investigate. The sixth ground position is empty, and while it is still further away from the transfer point, there is no risk of causing reshuffles. This is the final candidate for stacking, because although there are more ground positions ahead, none of them is better in terms of reshuffling risk than the ones we already selected, and they are all worse in terms of distance to travel.

Now that we found the "best" (note that we do not use residence times or categories here, these factors seriously complicate matters) three piles, we need to choose which one is best, i.e. which one has the lowest associated cost in time. Time costs consist of two parts: extra ASC driving time and time due to reshuffling. The latter will often be greater, but it is not certain it will occur.

The ASC driving time can easily be calculated using the distance from the transfer point to the selected ground position and the ASC's lengthwise and widthwise speeds. Lifting times can also be calculated, since we know exactly at which level the container is and will be stacked. The cost of reshuffling is difficult to determine, because it is not known in advance whereto a container would be reshuffled. We assume this to be a (configurable) time period of driving away. The lifting times are also not known, so we estimate the lifting time to be that of a container in the middle of a pile (i.e. at the level of half the maximum height). The cost of a reshuffle is then multiplied by the expected number of reshuffles that putting the container at a particular place will generate. This gives the following cost function to minimize:

$$\min[2 \cdot TT_{qp} + L_{qp} + (TT_r + L_r) \cdot ER(n)] \tag{8.1}$$

Where TT_{qp} is the travel time from the transfer point quayside to the target pile, TT_r is the (fixed) travel time for reshuffles, L_{qp} and L_r are the lifting times per container (including pickup, lift up, lift down and set down) for the container itself and reshuffles, respectively, and $ER(n)$ is the expected number of reshuffles. The travel time to the pile has to be doubled because containers need to go back to the transfer point at one time; reshuffling can lead to a position closer to the transfer point, so we do not double that time.

This leaves one problem: calculating $ER(n)$, the expected number of reshuffles. We can estimate this by assuming that whenever a container in a pile is demanded, there is an equal probability for any of the containers in it to be selected. This means every container has a probability of $(1/n)$ to be selected, in our example $(1/3)$. If the top container is chosen, there are no reshuffles, but the second container leads to one reshuffle (the one on top) and choosing the third container would lead to two reshuffles (both the containers on top of it), and so on. This gives the following equations for the number of expected reshuffles in a pile (equation 8.2):

$$ER(n)_1 = \sum_{i=1}^{n} \frac{1}{n} \cdot (i-1) \tag{8.2}$$

$$= \frac{n-1}{2} \tag{8.3}$$

However, this formula does not take into account the possibility of a container being chosen, leading to a reshuffle, and then another container being chosen, leading to another reshuffle. This situation occurs in piles of height 4 and larger. For example, in a pile of 4, the second container from the top may be chosen, leading to a reshuffle of the top container. Later, the bottom container (of the two containers left by then) may be chosen, leading to a reshuffle of the top (originally third) container. We thus expand equation 8.2 as follows:

$$ER(n)_2 = \frac{n-1}{2} + ERX(n) \tag{8.4}$$

Where $ERX(n)$ is the expected number of additional reshuffles for pile height n. $ERX(n)$ is defined as:

$$ERX(n) = \begin{cases} \sum_{i=1}^{n-2} \frac{1}{n} \cdot ER(i)_2 & \text{if } n \geqq 4 \\ 0 & \text{otherwise} \end{cases} \tag{8.5}$$

This means that the number of expected reshuffles is calculated with a recursive function, using the probability of occurrence and the expected number of reshuffles of the smaller resulting pile.

However, a new container can, by itself, only generate one *extra* reshuffle, at most. Any other reshuffles were already in the pile when the container was stacked, or are added later. This means that we have to calculate the probability $P(nr)$ of the new container leading to an extra reshuffle. Since we assume every container in a pile has an equal chance of being chosen, this gives the following formula:

$$P(nr) = \frac{n-1}{n} \tag{8.6}$$

We use this probability as $ER(n)$ in equation 8.1.

Summarising, we use the following algorithm:

1. Select for every possible stacking level the (available) position closest to the transfer point quayside, if any position is available.
2. Calculate the costs of every position, according to equation 8.1 (using equation 8.6).
3. Select the position with the lowest cost and stack there.

While this algorithm can only stack sea-sea containers of a single type, it is easy to extend the algorithm for other containers. As discussed in the previous experiment, sea-land and land-sea containers should be stacked as close to the transfer point landside as possible. We can determine costs in the same way. Containers of different sizes can be put in different lane segments, so that different sizes do not mix (which would limit the possibilities of stacking and greatly reduce insight in what is happening).

8.4.1 Experimental setup

To test this algorithm, we used the settings of section 7.1, with the exception of the stack size, just like in Experiment 1. We do not include reefer containers, or any other containers over 20 ft. in length. Other parameters were varied, as can be seen in table 8.8.

In this experiment we are particularly interested in the value of putting sea-sea containers close to the transfer point quayside. We therefore compare the results to random stacking. We would

Parameter	Values used
Containers allowed	Only sea-sea / both sea-sea and other
# Lanes	6 / 8
Max. pile height	3 / 4
Reshuffling movement penalty	-0.03, -0.01, 0, 0.003, 0.01, 0.02, 0.03, 0.04, 0.05, 0.1

Table 8.8: Parameters used experimenting with Experiment 3.

also like to know what the effects of different penalties are. The difference is measured by looking at the times it takes for a container to enter and leave the stack.

8.4.2 Comparison setups

We compare this algorithm with random stacking, levelling and a modified version of random stacking, which we shall refer to as TPRL (Transfer Point Random Level), in which the algorithm chooses one of the possibilities offered (i.e. it randomly selects the level where the container is to be stacked). This means containers will be near the quay, and since we know one of the positions is the "best" one, we can see the influence of the complicated calculations, when we compare it with chance.

8.4.3 Hypotheses

Hypothesis 3.1 The best TVR stacking algorithm will have a lower number of reshuffles, a lower exiting time and a lower ASC workload than the benchmarks RS and LEV.

Hypothesis 3.2 The TPRL stacking algorithm will have a worse performance than TVR.

Hypothesis 3.3 The TPRL stacking algorithm will have a worse performance than RS and LEV.

Hypothesis 3.4 The TVR stacking algorithm is equal to LEV with a high penalty and only sea-sea containers.

Hypothesis 3.5 The best TVR stacking algorithm will lead to a monotone decreasing average pile height away from the transfer point quayside, when using only sea-sea containers.

Hypothesis 3.6 The TVR stacking algorithm, with very low penalties, will have a worse performance than RS, LEV and TVR and TPRL.

Again, we predict this algorithm will outperform the basic random stacking and levelling algorithms, when a somewhat realistic penalty value is chosen (i.e. not a negative value). We again look at the reshuffle percentage (which we expect will be lower for this algorithm), time to exit (will also be lower), ASC workload (will also be lower, as it is related to the previous ones), stack usage (will be slightly lower due to containers exiting quicker) and ground position usage (will be somewhat lower than with levelling, which maximises ground usage).

The ground usage depends on the reshuffling movement penalty applied. A higher penalty will lead to more containers being stacked on the ground and this leads to a higher ground position

usage. A high penalty, when stacking only sea-sea containers, would lead to the algorithm behaving as normal levelling. Stacking land containers as well would not lead to levelling, since levelling stacks from one side of the lane on only. This would also mean that the effect of allowing to stack 4 containers on top of each other, rather than 3, would be almost completely gone (there is still a minor effect on crane lifting times). Low penalties, on the other hand, would lead to a very low ground position usage, with high piles near the transfer points.

In any case, when only sea-sea containers are included, the algorithm will stack containers mostly next to the transfer point quayside and will stack less and less containers further away. This would mean that the average pile height would be decreasing in a monotone way, when going away from the transfer point.

We expect the TPRL algorithm to perform worse than the original random stacking (and, for that matter, all other algorithms tested in this section, except those with a very low (i.e. negative) penalty), especially when there is a lot of space in the stack. This is because TPRL will too often build high piles, while the other algorithms place more containers on the ground, which leads to less reshuffles.

8.4.4 Results

Results using a 6-land, 3-high terminal can be found in table 8.9. In this table, we can see that the TVR algorithm performed better than random stacking on all statistics, but compared to basic levelling the differences are minute. Even the best TVR version, in this experiment with a moving penalty for reshuffles of 0.0 hours, scores only marginally less reshuffles and slightly lower exiting times. It should be noted, however, that these results are still statistically significant and outside the 95% confidence interval.

The very low penalty value of -0.03 hours results in very low ground position usage, meaning high piles. This in turn leads to more reshuffles and higher exiting times.

What further becomes clear is that penalties from -0.01 and higher yield almost the same results. Lower penalties give different (worse) scores (with the low ground position usage as predicted). With higher penalties, the results approach the benchmark result of levelling, but a penalty of 0.03 does not mean that the algorithm is the same yet. Extremely high penalties (e.g. 1 hour) give exactly the same results as levelling, when stacking only sea-sea containers. For larger stacks, the same behaviour is observed, although the value of -0.01, from whereon the results are very similar, appears to be somewhat higher for larger stacks. Appendix C.5 contains more results on these experiments.

The TPRL algorithm performs slightly worse than levelling, especially on larger stacks, w.r.t. exiting times and reshuffle occasions, but not by the amount we expected. Moreover, it performs better than normal random stacking. The number of actual reshuffles is higher though, because the round position usage is lower and piles are higher.

The effect of the penalty value on the pile heights along the length of the lane is illustrated in figure 8.2. If the penalty is zero (left), then the average pile height is the same for both types of containers up to position 20. Beyond that, there are no sea-sea containers which means that the penalty value causes the two types of containers (sea-sea versus land-sea/sea-land) to be

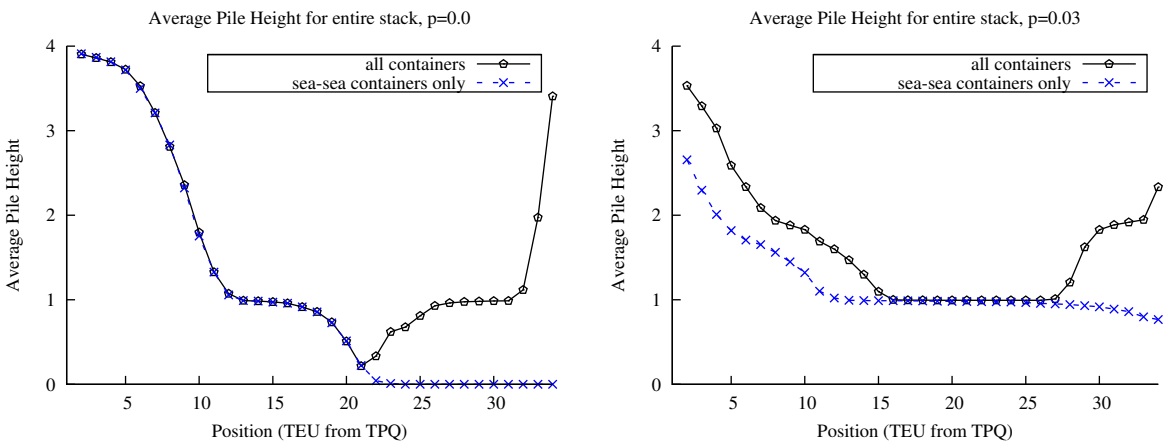

Figure 8.2: Time-average pile height for the entire stack for penalty value 0 (left) and 0.03 (right). Terminal layout: 8 lanes, 34 long, 6 wide, 4 high.

spatially separated. If we increase the penalty value, to e.g. 0.03, we see more sea-sea containers being moved along the entire length of the lane. The line for all containers shows that the sea-sea containers can no longer be stacked at the landside: more are now stacked towards the quayside and the average pile height increases.

8.4.5 Discussion

The TPRL algorithm performs not as bad as expected. This is likely partially due to the forced stacking near the transfer points, which leads to lower exiting times, even compared to levelling. However, reshuffles are also consistently lower than with random stacking. A possible explanation lies in the fact that, on average, there is about a 33% probability (25% for stacking of height 4) of TPRL stacking in any of the possible stack layers. With random stacking, however, the probability for stacking on top of a high pile increases with an increased stack usage. In the biggest stack tested (8 lanes, 4 high, which naturally had the lowest stack usage), random stacking had a ground position usage of 76.1% (1632 piles on average) and a stack usage of 42.1% (average 2748.3 TEU in the stack at any time). This means piles were, on average, 1.68 (out of 4) containers high. Random stacking thus had a $(1 - 0.761 = 23.9\%)$ probability of selecting an empty ground position and 76.1% probability of selecting one of another height (which was, on average, 1.68). This gives the expected pile height of random stacking as $(0.239 * 0 + 0.761 * 1.68 \approx 1.28)$. TPRL, on the other hand, had 25% probability of choosing an empty ground position, and 75% of choosing an existing pile, of which the average height was $(1 * 0.25 + 2 * 0.25 + 3 * 0.25 = 1.5)$. This gives the expected pile height stacked upon as $(0.25 * 0 + 0.75 * 1.5 = 1.125)$. Since this is a lower number, the expected number of reshuffles caused is also lower for TPRL than for random stacking. For smaller stacks, the probability for random stacking to stack on an existing pile only becomes greater, so this explanation applies there as well.

A reshuffle movement penalty of -0.01 hours appears to not lead to very bad results. There are only slight drops in performance compared to the higher penalties. We suspect this is due to the

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.92%	81.10%	99.60%	59.44%	0.48	0.33	1.30	0.76
Random stacking	69.44%	107.69%	89.56%	62.70%	0.73	0.47	2.09	1.23
TVR (penalty 0.0)	62.06%	95.47%	99.28%	57.02%	0.40	0.27	1.03	0.62
TPRL	62.82%	95.35%	96.41%	57.78%	0.43	0.29	1.13	0.66
-0.03	79.49%	130.91%	75.84%	57.81%	0.65	0.43	1.85	1.09
-0.01	65.59%	105.06%	93.69%	55.89%	0.44	0.30	1.17	0.70
0	62.06%	95.47%	99.28%	57.02%	0.40	0.27	1.03	0.62
0.003	62.38%	94.27%	99.44%	57.15%	0.40	0.27	1.02	0.61
0.01	62.95%	92.26%	99.57%	57.27%	0.40	0.27	1.00	0.61
0.02	63.39%	89.53%	99.63%	57.45%	0.40	0.27	1.01	0.61
0.03	63.63%	86.83%	99.65%	57.60%	0.40	0.28	1.02	0.61
0.04	63.41%	84.85%	99.65%	57.66%	0.40	0.27	0.99	0.61
0.05	63.41%	83.69%	99.65%	57.73%	0.40	0.28	1.03	0.61
0.1	63.27%	81.96%	99.64%	58.20%	0.43	0.29	1.12	0.65
1	63.18%	81.66%	99.62%	58.48%	0.45	0.30	1.18	0.68
100	63.18%	81.66%	99.62%	58.48%	0.45	0.30	1.18	0.68

Table 8.9: Results for experiment 3 (TVR). A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 3. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

feature of the algorithm which estimates the lifting time for reshuffles (L_r in equation 8.1). We found that this variable take on the value of 0.019 hours for a 4-high stack. When the penalty of -0.01 is added, this still leaves a sizable reshuffling penalty of 0.009 hours, which is usually more than the cost of driving a little further, which the algorithm chooses to do. Hence, the value of -0.01 hours, which is not possible to have as a travel time in reality, still gives acceptable results.

The results show that different reshuffle movement penalties lead to different outcomes. Also, the best value is not the same for every stack configuration. See table 8.10 for an overview of the best penalties per setup we found in the experiments. More specifically, a maximum height of 3 requires a lower penalty than one of height 4.

8.4.6 Conclusion

TVR performs better than the benchmark tests, so when no residence time information is available, it is a good strategy to use. TPRL is also better than normal random stacking, but TVR yields much greater benefits and, since it requires no extra information, is the preferred option.

Hypothesis 3.1 Confirmed

Hypothesis 3.2 Confirmed

Hypothesis 3.3 Rejected

Setup	Best penalty
5 lanes, 5 high (all)	0.04
6 lanes, 3 high (all)	0.01
6 lanes, 4 high (all)	0.04
8 lanes, 3 high (all)	0.003
8 lanes, 4 high (all)	0.01
5 lanes, 5 high (sea)	0.04
6 lanes, 3 high (sea)	0.01
6 lanes, 4 high (sea)	0.02
8 lanes, 3 high (sea)	0.01
8 lanes, 4 high (sea)	0.03

Table 8.10: Best penalties found in experiment 3. Penalties are reshuffle movement penalties in hours. The top half of the table shows results for experiments done with all normal 20 ft. containers, the bottom half is for sea-sea only experiments.

Hypothesis 3.4 Confirmed

Hypothesis 3.5 Confirmed

Hypothesis 3.6 Confirmed

8.5 Experiment 4 - Peak-Adjusted TVR

In experiment 3, we argued that sea-sea containers should be stacked as close to the transfer point quayside as possible, because every move further on is a waste of time. Likewise, we said that sea-land and land-sea containers should be stacked near the transfer point landside, because for them distance does not matter (as they need to traverse the entire lane anyway), and they are out of the way for sea-sea containers there.

There is a slight problem with this reasoning, though. The above is true only if the time spent driving across the lane by ASCs for sea-land containers is valued the same at every point in time. However, since sea containers often arrive and depart many at a time (i.e. in a jumbo or deep sea ship), there are big peaks in crane workload. At these times, it would be not such a great idea to move sea-land containers all the way across the lane, because there is much else to do.

In this experiment, we try to counter this problem and extend the algorithm of experiment 3, by not putting the sea-land containers next to the transfer point landside, but somewhat further away. This is achieved by dividing every lane segment into two parts; one for sea-sea containers (near the transfer point quayside) and one for other containers (near the transfer point landside). We then stack all containers as close to the transfer point quayside, but in their own part of the segment. The size of the two parts is a configurable parameter. In the experiments it was set to 74% for sea-sea containers (which is roughly the fraction of that type in 20 ft. containers). There is also an option to allow stacking of sea-sea containers in the land part.

Parameter	Values used
Allow sea-sea containers in land part?	yes / no
# Lanes	6 / 8
Max. pile height	3 / 4
Reshuffling movement penalty	-0.03, -0.01, 0, 0.003, 0.01, 0.02, 0.03, 0.04, 0.05, 0.1

Table 8.11: Parameters used experimenting with Experiment 4.

8.5.1 Experimental setup

To test this algorithm, which we shall call TVR-PA (for *peak adjusted*) from now on, we used the settings of section 7.1, with the exception of the stack size, just like in Experiments 1, 2 and 3. We do not include reefer containers, or any other containers over 20 ft. in length. Other parameters were varied, as can be seen in table 8.11.

In this experiment we are particularly interested in the value of putting sea-sea containers close to the transfer point quayside. We therefore compare the results to random stacking. We would also like to know what the effects of different penalties are. The difference is measured by looking at the times it takes for a container to leave the stack.

It would not make much sense to test this algorithm with sea-sea containers only, because it is aimed only at improving the combination of both types.

8.5.2 Hypotheses

Hypothesis 4.1 The best TVR-PA stacking algorithm will have a lower exiting time than the best TVR.

Hypothesis 4.2 Allowing sea-sea containers in the land part in the TVR-PA stacking algorithm will lead to more reshuffles for smaller stacks (compared to not mixing).

Hypothesis 4.3 Allowing sea-sea containers in the land part in the TVR-PA stacking algorithm will lead to higher exiting times (compared to not mixing).

In this experiment, we expect roughly the same results as in experiment 3. The question is which algorithm will perform better.

We also suspect a slightly lower exiting time in experiment 4 than in experiment 3, when using a low penalty in both. This is because there should be slightly less pressure on the crane at peak times, while there is no other change (sea-sea containers should not be interfering with other containers). With a high penalty and a small stack, the levelling process is less efficient because of the 2 parts, which probably increases the exiting time and the number of reshuffles, when compared to experiment 3.

Regarding the mixed or unmixed version of the algorithm, we suspect there to be very little difference (if any at all) between both results when using a big stack. This is because there will likely not be a need for any sea-sea containers to be put in the "land" part, especially with low penalties, and even if there was a need, there is plenty of space. In small stacks, on the other

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.92%	81.10%	99.60%	59.44%	0.48	0.33	1.30	0.76
Random stacking	69.44%	107.69%	89.56%	62.70%	0.73	0.47	2.09	1.23
TPRL	62.82%	95.35%	96.41%	57.78%	0.43	0.29	1.13	0.66
TVR (penalty 0.0)	62.06%	95.47%	99.28%	57.02%	0.40	0.27	1.03	0.62
TVR-PA (mixed, 0.0)	63.23%	95.49%	98.66%	57.13%	0.41	0.28	1.07	0.63
TVR-PA (unmixed, 0.0)	63.41%	95.58%	98.70%	57.11%	0.41	0.28	1.07	0.62
TVR (penalty 0.03)	63.63%	86.83%	99.65%	57.60%	0.40	0.28	1.02	0.61
TVR-PA (mixed, 0.03)	63.52%	90.60%	99.50%	57.69%	0.42	0.28	1.08	0.63
TVR-PA (unmixed, 0.03)	64.80%	90.13%	99.48%	57.43%	0.40	0.28	1.03	0.62

Table 8.12: Results of experiment 3 vs 4. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 3. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

hand, there will probably be larger differences, since a lack of space is far more an issue. We cannot predict in advance which version is best, because they both have their advantages and drawbacks. Mixed segments leave more room for the sea-sea containers, but this goes at the expense of land containers.

Thus, allowing sea-sea containers in the land part offers some extra possibilities, which may be needed in a small stack. However, it could also limit the options to stack sea-land containers, which could undo this. Generally speaking, a high reshuffling penalty will level out the stack, and thus also increase the number of sea-sea containers in the land part. Conversely, there should be very little difference in the results of allowing and not allowing the mix, when a low penalty (which encourages high piles) is used.

8.5.3 Results

In table 8.12 is a comparison of some results of the TVR and TVR-PA algorithms on a relatively small 6 lane, 3 high stack. Table 8.13 contains data for a relatively big 8 lane, 4 high stack.

In the case of a small stack, the TVR-PA algorithm seems to perform slightly worse compared to normal TVR. Both exiting times and reshuffles are up, as well as ASC workload. In the table two penalty values are shown, but these results also appear with other penalties.

With a larger stack, however, the results are less clear. With a low penalty, reshuffles are up, but with a high penalty they are down, all compared to the TVR equivalent. Normal TVR's exiting times appear to be slightly lower than those of TVR-PA, but the differences are minute and well inside the 95% confidence intervals. ASC workloads also differ slightly, with a slightly lower value for normal TVR.

In both setups, there are differences between the results for mixed (i.e. allowing sea-sea containers to be stacked in the "land" part) and unmixed TVR-PA. Unmixed exiting times are generally lower than mixed. These differences are small, but (for the small stack) outside the 95% confidence interval limits, so they are significant.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.20%	80.60%	99.35%	63.17%	0.84	0.53	2.52	1.44
Random stacking	69.32%	134.11%	84.68%	69.41%	1.83	1.15	5.48	3.64
TPRL	58.91%	116.67%	90.21%	61.89%	0.77	0.49	2.26	1.33
TVR (penalty 0.0)	47.68%	96.34%	85.22%	45.85%	0.21	0.18	0.40	0.30
TVR-PA (mixed, 0.0)	46.07%	85.99%	86.98%	46.89%	0.21	0.18	0.40	0.29
TVR-PA (unmixed, 0.0)	46.07%	85.99%	86.98%	46.89%	0.21	0.18	0.40	0.29
TVR (penalty 0.03)	45.02%	64.37%	99.52%	47.23%	0.19	0.16	0.34	0.26
TVR-PA (mixed, 0.03)	44.42%	64.36%	98.72%	47.43%	0.20	0.17	0.37	0.28
TVR-PA (unmixed, 0.03)	43.89%	64.91%	98.74%	47.14%	0.20	0.17	0.35	0.27

Table 8.13: Results of experiment 3 vs 4. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 4. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

8.5.4 Discussion

When using TVR-PA, it appears that not mixing the two parts of the stack is best. The sea-sea containers in the land section take up much valuable space and also have to move further to get there. Whether to mix or not to mix only matters when space is tight, with large stacks, the algorithm almost never puts sea-sea containers in the land section.

TVR-PA is, compared to TVR, almost the same in terms of results. As predicted, TVR performs relatively best in a small stack, because the levelling part of the algorithm has to go up another stacking level a bit sooner. For larger stacks, it appears normal TVR still holds a tiny advantage. We can therefore not say that TVR-PA is better than TVR.

8.5.5 Conclusion

TVR-PA does not lead to improvements, compared to TVR. Mixing TVR-PA is not a good idea, because it increases exiting times.

Hypothesis 4.1 Rejected

Hypothesis 4.2 Rejected

Hypothesis 4.3 Confirmed

8.6 Experiment 5 - TVR with departure time classes (TVR-DTC)

In the past two experiments, any knowledge about departure times was ignored. In this experiment, we put it back into the equation, to combine the two ideas of using residence time knowledge and calculating the costs and reshuffle probabilities of possible locations. We use the departure time classes idea from section 8.3.

Departure time classes are pretty easy to use with the existing TVR algorithm. With the classes, we can more accurately estimate the number of expected reshuffles, which leads to a better calculation of costs for each pile, which, in turn, leads to better stacking decisions.

One major difference with the original TVR algorithm that we will have to make, is that it is now no longer necessarily optimal to stack on the closest positions available near the transfer points. This is because piles further away may have favourable departure times and lead to less reshuffles. We thus have to check every position from the transfer point on further down the lane. We can only stop once we have found a pile where stacking would lead to no extra reshuffles, for every possible stacking level. This will make the algorithm much slower.

In addition, equation 8.6 is not valid for this algorithm, because we no longer assume that every pile has the same probability of being chosen. Rather, the probability of an extra reshuffle for every pile are decided using the following algorithm:

1. Set c_{min} as the earliest departure class in the current pile and c_{new} as the departure class of the container to be stacked.
2. if $c_{min} > c_{new}$, there is no reshuffle ($P(nr) = 0$).
3. else if $c_{min} < c_{new}$, there is definitely a new reshuffle ($P(nr) = 1$).
4. If the classes are equal, count the number of times the class occurs in the current pile as n. Then $P(nr) = \frac{n}{n+1}$.

8.6.1 Experimental setup

We use the same setup as used previously, see table 8.11. The classes of departure times are the same as in experiment 2; see table 8.6.

8.6.2 Hypotheses

Hypothesis 5.1 The best TVR-LDT stacking algorithm will have lower exiting times, reshuffles and ASC workloads than the best TVR algorithm.

Hypothesis 5.2 The best TVR-LDT stacking algorithm will have lower exiting times, reshuffles and ASC workloads than the best LDT algorithm.

Since this algorithm combines "the best of both worlds", in this case of the two ideas we use to improve stacking efficiency (LDT and TVR), we expect it to perform better than the two ideas individually, when comparing the "best" penalties for both algorithms. For other penalties, this may not be the case, especially for negative penalties. This is because the residence time knowledge allows us to make better estimates of reshuffle probabilities, which will lead to a very high number of reshuffles, since these penalties favour reshuffles, rather than penalise them.

8.6.3 Results

Some results (for a relatively small stack of 6 lanes, 3 high), are in table 8.14. It is clear that TVR-DTC outperforms normal TVR, but the exiting times, ASC workloads and reshuffle percentages are significantly higher than those of LDT-DTC.

Exp.	Description	ROC %	RDC %	GPU %	ASC %	ETQ hrs.	ETL hrs.	90% ETQ	90% ETL
-	LEV	62.92	81.10	99.60	59.44	0.48	0.33	1.30	0.76
-	RS	69.44	107.69	89.56	62.70	0.73	0.47	2.09	1.23
2	RS-DTC	54.07	77.55	86.25	59.14	0.46	0.32	1.22	0.74
2	LDT-DTC	12.20	18.08	88.67	50.42	0.22	0.18	0.44	0.32
3	TPRL	62.82	95.35	96.41	57.78	0.43	0.29	1.13	0.66
3	TVR (p=-0.03)	80.01	131.68	75.85	57.90	0.66	0.42	1.87	1.09
3	TVR (p=0)	62.24	94.95	99.35	57.01	0.40	0.28	1.02	0.62
3	TVR (p=0.03)	63.71	86.50	99.65	57.57	0.40	0.28	1.00	0.61
3	TVR (p=0.04)	63.52	84.76	99.65	57.61	0.40	0.27	1.00	0.60
5	TVR-DTC (p=0)	32.62	47.44	95.20	52.28	0.28	0.21	0.65	0.39
5	TVR-DTC (p=0.01)	30.19	43.26	97.43	52.37	0.28	0.21	0.67	0.39
5	TVR-DTC (p=0.03)	30.82	43.68	97.91	52.63	0.30	0.22	0.74	0.41

Table 8.14: Results of experiment 5, compared with 2 and 3. Terminal layout: 6x34x6x3. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation. "p" is used to indicate the values for the reshuffle penalty TT_r.

8.6.4 Discussion

Apparently, the combination of TVR and LDT-DTC is not as good as we predicted. The explanation for this lies in the fact that the LDT-DTC algorithm searches for the pile with the smallest difference in departure time. TVR-DTC does not do this, however. This means stacking is less efficient, and hence exiting times are higher.

8.6.5 Conclusion

The current form of LDT-DTC does not work. The strategy of putting containers with close (preferably adjacent classes of) departure times on top of each other is a good one. The LDT-DTC algorithm should be improved to include this idea.

Hypothesis 5.1 Confirmed

Hypothesis 5.2 Rejected

8.7 Experiment 6 - TVR-DTC with minimal class differences

In experiment 5 (section 8.6), we found that TVR-DTC did not work as hoped. We suspect this is due to the fact that this version of the TVR algorithm does not use the strategy of putting containers on top of each other, which depart with a small amount of time after the one above them. In this experiment, we try to extend the TVR-DTC algorithm with this functionality. We call this algorithm TVR-DTC-MD (for Minimal Differences).

We implement the MD functionality by changing equation 8.1 as follows (equation 8.7):

$$\min[2 \cdot TT_{qp} + L_{qp} + (TT_r + L_r) \cdot ER(n) + CP \cdot (C_{diff} - 1)] \tag{8.7}$$

Where TT_{qp} is the travel time from transfer point to pile, L_{qp} are the combined lifting times for the entire operation, TT_r and L_r are the reshuffle travel time and lifting time, respectively. $ER(n)$ is the reshuffle probability (as per the algorithm discussed in section 8.6). CP is the class difference penalty and C_{diff}, the class difference, which we deduct by 1, since a difference of 1 class between the two containers is what we aim for, is defined as follows:

$$C_{diff} = \begin{cases} 6 - C_{new} & \text{if we stack on a new pile} \\ C_{below} - C_{new} & \text{if } C_{new} < C_{below} \\ 2 & \text{otherwise} \end{cases} \tag{8.8}$$

With C_{new} and C_{below} being the departure time class of the containers to be stacked and stacked upon, respectively. If we stack on a new pile, this can be seen as stacking on top of a container of class 6 (which does not exist), because this would be the perfect place for a container of class 5. If the container to be stacked might lead to reshuffles (i.e. it does not depart before C_{below}), a different approach is needed. If we use the same formula, $C_{below} - C_{new}$, we would get negative penalties, which has the undesirable effect of rewarding reshuffles. An alternative could be using the absolute value instead, but this would lead to lower penalties for minimally different classes stacked on top of each other. This does not make much sense, because a reshuffle is always equally bad. We thus opt for a fixed value of 2 as class difference if the actual difference is 0 or lower. Note that this in effect is a further reshuffle penalty. We tested some other values for this parameter, but found little differences.

8.7.1 Experimental setup

We use the same setup as used previously, see table 8.15, but test only a limited number of reshuffling movement penalties, in order to keep down the number of experiments we have to perform, since we also test a number of class difference penalties. The classes of departure times are the same as in experiment 2 and 5; see table 8.6.

8.7.2 Hypotheses

Hypothesis 6.1 The best TVR-DTC-MD stacking algorithm will have lower exiting times, reshuffles and ASC workloads than the best TVR-DTC algorithm.

Parameter	Values used
Containers allowed	both sea-sea and other
# Lanes	6 / 8
Max. pile height	3 / 4
Class difference penalty (CP)	0.003, 0.01, 0.03, 0.04
Reshuffling movement penalty	0.01, 0.02, 0.03, 0.04

Table 8.15: Parameters used experimenting with Experiment 6.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
TVR-DTC								
0.01	30.16%	42.77%	97.95%	52.13%	0.26	0.20	0.61	0.36
LDT-DTC	12.20%	18.08%	88.67%	50.42%	0.22	0.18	0.44	0.32
TVR-DTC-MD								
0.01rm, 0.003 cd	23.00%	31.50%	95.86%	51.15%	0.22	0.18	0.47	0.31
0.01rm, 0.01 cd	22.78%	29.88%	91.60%	50.65%	0.19	0.17	0.38	0.30
0.01rm, 0.03 cd	45.18%	63.55%	84.30%	53.72%	0.33	0.24	0.85	0.49
0.01rm, 0.04 cd	46.71%	66.20%	83.82%	54.18%	0.34	0.25	0.92	0.52

Table 8.16: Results of experiment 6. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 3. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

Hypothesis 6.2 The best TVR-DTC-MD stacking algorithm will have lower exiting times, reshuffles and ASC workloads than the best LDT-DTC algorithm.

The improvement of the minimal differences should give this algorithm the final edge versus all other algorithms tested, at least for the "best" penalty applied. This effect should be particularly clear in the smaller stacks, where a lot of reshuffles (and hence is where most improvements are likely to be made) take place. Larger stacks are more easily managed and already have very low reshuffling percentages and exiting times. Still, this algorithm should be able to lower the exiting times even more, perhaps at the cost of extra reshuffles.

8.7.3 Results

Some results can be found in table 8.16, more results are in appendix C.12.

In these results we can see that exiting times for the best TVR-DTC-MD algorithm are lower than for LDT-DTC. The number of reshuffles is much higher though. Generally, for every TVR-DTC version compared, there is a TVR-DTC-MD version with lower exiting times and less reshuffles.

These results are for a small 6 lane, 3 high, 6 wide, 34 long stack. In larger stacks, the algorithm's performance was relatively weaker and not better than LDT-DTC. Still, the exiting times and reshuffles remain lower than with TVR-DTC.

8.7.4 Discussion

The TVR-DTC-MD seems to be a good improvement over TVR-DTC. This confirms the earlier conclusion that the minimal difference (MD) functionality in the residence time based algorithms is important. However, TVR-DTC-MD is only slightly better than LDT-DTC, and only on relatively small stacks. On larger stacks, the algorithm performs just as good or slightly worse. This indicates that the proposed cost function (equation 8.7) is not accurate enough at estimating the differences in time costs for every pile. It remains to be seen changing which part(s) of the equation might lead to improvements.

8.7.5 Conclusion

The results clearly show that TVR-DTC-MD is not always better than LDT-DTC, so we reject Hypothesis 6.1; Hypothesis 6.2 is not rejected, because results consistently show that TVR-DTC-MD outperforms TVR-DTC. This, in turn, reconfirms the earlier conclusion that it is worthwhile to aim for adjacent containers to have adjacent departure time classes.

Hypothesis 6.1 Confirmed

Hypothesis 6.2 Rejected

Chapter 9

Conclusions and further research

In this chapter we present the conclusions drawn from the research carried out in this thesis, answer the research questions and discuss possible avenues of further research in this area.

9.1 Conclusions

In this section we present the conclusions of the research by answering the research questions posed in section 1.4. In this thesis, the main objective has been "to discover and analyse the effects of various container stacking policies, and to find better ones" (see section 1.3). The main research question derived from this stated objective has been "Which container stacking rules lead to performance improvements at container terminals?" Additionally, several subquestions were formulated, which we need to answer, in order to answer our main research question. To be able to answer the research questions, we performed a number of experiments, which were done using a custom-made computer program. This program, which was created in Java, with the use of the SSJ discrete-event simulation library, uses a file containing container arrivals, based on actual data from ECT.

We now give answers to these subquestions, one by one.

The first subquestion is: "How can the performance of a container stack be measured?" In section 7.4, we discussed which performance measures we used and why. Our main performance indicator is the exiting time, because the terminal service level for trucks, ships and trains directly depends on it. Secondary measures, which are directly related to the terminal operating costs (and indirectly also to the service level). These other measures include the reshuffling percentages, ground position usage, ASC workload and entry times.

The second subquestion is: "Which strategies for container stacking can be used, using only basic information?" We experimented with several stacking strategies, which use only very basic information (i.e. information that would always be known about the container, namely point of entry and point of exit, as well as container size) about the container (see sections 8.1, 8.4 and 8.5). The first experiments were used as benchmark tests for the other algorithms (section 8.1). The tested strategies were random stacking, in which a random allowed position is chosen for every container, and levelling, which minimises pile height by filling the stack layer by layer.

We also tested the TVR algorithm (short for Travel time Versus Reshuffling, see section 8.4), which tries to calculate the costs of putting a container at a certain pile by calculating its distance from the transfer point and the probability of reshuffles. The penalty for moving reshuffles is a variable algorithm parameter. We found that for different setups, different penalties perform best, but the results vary little for values between 0 and 0.05, meaning that the strategy is rather robust w.r.t. this parameter.

A special version of the TVR algorithm to adjust for peaks in ASC workload, TVR-PA (for Peak-Adjusted), was also programmed and experimented with.

The third subquestion is: "Which strategies for container stacking can be formulated, using only basic information, given the performance measured used?" We used residence time knowledge in some or our experiments (see sections 8.2, 8.3, 8.6 and 8.7), expecting to improve container stack performance. Firstly, we used exact residence times to determine which containers could be stacked on top of each other without leading to reshuffles. We choose the pile with the smallest difference in times, so that we keep our options open. If no such pile is found, the container is stacked at the terminal floor, as close as possible to the correct transfer point.

Variations on this algorithm included the use of expected departure times (which are somewhat imperfect and hence may lead to unintentional reshuffles) and dividing the (actual) departure times into a number of classes, and use these classes to determine where reshuffles will be (LDT-DTC). We also tried mixing the LDT-DTC algorithm with the TVR algorithm to improve results even further (see TVR-DTC in section 8.6 and TVR-DTC-MD in section 8.7).

The fourth subquestion is: "Which strategies for container stacking can be formulated, using residence time information, given the performance measured used?" From the two benchmark algorithms tested, we found that levelling always performed better than random stacking. However, the best TVR algorithms always outperformed both random stacking and levelling in our tests. The TVR-PA strategy mostly produces slightly worse results when compared with normal TVR, for the same penalty.

Considering the algorithms using residence times, we found that these strategies did much to improve the performance of the stack. We created the LDT algorithm (Levelling with Departure Times) to make use of residence time knowledge. The algorithm searches for the pile with a container on top which departs as soon as possible after the new container. If no such pile is found, the container is put on the ground, as close to the transfer point as possible. The LDT algorithm performed much better than the benchmark tests and the TVR algorithm. Even with the use of expected departure times, or the use of departure time classes, results were much better than TVR. The LDT-DTC algorithm performed only slightly worse than LDT with actual departure times.

The tests with the combination of LDT-DTC and TVR did not meet with much success, as it became clear that the minimal departure time difference between the 2 containers is very important. The TVR-DTC-MD algorithm was developed to solve this, but its performance was not better than LDT-DTC.

This brings us to the main research question of this thesis: "Which container stacking strategies lead to performance improvements at ECT Maasvlakte-like container terminals?" From the subquestions' answers above, we can conclude that both random stacking and levelling are not

very good strategies. When no special information is available about a container, the standard TVR algorithm performs much better than these two. However, in case some knowledge about residence times is known, the results indicate that using the LDT algorithm is a better idea. A combination of both strategies has so far not yielded better results.

9.2 Further research

We have tested the strategies using a number of variations of the basic layout (in terms of the number of lanes, the lane width and length, and the maximum stacking height). The overall layout of the yard has been the same for all experiments and it would be interesting to evaluate these simple rules for other basic layouts such as lanes that are parallel rather than perpendicular to the quay, or testing layouts with multiple ASCs per lane, such as used in Hamburg and at the new ECT Euromax terminal in Rotterdam.

Another interesting avenue of research might be taking the used knowledge of residence times even further towards reality, for example by using more short-term categories (since the departure times are much easier to estimate on the short term than the long term), or by no longer explicitly guaranteeing that lower classes mean earlier departures. It remains to be seen what the effects of this might be on exiting times and reshuffles.

In terms of improving the strategies, we expect that the LDT-DTC algorithm, which we tried to improve upon using TVR-DTC and TVR-DTC-MD, but failed to, can still be improved using the TVR principle. One way to perhaps improve all strategies, is to make use of the knowledge of ASC workloads, and adjust the algorithms in such a way, that very busy lanes are not selected for new containers. This should balance workload better and improve exiting times.

Appendices

Appendix A

UML class diagrams

In this appendix we give some UML class diagrams of the SSTACk simulation program. These diagrams are not meant to be complete representations of the program, and are for clarification purposes only. For clarity, some variables and most methods have been omitted from the diagram.

In figure A.1, the main components of the simulation are shown, as well as their relations between each other and with other packages, namely for AGVs and straddle carriers (packages SSTACk.agv and SSTACk.sc, figure A.2) and ASC tasks (package SSTACk.asc, figure A.3).

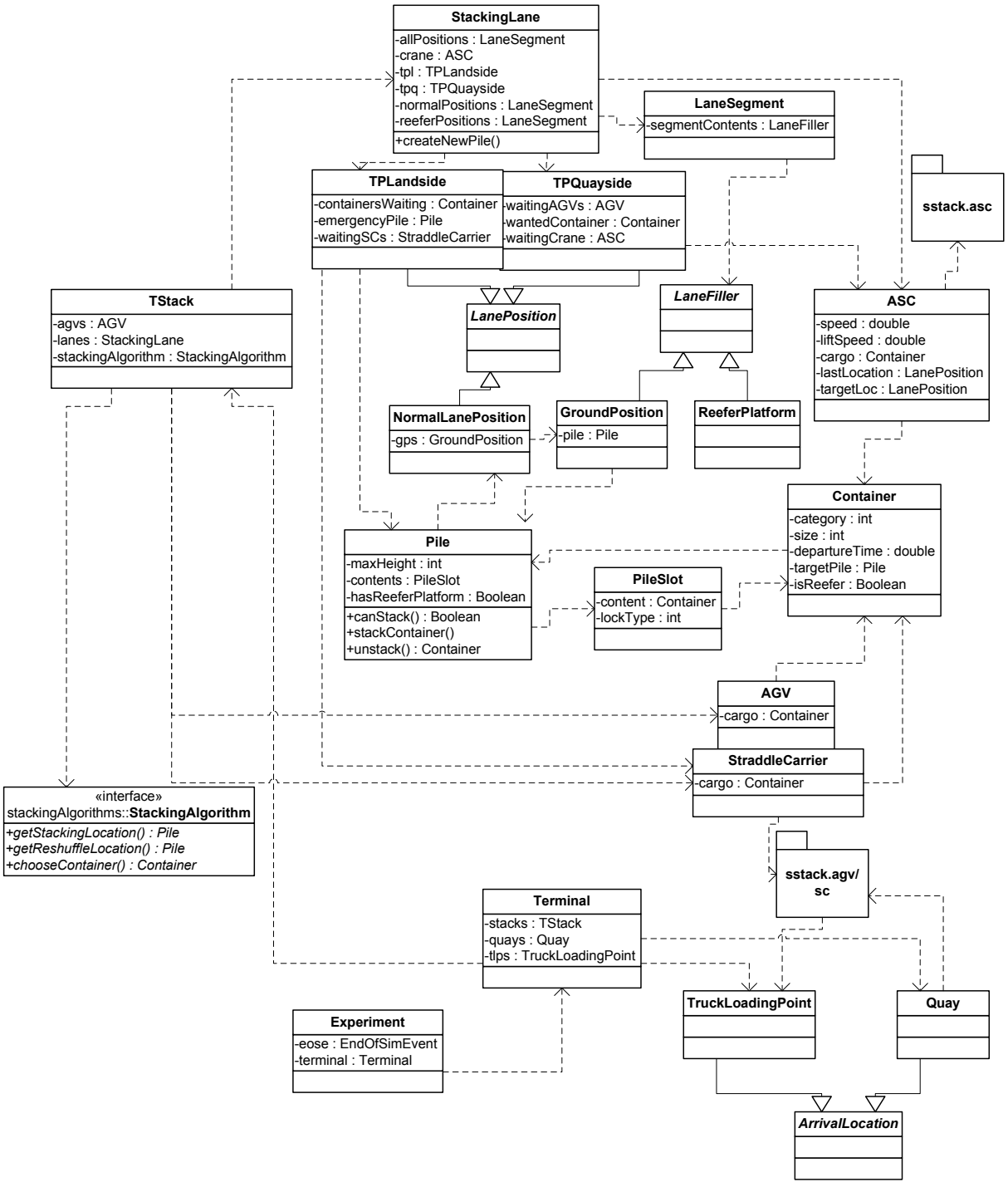

Figure A.1: UML class diagram for SSTACk.main package.

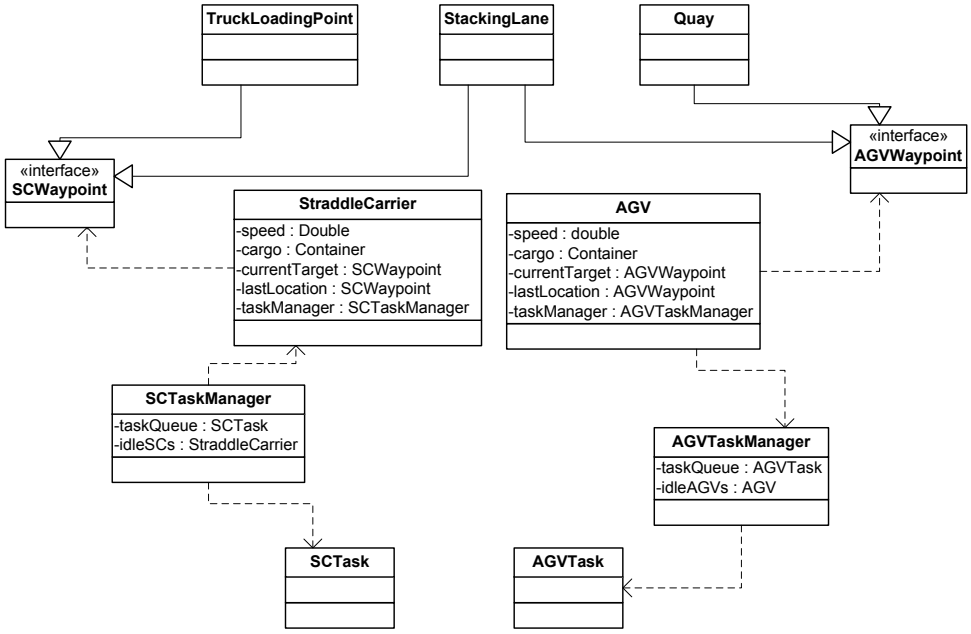

Figure A.2: UML class diagram for SSTACk.agv and SSTACk.sc packages.

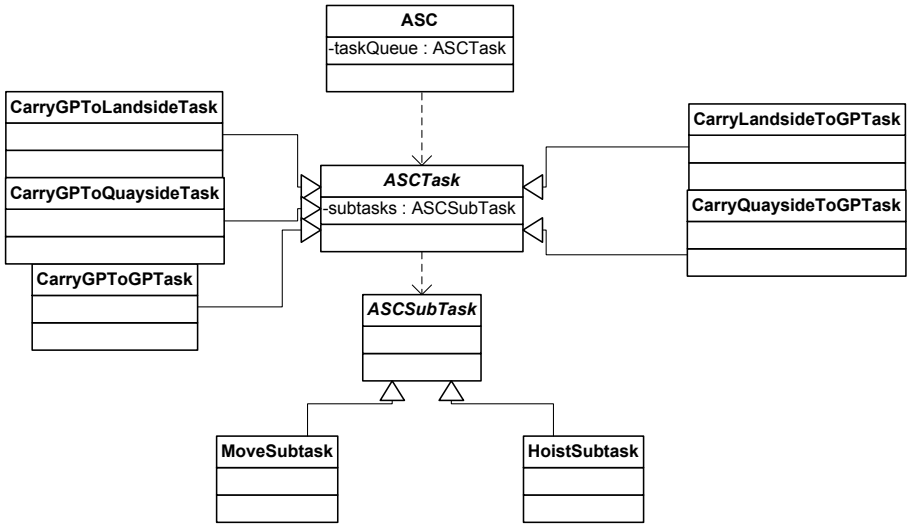

Figure A.3: UML class diagram for SSTACk.asc package.

Appendix B

Generator algorithm

```
Step 1: generate arrival times of jumbo and deep sea ships
Step 2: generate arrival times of other ships and trains
Step 3: generate arrival times of trucks
Step 4: generate arrival and departure times per individual container
Step 5: match departure times with arrival times
```

B.1 Step 1: generate arrival times of jumbo and deep sea ships

```
Input: -schedule: an array containing scheduled ship arrival times
                  (3-week schedule)
       -periodLength (in hours)

Output: -arrivals: an array containing actual ship arrival times
                   (schedule for period length)

        BEGIN
            currentPeriod = 0
            time = 0
            WHILE (time <= periodLength) DO
                FOREACH value in schedule
                    time = value + UNIF(-12,12) + currentPeriod * 504 hours
                                                    (= 7 * 24 * 3 weeks)
                    IF (time > periodLength) THEN
                        BREAK
                    ELSE
                        // add time to arrivals
                    ENDIF
                ENDFOR
                currentPeriod = currentPeriod + 1
            ENDWHILE
```

```
END
```

B.2 Step 2: generate arrival times of other ships and trains

```
Input: -interArrival: a value specifying the interarrival time of
                      this mode (in hours)
       -firstArrival: a value specifying the first arrival time of
                      this mode (in hours)
       -periodLength (in hours)

Output: -arrivals: an array containing actual ship/train arrival times
                   (schedule for period length)

        BEGIN
            scheduleTime = firstArrival
            time = scheduletime + UNIF(-1,1)
            WHILE (time <= periodLength) DO
                IF (time > 0) THEN
                    // add time to arrivals
                ENDIF
                scheduleTime = scheduleTime + interArrival
                time = scheduleTime + UNIF(-1,1)
            ENDWHILE
        END
```

B.3 Step 3: generate arrival times of trucks

This step consists of two parts. First, it is determined how many containers are to be generated in every hour. Second, for every hour a number of times are generated, corresponding to the numbers calculated in the first part.

```
Input: -hourlyPercentages: an array of size 168 (168 hours = 7 * 24 = 1 week),
                           containing in every cell the percentage of
                           arriving trucks for the corresponding hour
       -numberOfTrucks: integer variable for the total number of trucks per week
       -periodLength (in hours)

Output: -hourlyNumbers: an array of size periodLength (rounded up to nearest
                integer), containing in every cell the number of arriving
                trucks for the corresponding hour
```

```
        BEGIN
            week = 0
            time = 0
            FOR hour=0 TO length(hourlyPercentages-1) STEP 1
                perc = hourlyPercentages[hour]
                numTrucks = Round(perc * numberOfTrucks)
                // add numTrucks to hourlyNumbers

                // check whether time is not above periodLength
                time = hour + week * 168
                IF (time > periodLength) THEN
                    BREAK
                ENDIF
            ENDFOR
            week = week + 1


            IF (Sum(hourlyNumbers) > numberOfTrucks) THEN
                // remove some random arrivals
            ELSEIF (Sum(hourlyNumbers) < numberOfTrucks) THEN
                // add some random arrivals
            ENDIF
        END


Input: -hourlyNumbers: an array of size periodLength (rounded up to nearest
                integer), containing in every cell the number of arriving
                trucks for the corresponding hour
       -periodLength (in hours)

Output: -arrivals: an array containing actual truck arrival times
                    (schedule for period length)

        BEGIN
            time = 0
            WHILE (time <= periodLength) DO
                FOR hour=0 TO periodLength-1 STEP 1
                    numTrucks = hourlyNumbers[hour]
                    FOR i=0 TO numTrucks-1 STEP 1
                        time = hour + UNIF(0,1)
                        IF (time > periodLength) THEN
                            BREAK
                        ELSEIF (time > 0) THEN
                            // add time to arrivals
                        ENDIF
```

```
                ENDFOR
            ENDFOR
        ENDWHILE
    END
```

B.4 Step 4: generate arrival and departure times per individual container

This step consists of two parts. In the first part, times for deep sea and jumbo ships are generated and in the second part the other modes are done. Note that trucks only deliver one container, so their times do not need to be calculated.

```
Input: -holdContentsLoad and holdContentsUnload:
                a three-dimensional array of the ship's holds with their contents
       -shipArrival: a value specifying the arrival time of this ship (in hours)
       -craneSchedules: a two-dimensional array, containing the work order for
                        every crane.
       -repositioningTime: a value specifying the time it takes a quay crane to
                        position for the next hold
       -craneTime: time it takes a quay crane to load or unload one container (in hours)

Output: -arrivalsLoad: an array containing actual container departure times
                (schedule for period length)
        -arrivalsUnload: an array containing actual container arrival times
                (schedule for period length)

        BEGIN
            // generate times per crane
            FOREACH schedule IN craneSchedules
                startTime = shipArrival + repositioningTime
                genTime = startTime

                FOREACH hold IN schedule
                    // unload containers
                    FOREACH container IN holdContentsUnload[hold]
                        genTime = genTime + craneTime
                        // add genTime to arrivalsUnload
                    ENDFOR

                    // load containers
                    FOREACH container IN holdContentsLoad[hold]
                        genTime = genTime + craneTime
                        // add genTime to arrivalsLoad
```

```
                ENDFOR

                // reposition for next hold
                genTime = genTime + repositioningTime
            ENDFOR
        ENDFOR
    END


Input: -numUnload: value specifying the number of containers to be unloaded
                from this ship
       -numLoad: value specifying the number of containers to be loaded
                onto this ship
       -shipArrival: a value specifying the arrival time of this ship (in hours)
       -craneTime: time it takes a quay crane to load or unload one container
                (in hours)

Output: -arrivalsLoad: an array containing actual container departure times
                (schedule for period length)
        -arrivalsUnload: an array containing actual container arrival times
                (schedule for period length)

        BEGIN
            // there is only one crane
            genTime = shipArrival

            // unload containers
            FOR i=0 TO numUnload-1 STEP 1
                genTime = genTime + craneTime
                // add genTime to arrivalsUnload
            ENDFOR

            // load containers
            FOR j=0 TO numLoad-1 STEP 1
                genTime = genTime + craneTime
                // add genTime to arrivalsLoad
            ENDFOR
        END
```

B.5 Step 5: match departure times with arrival times

This step is the most complicated one, and consists of four parts. First, the containers both arriving and leaving via jumbo or deep sea ship are done. The second algorithm describes how

the arrivals and departures of containers leaving via jumbo or deep sea ship, but arriving via a different mode, are matched. The third one describes the case of containers arriving, but not leaving via jumbo or deep sea ship. Finally, the fourth part is to determine how all other containers are matched.

Match DS/Jumbo

```
Input: -ODjumboDS: a matrix of the flows between modes (incl individual DS/jumbo ships),
                        per 3-week period
       -fromArrivals: a 2-dimensional array, containing the unloads for every
                        ship of the arrival mode
       -toDepartures: a 2-dimensional array, containing the loads for every
                        ship of the departure mode
       -fromModeODIndices: an array of integers, representing the indices of
                        the ships of the arrival mode in the OD-matrix
       -toModeODIndices: an array of integers, representing the indices of
                        the ships of the departure mode in the OD-matrix

Output: -matchedArrivals: an array containing actual ship/train arrival times,
                            matched with a departure (schedule for period length)

        BEGIN
            currentPeriod = 0
            FOR i=0 TO Length(toDepartures) STEP 1
                departingShip = toDepartures[i]
                ODindexTo = toModeODIndices[i%Length(toModeODIndices)]

                // select priorShips = ships arriving prior to this

                FOR j=0 TO Length(priorShips-1) STEP 1
                    arrivalShip = priorShips[j]
                    ODindexFrom = fromModeODIndices[j]
                    containerFlow = ODjumboDS[ODindexFrom][ODindexTo]

                    FOR k=0 TO containerFlow-1 STEP 1
                        // choose random departure
                        // choose random arrival with matching type & size
                        IF (found)
                            // add the combination to matchedArrivals
                            // remove departure and arrival from their lists
                        ELSE
                            k=k-1 // no match: try another departure
                        ENDIF
                    END FOR
                ENDFOR
```

```
            IF ((i+1)%Length(toModeODIndices)=0 AND i>0) THEN
                currentPeriod = currentPeriod + 1
            ENDIF
        ENDFOR
    END
```

Match jumbo/ds, arriving with other

```
Input: -ODjumboDS: a matrix of the flows between modes (incl individual DS/jumbo ships),
                   per 3-week period
       -fromArrivals: a 1-dimensional array, containing the unloads for all
                   ships/trains/trucks of the arrival mode
       -toDepartures: a 2-dimensional array, containing the loads (which have
                   not been allocated yet) for every ship of the departure
                   mode (i.e. jumbo or deep sea)
       -fromModeODIndex: an integer, representing the index of
                   the arrival mode in the OD-matrix
       -toModeODIndices: an array of integers, representing the indices of
                   the ships of the departure mode in the OD-matrix

Output: -matchedArrivals: an array containing actual ship/train container arrival times,
                          matched with a departure (schedule for period length)

    BEGIN
        FOR i=0 TO Length(toDepartures) STEP 1
            departingShip = toDepartures[i]
            ODindexTo = toModeODIndices[i%Length(toModeODIndices)]

            containerFlow = ODjumboDS[fromModeODIndex][ODindexTo]

            FOR j=0 TO containerFlow-1 STEP 1
                // choose random departure
                wantedArrivalTime = departureTime - TRIA(48, 72, 144 hours)
                // choose closest matching arrival time in the list
                // add the combination to matchedArrivals
                // remove departure and arrival from their lists
            ENDFOR
        ENDFOR
    END
```

Match jumbo/ds, departing with other

```
Input: -ODjumboDS: a matrix of the flows between modes (incl individual DS/jumbo ships),
```

```
                    per 3-week period
       -fromArrivals: a 2-dimensional array, containing the unloads (which have not been
                    allocated yet) for every ship of the departure mode
                    (i.e. jumbo or deep sea)
       -toDepartures: a 1-dimensional array, containing the loads for all
                    ships/trains/trucks of the arrival mode
       -toModeODIndex: an integer, representing the index of
                    the departure mode in the OD-matrix
       -fromModeODIndices: an array of integers, representing the indices of
                    the ships of the arrival mode in the OD-matrix

Output: -matchedArrivals: an array containing actual ship/train container arrival times,
                        matched with a departure (schedule for period length)

        BEGIN
            FOR i=0 TO Length(fromArrivals) STEP 1
                arrivingShip = fromArrivals[i]
                ODindexFrom = fromModeODIndices[i%Length(fromModeODIndices)]

                containerFlow = ODjumboDS[ODindexFrom][toModeODIndex]

                FOR j=0 TO containerFlow-1 STEP 1
                    // choose random arrival
                    wantedDepartureTime = arrivalTime + TRIA(48, 72, 144 hours)
                    // choose closest matching departure time in the list
                    // add the combination to matchedArrivals
                    // remove departure and arrival from their lists
                ENDFOR
            ENDFOR
        END
```

Match other modes (i.e. arriving or leaving via short sea)

```
Input: -ODjumboDS: a matrix of the flows between modes (incl individual DS/jumbo ships),
                    for the entire generated period
       -fromArrivals: a 2-dimensional array, containing the unloads (which have not been
                    allocated yet) for all other departure mode (i.e. not jumbo or
                    deep sea), with a 1-dimensional array for every mode, containing
                    all its arrivals
       -toDepartures: a 2-dimensional array, containing the loads for all
                    ships/trains/trucks of all the arrival modes
       -toModeODIndices: an array of integers, representing the indices of
                    the departure modes in the OD-matrix
       -fromModeODIndices: an array of integers, representing the indices of
```

```
                    the of the arrival modes in the OD-matrix

Output: -matchedArrivals: an array containing actual ship/train container arrival
                  times, matched with a departure (schedule for period length)

        BEGIN

            WHILE (departures left AND arrivals left)
                // choose a random departure mode
                // choose a random arrival mode
                ODindexFrom = fromModeODIndices[chosenArrMode]
                ODindexTO = toModeODIndex[chosenDepMode]

                containerFlow = ODjumboDS[ODindexFrom][ODindexTO]

                IF (containerFlow > 0)
                    // choose random departure
                    wantedArrivalTime = departureTime - TRIA(48, 72, 144 hours)
                    // choose closest matching arrival time in the list
                    IF (no match found OR (departureTime - arrivalTime) > 210) THEN
                        arrivalTime = wantedArrivalTime
                        // add the combination to matchedArrivals
                        // remove departure from the list
                    ELSE
                        // add the combination to matchedArrivals
                        // remove departure and arrival from their lists
                    ENDIF
                    ODjumboDS[ODindexFrom][ODindexTO] =
                        ODjumboDS[ODindexFrom][ODindexTO]-1
                ENDIF
            ENDWHILE
        END
```

Appendix C

Results of the experiments

In this appendix, we present tables containing the results of our experiments. The following abbreviations are used for the columns:

ROC Percentage of reshuffle occasions

RDC Percentage of reshuffled containers

GPU Ground position usage

STU Stack usage

ASC Average ASC workload

ETQ Exiting time quay

ETL Exiting time TLP

NTQ Entry time quay

NTL Entry time TLP

C.1 Experiment 1 - all containers

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.92%	81.10%	99.60%	59.44%	0.48	0.33	1.30	0.76
Random stacking	69.44%	107.69%	89.56%	62.70%	0.73	0.47	2.09	1.23
Modified RS (real)	35.27%	52.10%	82.94%	54.44%	0.37	0.26	0.94	0.56
LDT (unmixed)	10.20%	15.21%	87.39%	49.93%	0.21	0.17	0.42	0.30
LDT (mixed)	9.09%	13.62%	86.64%	49.62%	0.22	0.18	0.44	0.32
Modified RS (exp)	56.10%	82.60%	77.85%	57.98%	0.41	0.31	1.05	0.71
LDT (unmixed, exp)	47.93%	70.20%	79.15%	54.67%	0.29	0.24	0.63	0.46
LDT (mixed, exp)	48.54%	71.62%	77.96%	55.83%	0.36	0.29	0.83	0.60

Table C.1: Results of experiment 1. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 3. "exp" denotes experiments done with expected departure times, rather than actual. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.20%	80.60%	99.35%	63.17%	0.84	0.53	2.52	1.44
Random stacking	69.32%	134.11%	84.68%	69.41%	1.83	1.15	5.48	3.64
Modified RS (real)	23.79%	46.22%	69.57%	56.34%	0.48	0.33	1.31	0.82
LDT (unmixed)	0.97%	1.96%	72.49%	50.18%	0.20	0.17	0.39	0.30
LDT (mixed)	0.79%	1.62%	71.11%	50.19%	0.22	0.19	0.45	0.35
Modified RS (exp)	56.86%	106.59%	61.76%	63.45%	0.77	0.54	2.05	1.46
LDT (unmixed, exp)	51.42%	98.59%	61.93%	58.78%	0.48	0.36	1.18	0.85
LDT (mixed, exp)	51.53%	100.85%	60.34%	61.79%	0.73	0.53	1.91	1.35

Table C.2: Results of experiment 1. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 4. "exp" denotes experiments done with expected departure times, rather than actual. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	52.73%	52.74%	99.57%	46.15%	0.17	0.16	0.28	0.25
Random stacking	65.11%	97.37%	79.12%	49.84%	0.24	0.20	0.46	0.34
Modified RS (real)	17.73%	26.40%	64.80%	41.15%	0.15	0.14	0.25	0.22
LDT (unmixed)	0.52%	0.78%	67.09%	38.26%	0.12	0.12	0.19	0.18
LDT (mixed)	0.50%	0.77%	66.40%	38.05%	0.13	0.12	0.19	0.19
Modified RS (exp)	52.11%	76.26%	59.12%	45.60%	0.18	0.19	0.31	0.29
LDT (unmixed, exp)	46.04%	67.07%	59.90%	42.55%	0.15	0.15	0.24	0.23
LDT (mixed, exp)	47.27%	69.52%	58.76%	43.96%	0.18	0.18	0.29	0.28

Table C.3: Results of experiment 1. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 3. "exp" denotes experiments done with expected departure times, rather than actual. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	52.63%	52.64%	99.54%	48.53%	0.21	0.19	0.38	0.31
Random stacking	65.77%	118.35%	76.14%	54.38%	0.41	0.30	0.99	0.61
Modified RS (real)	14.16%	27.44%	53.92%	43.15%	0.17	0.16	0.31	0.26
LDT (unmixed)	0.04%	0.07%	54.88%	39.68%	0.13	0.12	0.20	0.19
LDT (mixed)	0.03%	0.06%	53.75%	39.64%	0.13	0.13	0.21	0.20
Modified RS (exp)	55.77%	103.08%	46.97%	49.96%	0.25	0.25	0.50	0.43
LDT (unmixed, exp)	52.22%	99.75%	46.86%	46.27%	0.20	0.19	0.34	0.30
LDT (mixed, exp)	53.23%	104.16%	45.35%	49.06%	0.28	0.26	0.53	0.44

Table C.4: Results of experiment 1. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 4. "exp" denotes experiments done with expected departure times, rather than actual. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

C.2 Experiment 1 - sea-sea containers

	ROC	RDC	GPU	ASC	ETQ	90% ETQ
Levelling	55.17%	55.47%	99.43%	44.35%	0.18	0.31
Random stacking	64.68%	96.63%	78.55%	47.57%	0.26	0.53
Modified RS (real)	18.15%	26.86%	64.20%	39.88%	0.15	0.26
LDT (unmixed)	0.53%	0.81%	65.62%	36.60%	0.12	0.19
Modified RS (exp)	50.04%	73.91%	58.74%	43.86%	0.19	0.34
LDT (unmixed, exp)	43.80%	64.69%	58.45%	40.86%	0.17	0.27

Table C.5: Results of experiment 1. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 3. "exp" denotes experiments done with expected departure times, rather than actual. The "90%" values are the average 90% percentile values. All normal 20 ft. sea-sea containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	90% ETQ
Levelling	55.09%	55.40%	99.39%	46.68%	0.23	0.45
Random stacking	65.32%	117.36%	75.68%	51.93%	0.46	1.14
Modified RS (real)	14.72%	28.26%	53.66%	41.78%	0.18	0.33
LDT (unmixed)	0.03%	0.06%	53.23%	37.55%	0.13	0.20
Modified RS (exp)	53.23%	100.14%	46.87%	48.04%	0.28	0.57
LDT (unmixed, exp)	49.51%	97.24%	45.17%	44.13%	0.22	0.39

Table C.6: Results of experiment 1. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 4. "exp" denotes experiments done with expected departure times, rather than actual. The "90%" values are the average 90% percentile values. All normal 20 ft. sea-sea containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	90% ETQ
Levelling	31.78%	31.78%	97.60%	33.76%	0.13	0.19
Random stacking	58.73%	83.74%	68.41%	37.47%	0.17	0.26
Modified RS (real)	10.58%	15.68%	49.08%	30.86%	0.12	0.19
LDT (unmixed)	0.01%	0.02%	49.36%	28.31%	0.11	0.16
Modified RS (exp)	48.13%	70.65%	44.53%	34.54%	0.15	0.23
LDT (unmixed, exp)	43.76%	64.42%	44.10%	31.45%	0.13	0.19

Table C.7: Results of experiment 1. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 3. "exp" denotes experiments done with expected departure times, rather than actual. The "90%" values are the average 90% percentile values. All normal 20 ft. sea-sea containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	90% ETQ
Levelling	31.66%	31.66%	97.58%	35.41%	0.14	0.21
Random stacking	59.11%	97.31%	66.72%	40.38%	0.21	0.36
Modified RS (real)	9.41%	18.02%	41.05%	32.45%	0.13	0.21
LDT (unmixed)	0.00%	0.00%	40.19%	29.31%	0.11	0.16
Modified RS (exp)	52.03%	96.52%	35.77%	37.85%	0.18	0.29
LDT (unmixed, exp)	49.69%	97.32%	34.22%	34.23%	0.15	0.23

Table C.8: Results of experiment 1. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 4. "exp" denotes experiments done with expected departure times, rather than actual. The "90%" values are the average 90% percentile values. All normal 20 ft. sea-sea containers were included in the simulation.

C.3 Experiment 2 - all containers

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.92%	81.10%	99.60%	59.44%	0.48	0.33	1.30	0.76
Random stacking	69.44%	107.69%	89.56%	62.70%	0.73	0.47	2.09	1.23
Modified RS (real)	35.27%	52.10%	82.94%	54.44%	0.37	0.26	0.94	0.56
Modified RS (class)	54.07%	77.55%	86.25%	59.14%	0.46	0.32	1.22	0.74
Modified RS (exp)	56.10%	82.60%	77.85%	57.98%	0.41	0.31	1.05	0.71
LDT (unmixed)	10.20%	15.21%	87.39%	49.93%	0.21	0.17	0.42	0.30
LDT (unmixed, exp)	47.93%	70.20%	79.15%	54.67%	0.29	0.24	0.63	0.46
LDT (unmixed, class)	12.20%	18.08%	88.67%	50.42%	0.22	0.18	0.44	0.32

Table C.9: Results of experiment 2. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 3. "exp" denotes experiments done with expected departure times, rather than actual. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.20%	80.60%	99.35%	63.17%	0.84	0.53	2.52	1.44
Random stacking	69.32%	134.11%	84.68%	69.41%	1.83	1.15	5.48	3.64
Modified RS (real)	23.79%	46.22%	69.57%	56.34%	0.48	0.33	1.31	0.82
Modified RS (class)	51.12%	92.95%	77.98%	63.80%	0.97	0.60	2.87	1.74
Modified RS (exp)	56.86%	106.59%	61.76%	63.45%	0.77	0.54	2.05	1.46
LDT (unmixed)	0.97%	1.96%	72.49%	50.18%	0.20	0.17	0.39	0.30
LDT (unmixed, exp)	51.42%	98.59%	61.93%	58.78%	0.48	0.36	1.18	0.85
LDT (unmixed, class)	1.51%	3.05%	75.39%	50.64%	0.20	0.17	0.40	0.31

Table C.10: Results of experiment 2. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 4. "exp" denotes experiments done with expected departure times, rather than actual. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	52.73%	52.74%	99.57%	46.15%	0.17	0.16	0.28	0.25
Random stacking	65.11%	97.37%	79.12%	49.84%	0.24	0.20	0.46	0.34
Modified RS (real)	17.73%	26.40%	64.80%	41.15%	0.15	0.14	0.25	0.22
Modified RS (class)	45.94%	64.78%	73.30%	46.94%	0.18	0.17	0.31	0.27
Modified RS (exp)	52.11%	76.26%	59.12%	45.60%	0.18	0.19	0.31	0.29
LDT (unmixed)	0.52%	0.78%	67.09%	38.26%	0.12	0.12	0.19	0.18
LDT (unmixed, exp)	46.04%	67.07%	59.90%	42.55%	0.15	0.15	0.24	0.23
LDT (unmixed, class)	0.63%	0.94%	67.83%	38.43%	0.12	0.12	0.19	0.18

Table C.11: Results of experiment 2. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 3. "exp" denotes experiments done with expected departure times, rather than actual. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	52.63%	52.64%	99.54%	48.53%	0.21	0.19	0.38	0.31
Random stacking	65.77%	118.35%	76.14%	54.38%	0.41	0.30	0.99	0.61
Modified RS (real)	14.16%	27.44%	53.92%	43.15%	0.17	0.16	0.31	0.26
Modified RS (class)	45.30%	79.96%	67.71%	50.40%	0.26	0.21	0.51	0.38
Modified RS (exp)	55.77%	103.08%	46.97%	49.96%	0.25	0.25	0.50	0.43
LDT (unmixed)	0.04%	0.07%	54.88%	39.68%	0.13	0.12	0.20	0.19
LDT (unmixed, exp)	52.22%	99.75%	46.86%	46.27%	0.20	0.19	0.34	0.30
LDT (unmixed, class)	0.06%	0.12%	56.70%	39.96%	0.13	0.12	0.20	0.19

Table C.12: Results of experiment 2. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 4. "exp" denotes experiments done with expected departure times, rather than actual. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

C.4 Experiment 2 - sea-sea containers

	ROC	RDC	GPU	ASC	ETQ	90% ETQ
Levelling	55.17%	55.47%	99.43%	44.35%	0.18	0.31
Random stacking	64.68%	96.63%	78.55%	47.57%	0.26	0.53
Modified RS (real)	18.15%	26.86%	64.20%	39.88%	0.15	0.26
LDT (unmixed)	0.53%	0.81%	65.62%	36.60%	0.12	0.19
Modified RS (exp)	50.04%	73.91%	58.74%	43.86%	0.19	0.34
LDT (unmixed, exp)	43.80%	64.69%	58.45%	40.86%	0.17	0.27
Modified RS (class)	45.53%	64.20%	72.65%	44.81%	0.19	0.34
LDT (class)	0.64%	0.95%	66.47%	36.82%	0.13	0.19

Table C.13: Results of experiment 2. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 3. "exp" denotes experiments done with expected departure times, rather than actual. The "90%" values are the average 90% percentile values. All normal 20 ft. sea-sea containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	90% ETQ
Levelling	55.09%	55.40%	99.39%	46.68%	0.23	0.45
Random stacking	65.32%	117.36%	75.68%	51.93%	0.46	1.14
Modified RS (real)	14.72%	28.26%	53.66%	41.78%	0.18	0.33
LDT (unmixed)	0.03%	0.06%	53.23%	37.55%	0.13	0.20
Modified RS (exp)	53.23%	100.14%	46.87%	48.04%	0.28	0.57
LDT (unmixed, exp)	49.51%	97.24%	45.17%	44.13%	0.22	0.39
Modified RS (class)	44.93%	79.23%	67.11%	48.08%	0.28	0.60
LDT (class)	0.04%	0.08%	55.13%	37.96%	0.13	0.20

Table C.14: Results of experiment 2. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 4. "exp" denotes experiments done with expected departure times, rather than actual. The "90%" values are the average 90% percentile values. All normal 20 ft. sea-sea containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	90% ETQ
Levelling	31.78%	31.78%	97.60%	33.76%	0.13	0.19
Random stacking	58.73%	83.74%	68.41%	37.47%	0.17	0.26
Modified RS (real)	10.58%	15.68%	49.08%	30.86%	0.12	0.19
LDT (unmixed)	0.01%	0.02%	49.36%	28.31%	0.11	0.16
Modified RS (exp)	48.13%	70.65%	44.53%	34.54%	0.15	0.23
LDT (unmixed, exp)	43.76%	64.42%	44.10%	31.45%	0.13	0.19
Modified RS (class)	40.03%	55.42%	61.48%	35.55%	0.14	0.22
LDT (class)	0.01%	0.02%	49.91%	28.44%	0.11	0.16

Table C.15: Results of experiment 2. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 3. "exp" denotes experiments done with expected departure times, rather than actual. The "90%" values are the average 90% percentile values. All normal 20 ft. sea-sea containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	90% ETQ
Levelling	31.66%	31.66%	97.58%	35.41%	0.14	0.21
Random stacking	59.11%	97.31%	66.72%	40.38%	0.21	0.36
Modified RS (real)	9.41%	18.02%	41.05%	32.45%	0.13	0.21
LDT (unmixed)	0.00%	0.00%	40.19%	29.31%	0.11	0.16
Modified RS (exp)	52.03%	96.52%	35.77%	37.85%	0.18	0.29
LDT (unmixed, exp)	49.69%	97.32%	34.22%	34.23%	0.15	0.23
Modified RS (class)	39.81%	67.19%	57.90%	37.97%	0.17	0.28
LDT (class)	0.00%	0.00%	41.41%	29.59%	0.11	0.16

Table C.16: Results of experiment 2. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 4. "exp" denotes experiments done with expected departure times, rather than actual. The "90%" values are the average 90% percentile values. All normal 20 ft. sea-sea containers were included in the simulation.

C.5 Experiment 3 - all containers

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	60.41%	97.60%	96.42%	80.05%	4.68	3.31	12.24	10.52
Random stacking	65.76%	152.10%	87.80%	87.58%	7.20	5.58	16.37	14.85
TVR (penalty 0.003)	56.11%	132.56%	97.48%	75.78%	3.75	2.51	10.79	8.73
TPRL	59.01%	140.22%	90.33%	78.91%	4.29	2.95	11.63	9.76
-0.03	72.07%	205.12%	61.61%	84.71%	6.80	5.11	16.16	14.38
-0.01	56.62%	148.43%	89.54%	76.60%	4.35	2.97	12.15	10.07
0	55.56%	134.25%	97.48%	75.90%	3.79	2.54	10.87	8.86
0.003	56.11%	132.56%	97.48%	75.78%	3.75	2.51	10.79	8.73
0.01	57.06%	127.61%	97.44%	75.83%	3.67	2.46	10.56	8.56
0.02	57.72%	118.87%	97.44%	76.13%	3.50	2.34	10.19	8.30
0.03	58.71%	112.27%	97.47%	75.95%	3.33	2.23	9.68	7.79
0.04	59.30%	108.26%	97.52%	75.87%	3.21	2.15	9.35	7.52
0.05	59.53%	106.40%	97.49%	76.05%	3.24	2.17	9.49	7.55
0.1	60.62%	101.54%	97.34%	77.02%	3.44	2.33	9.83	7.97

Table C.17: Results of experiment 3. A terminal of 5 lanes of 34x6 positions each was used, with a maximum stacking height of 5. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.92%	81.10%	99.60%	59.44%	0.48	0.33	1.30	0.76
Random stacking	69.44%	107.69%	89.56%	62.70%	0.73	0.47	2.09	1.23
TVR (penalty 0.0)	62.06%	95.47%	99.28%	57.02%	0.40	0.27	1.03	0.62
TPRL	62.82%	95.35%	96.41%	57.78%	0.43	0.29	1.13	0.66
-0.03	79.49%	130.91%	75.84%	57.81%	0.65	0.43	1.85	1.09
-0.01	65.59%	105.06%	93.69%	55.89%	0.44	0.30	1.17	0.70
0	62.06%	95.47%	99.28%	57.02%	0.40	0.27	1.03	0.62
0.003	62.38%	94.27%	99.44%	57.15%	0.40	0.27	1.02	0.61
0.01	62.95%	92.26%	99.57%	57.27%	0.40	0.27	1.00	0.61
0.02	63.39%	89.53%	99.63%	57.45%	0.40	0.27	1.01	0.61
0.03	63.63%	86.83%	99.65%	57.60%	0.40	0.28	1.02	0.61
0.04	63.41%	84.85%	99.65%	57.66%	0.40	0.27	0.99	0.61
0.05	63.41%	83.69%	99.65%	57.73%	0.40	0.28	1.03	0.61
0.1	63.27%	81.96%	99.64%	58.20%	0.43	0.29	1.12	0.65
1	63.18%	81.66%	99.62%	58.48%	0.45	0.30	1.18	0.68
100	63.18%	81.66%	99.62%	58.48%	0.45	0.30	1.18	0.68

Table C.18: Results of experiment 3. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 3. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.20%	80.60%	99.35%	63.17%	0.84	0.53	2.52	1.44
Random stacking	69.32%	134.11%	84.68%	69.41%	1.83	1.15	5.48	3.64
TVR (penalty 0.003)	55.57%	109.92%	98.98%	59.50%	0.69	0.42	1.98	1.11
TPRL	58.91%	116.67%	90.21%	61.89%	0.77	0.49	2.26	1.33
-0.03	82.23%	188.11%	58.35%	65.61%	1.69	1.06	5.12	3.34
-0.01	66.01%	143.67%	77.87%	60.51%	0.97	0.59	2.96	1.65
0	55.31%	112.72%	97.99%	59.50%	0.71	0.44	2.08	1.15
0.003	55.57%	109.92%	98.98%	59.50%	0.69	0.42	1.98	1.11
0.01	55.91%	105.77%	99.34%	60.41%	0.66	0.41	1.90	1.07
0.02	57.22%	100.04%	99.44%	60.64%	0.66	0.41	1.87	1.06
0.03	58.21%	94.02%	99.47%	60.74%	0.64	0.40	1.80	1.04
0.04	59.44%	89.86%	99.48%	60.84%	0.63	0.40	1.76	1.02
0.05	60.47%	86.60%	99.49%	60.91%	0.63	0.40	1.78	1.03
0.1	61.84%	82.15%	99.45%	61.59%	0.69	0.44	2.00	1.13
1	62.71%	81.35%	99.40%	62.13%	0.76	0.48	2.23	1.28
100	62.71%	81.35%	99.40%	62.13%	0.76	0.48	2.23	1.28

Table C.19: Results of experiment 3. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 4. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	52.73%	52.74%	99.57%	46.15%	0.17	0.16	0.28	0.25
Random stacking	65.11%	97.37%	79.12%	49.84%	0.24	0.20	0.46	0.34
TVR (penalty 0.003)	44.99%	67.09%	97.47%	44.01%	0.16	0.14	0.25	0.22
TPRL	54.74%	83.28%	81.11%	44.99%	0.17	0.15	0.27	0.24
-0.03	81.88%	135.30%	56.82%	44.38%	0.23	0.19	0.42	0.32
-0.01	63.25%	100.90%	72.96%	43.39%	0.18	0.16	0.30	0.25
0	45.91%	70.53%	94.42%	43.98%	0.16	0.14	0.26	0.22
0.003	44.99%	67.09%	97.47%	44.01%	0.16	0.14	0.25	0.22
0.01	45.50%	65.10%	99.06%	44.28%	0.16	0.14	0.25	0.22
0.02	46.70%	63.02%	99.44%	44.63%	0.16	0.14	0.25	0.22
0.03	47.86%	60.89%	99.55%	44.86%	0.16	0.15	0.26	0.22
0.04	49.02%	59.11%	99.58%	45.00%	0.16	0.15	0.26	0.22
0.05	49.59%	57.96%	99.58%	45.04%	0.16	0.15	0.26	0.22
0.1	52.37%	54.35%	99.59%	45.32%	0.16	0.15	0.26	0.23

Table C.20: Results of experiment 3. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 3. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	52.63%	52.64%	99.54%	48.53%	0.21	0.19	0.38	0.31
Random stacking	65.77%	118.35%	76.14%	54.38%	0.41	0.30	0.99	0.61
TVR (penalty 0.01)	41.63%	75.13%	98.79%	46.54%	0.19	0.16	0.34	0.26
TPRL	58.52%	115.71%	68.55%	48.81%	0.24	0.20	0.47	0.35
-0.03	87.11%	200.99%	43.42%	50.27%	0.44	0.32	1.08	0.66
-0.01	65.72%	142.56%	60.85%	46.52%	0.27	0.22	0.58	0.40
0	47.68%	96.34%	85.22%	45.85%	0.21	0.18	0.40	0.30
0.003	43.50%	84.60%	93.37%	45.72%	0.20	0.16	0.37	0.28
0.01	41.63%	75.13%	98.79%	46.54%	0.19	0.16	0.34	0.26
0.02	43.03%	68.79%	99.38%	46.91%	0.19	0.16	0.34	0.27
0.03	45.02%	64.37%	99.52%	47.23%	0.19	0.16	0.34	0.26
0.04	46.77%	61.34%	99.55%	47.31%	0.19	0.17	0.34	0.27
0.05	48.24%	59.15%	99.56%	47.41%	0.19	0.17	0.34	0.27
0.1	52.32%	54.29%	99.57%	47.71%	0.20	0.17	0.34	0.27

Table C.21: Results of experiment 3. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 4. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

C.6 Experiment 3 - sea-sea containers

	ROC	RDC	GPU	ASC	ETQ	90% ETQ
Levelling	61.87%	72.88%	99.26%	58.34%	0.86	2.36
Random stacking	66.45%	138.86%	79.74%	65.91%	2.20	5.96
TVR (penalty 0.003)	53.62%	133.24%	81.47%	56.30%	0.90	2.56
TPRL	60.52%	144.47%	72.26%	58.69%	0.98	2.72
-0.03	81.46%	235.59%	42.28%	63.47%	2.14	5.76
-0.01	65.38%	175.12%	61.78%	57.62%	1.23	3.52
0	56.04%	141.52%	76.28%	56.53%	0.96	2.74
0.003	53.62%	133.24%	81.47%	56.30%	0.90	2.56
0.01	48.29%	113.96%	93.08%	55.95%	0.81	2.26
0.02	47.07%	101.46%	99.11%	56.41%	0.71	1.91
0.03	48.58%	96.68%	99.33%	56.42%	0.68	1.85
0.04	50.09%	92.31%	99.34%	56.52%	0.68	1.83
0.05	51.60%	87.82%	99.34%	56.68%	0.68	1.84
0.1	59.94%	75.38%	99.30%	57.85%	0.80	2.21

Table C.22: Results of experiment 3. A terminal of 5 lanes of 34x6 positions each was used, with a maximum stacking height of 5. The "90%" values are the average 90% percentile values. All normal 20 ft. sea-sea containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	90% ETQ
Levelling	55.17%	55.47%	99.43%	44.35%	0.18	0.31
Random stacking	64.68%	96.63%	78.55%	47.57%	0.26	0.53
TVR (penalty 0.0)	56.04%	87.93%	80.23%	42.76%	0.18	0.31
TPRL	55.73%	84.75%	81.03%	43.35%	0.18	0.31
-0.03	80.23%	131.98%	56.00%	42.35%	0.24	0.47
-0.01	67.66%	108.96%	68.20%	41.87%	0.20	0.35
0	56.04%	87.93%	80.23%	42.76%	0.18	0.31
0.003	53.65%	82.48%	83.68%	42.74%	0.18	0.31
0.01	47.66%	71.86%	92.33%	42.85%	0.17	0.29
0.02	45.71%	65.93%	98.06%	43.16%	0.17	0.29
0.03	46.55%	64.49%	99.27%	43.42%	0.17	0.29
0.04	47.49%	63.57%	99.41%	43.52%	0.17	0.29
0.05	48.27%	62.55%	99.43%	43.58%	0.17	0.29
0.1	51.86%	58.18%	99.42%	43.95%	0.18	0.30
1	55.17%	55.47%	99.43%	44.35%	0.18	0.31

Table C.23: Results of experiment 3. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 3. The "90%" values are the average 90% percentile values. All normal 20 ft. sea-sea containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	90% ETQ
Levelling	55.09%	55.40%	99.39%	46.68%	0.23	0.45
Random stacking	65.32%	117.36%	75.68%	51.93%	0.46	1.14
TVR (penalty 0.003)	55.71%	112.70%	73.22%	44.44%	0.25	0.51
TPRL	59.29%	117.48%	68.60%	46.54%	0.26	0.53
-0.03	85.59%	196.35%	42.44%	47.63%	0.47	1.14
-0.01	70.03%	153.38%	55.91%	44.72%	0.31	0.70
0	58.19%	120.63%	69.20%	44.52%	0.26	0.55
0.003	55.71%	112.70%	73.22%	44.44%	0.25	0.51
0.01	48.33%	94.50%	84.05%	45.20%	0.23	0.46
0.02	41.70%	74.73%	96.86%	45.32%	0.22	0.41
0.03	42.17%	70.22%	99.23%	45.61%	0.22	0.42
0.04	43.72%	67.61%	99.37%	45.68%	0.22	0.41
0.05	45.27%	65.26%	99.38%	45.78%	0.22	0.41
0.1	51.81%	58.23%	99.40%	46.27%	0.22	0.42
1	55.09%	55.40%	99.39%	46.68%	0.23	0.45

Table C.24: Results of experiment 3. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 4. The "90%" values are the average 90% percentile values. All normal 20 ft. sea-sea containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	90% ETQ
Levelling	31.78%	31.78%	97.60%	33.76%	0.13	0.19
Random stacking	58.73%	83.74%	68.41%	37.47%	0.17	0.26
TVR (penalty 0.003)	48.76%	73.77%	67.72%	33.07%	0.13	0.20
TPRL	55.28%	84.03%	60.62%	33.65%	0.14	0.20
-0.03	81.72%	134.97%	42.01%	32.04%	0.16	0.24
-0.01	65.88%	105.51%	52.54%	31.95%	0.14	0.21
0	51.62%	80.00%	64.23%	33.03%	0.13	0.20
0.003	48.76%	73.77%	67.72%	33.07%	0.13	0.20
0.01	41.08%	60.30%	76.94%	33.18%	0.13	0.19
0.02	32.64%	44.07%	88.60%	33.33%	0.13	0.19
0.03	28.22%	35.19%	96.23%	33.52%	0.13	0.19
0.04	28.49%	34.00%	97.47%	33.68%	0.13	0.19
0.05	29.39%	33.32%	97.66%	33.69%	0.13	0.19
0.1	31.75%	31.75%	97.60%	33.75%	0.13	0.19

Table C.25: Results of experiment 3. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 3. The "90%" values are the average 90% percentile values. All normal 20 ft. sea-sea containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	90% ETQ
Levelling	31.66%	31.66%	97.58%	35.41%	0.14	0.21
Random stacking	59.11%	97.31%	66.72%	40.38%	0.21	0.36
TVR (penalty 0.01)	41.48%	78.21%	71.72%	35.01%	0.15	0.23
TPRL	58.79%	116.26%	51.24%	36.29%	0.16	0.26
-0.03	87.57%	202.01%	31.86%	36.06%	0.23	0.41
-0.01	68.84%	150.00%	43.23%	34.00%	0.18	0.30
0	54.16%	110.51%	56.34%	34.15%	0.16	0.25
0.003	51.17%	101.41%	60.36%	34.22%	0.16	0.25
0.01	41.48%	78.21%	71.72%	35.01%	0.15	0.23
0.02	31.65%	52.20%	85.82%	35.06%	0.14	0.22
0.03	26.49%	36.28%	96.23%	35.18%	0.14	0.21
0.04	27.57%	34.40%	97.46%	35.32%	0.14	0.21
0.05	29.10%	33.27%	97.65%	35.35%	0.14	0.21
0.1	31.77%	31.78%	97.58%	35.41%	0.14	0.21

Table C.26: Results of experiment 3. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 4. The "90%" values are the average 90% percentile values. All normal 20 ft. sea-sea containers were included in the simulation.

C.7 Experiment 4 - mixed piles

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.92%	81.10%	99.60%	59.44%	0.48	0.33	1.30	0.76
Random stacking	69.44%	107.69%	89.56%	62.70%	0.73	0.47	2.09	1.23
TVR (penalty 0.0)	63.23%	95.49%	98.66%	57.13%	0.41	0.28	1.07	0.63
TPRL	62.82%	95.35%	96.41%	57.78%	0.43	0.29	1.13	0.66
-0.03	77.34%	126.12%	75.94%	56.97%	0.57	0.38	1.57	0.94
-0.01	64.55%	101.99%	94.80%	56.94%	0.42	0.29	1.11	0.66
0	63.23%	95.49%	98.66%	57.13%	0.41	0.28	1.07	0.63
0.003	63.33%	94.74%	99.00%	57.23%	0.42	0.28	1.07	0.62
0.01	63.34%	93.01%	99.32%	57.40%	0.42	0.28	1.09	0.63
0.03	63.52%	90.60%	99.50%	57.69%	0.42	0.28	1.08	0.63

Table C.27: Results of experiment 4. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 3. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation. Sea-sea containers were allowed to mix with others.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.20%	80.60%	99.35%	63.17%	0.84	0.53	2.52	1.44
Random stacking	69.32%	134.11%	84.68%	69.41%	1.83	1.15	5.48	3.64
TVR (penalty 0.003)	56.07%	104.02%	98.37%	60.56%	0.68	0.43	1.97	1.11
TPRL	58.91%	116.67%	90.21%	61.89%	0.77	0.49	2.26	1.33
-0.03	80.36%	180.22%	58.33%	63.93%	1.37	0.86	4.14	2.57
-0.01	62.45%	125.48%	88.05%	60.38%	0.81	0.50	2.41	1.34
0	56.09%	106.32%	97.39%	60.54%	0.70	0.43	2.02	1.14
0.003	56.07%	104.02%	98.37%	60.56%	0.68	0.43	1.97	1.11
0.01	56.26%	100.71%	99.05%	60.72%	0.68	0.43	1.97	1.12
0.03	57.64%	95.29%	99.31%	61.09%	0.69	0.43	1.98	1.12

Table C.28: Results of experiment 4. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 4. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation. Sea-sea containers were allowed to mix with others.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	52.73%	52.74%	99.57%	46.15%	0.17	0.16	0.28	0.25
Random stacking	65.11%	97.37%	79.12%	49.84%	0.24	0.20	0.46	0.34
TVR (penalty 0.01)	46.32%	65.20%	95.86%	44.49%	0.17	0.15	0.27	0.23
TPRL	54.74%	83.28%	81.11%	44.99%	0.17	0.15	0.27	0.24
-0.03	79.59%	130.14%	56.90%	43.93%	0.21	0.19	0.39	0.30
-0.01	57.93%	91.88%	79.49%	44.34%	0.17	0.15	0.28	0.24
0	50.14%	74.29%	89.57%	44.28%	0.17	0.15	0.27	0.23
0.003	48.10%	70.52%	92.15%	44.33%	0.17	0.15	0.27	0.23
0.01	46.32%	65.20%	95.86%	44.49%	0.17	0.15	0.27	0.23
0.03	47.47%	61.33%	98.64%	44.90%	0.16	0.15	0.27	0.22

Table C.29: Results of experiment 4. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 3. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation. Sea-sea containers were allowed to mix with others.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	52.63%	52.64%	99.54%	48.53%	0.21	0.19	0.38	0.31
Random stacking	65.77%	118.35%	76.14%	54.38%	0.41	0.30	0.99	0.61
TVR (penalty 0.01)	41.44%	70.70%	96.24%	47.05%	0.21	0.17	0.38	0.29
TPRL	58.52%	115.71%	68.55%	48.81%	0.24	0.20	0.47	0.35
-0.03	84.65%	191.62%	43.54%	49.29%	0.38	0.29	0.90	0.58
-0.01	57.24%	114.25%	74.53%	46.48%	0.23	0.19	0.46	0.32
0	46.07%	85.99%	86.98%	46.89%	0.21	0.18	0.40	0.29
0.003	44.04%	79.70%	90.83%	46.89%	0.21	0.18	0.39	0.29
0.01	41.44%	70.70%	96.24%	47.05%	0.21	0.17	0.38	0.29
0.03	44.42%	64.36%	98.72%	47.43%	0.20	0.17	0.37	0.28

Table C.30: Results of experiment 4. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 4. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation. Sea-sea containers were allowed to mix with others.

C.8 Experiment 4 - unmixed piles

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.92%	81.10%	99.60%	59.44%	0.48	0.33	1.30	0.76
Random stacking	69.44%	107.69%	89.56%	62.70%	0.73	0.47	2.09	1.23
TVR (penalty 0.03)	64.80%	90.13%	99.48%	57.43%	0.40	0.28	1.03	0.62
TPRL	62.82%	95.35%	96.41%	57.78%	0.43	0.29	1.13	0.66
-0.03	77.73%	126.89%	76.00%	56.86%	0.56	0.38	1.54	0.92
-0.01	64.63%	102.09%	94.80%	56.97%	0.42	0.28	1.10	0.65
0	63.41%	95.58%	98.70%	57.11%	0.41	0.28	1.07	0.62
0.003	63.81%	95.00%	99.04%	57.22%	0.41	0.28	1.07	0.64
0.01	64.04%	92.94%	99.33%	57.28%	0.41	0.28	1.05	0.62
0.03	64.80%	90.13%	99.48%	57.43%	0.40	0.28	1.03	0.62

Table C.31: Results of experiment 4. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 3. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation. Sea-sea containers were not mixed with others.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.20%	80.60%	99.35%	63.17%	0.84	0.53	2.52	1.44
Random stacking	69.32%	134.11%	84.68%	69.41%	1.83	1.15	5.48	3.64
TVR (penalty 0.03)	58.39%	95.22%	99.29%	60.75%	0.66	0.41	1.89	1.06
TPRL	58.91%	116.67%	90.21%	61.89%	0.77	0.49	2.26	1.33
-0.03	80.61%	181.39%	58.40%	63.82%	1.36	0.85	4.13	2.55
-0.01	62.44%	125.34%	88.06%	60.36%	0.80	0.49	2.38	1.31
0	56.28%	106.58%	97.45%	60.56%	0.69	0.43	1.99	1.14
0.003	56.23%	104.29%	98.41%	60.52%	0.67	0.42	1.93	1.10
0.01	56.50%	100.74%	99.07%	60.60%	0.67	0.42	1.94	1.09
0.03	58.39%	95.22%	99.29%	60.75%	0.66	0.41	1.89	1.06

Table C.32: Results of experiment 4. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 4. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation. Sea-sea containers were not mixed with others.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	52.73%	52.74%	99.57%	46.15%	0.17	0.16	0.28	0.25
Random stacking	65.11%	97.37%	79.12%	49.84%	0.24	0.20	0.46	0.34
TVR (penalty 0.003)	50.12%	74.23%	89.54%	44.25%	0.17	0.15	0.27	0.23
TPRL	54.74%	83.28%	81.11%	44.99%	0.17	0.15	0.27	0.24
-0.03	79.83%	130.73%	56.92%	43.87%	0.21	0.19	0.38	0.30
-0.01	79.83%	130.73%	56.92%	43.87%	0.21	0.19	0.38	0.30
0	57.95%	91.89%	79.47%	44.34%	0.17	0.15	0.29	0.24
0.003	50.12%	74.23%	89.54%	44.25%	0.17	0.15	0.27	0.23
0.01	48.13%	70.63%	92.23%	44.35%	0.17	0.15	0.27	0.23
0.03	46.31%	65.15%	95.90%	44.49%	0.17	0.15	0.27	0.23

Table C.33: Results of experiment 4. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 3. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation. Sea-sea containers were not mixed with others.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	52.63%	52.64%	99.54%	48.53%	0.21	0.19	0.38	0.31
Random stacking	65.77%	118.35%	76.14%	54.38%	0.41	0.30	0.99	0.61
TVR (penalty 0.01)	41.55%	70.88%	96.27%	47.03%	0.21	0.17	0.38	0.29
TPRL	58.52%	115.71%	68.55%	48.81%	0.24	0.20	0.47	0.35
-0.03	84.74%	192.02%	43.56%	49.24%	0.38	0.29	0.87	0.57
-0.01	57.24%	114.25%	74.53%	46.48%	0.23	0.19	0.46	0.32
0	46.07%	85.99%	86.98%	46.89%	0.21	0.18	0.40	0.29
0.003	44.00%	79.65%	90.84%	46.88%	0.21	0.17	0.39	0.29
0.01	41.55%	70.88%	96.27%	47.03%	0.21	0.17	0.38	0.29
0.03	43.89%	64.91%	98.74%	47.14%	0.20	0.17	0.35	0.27

Table C.34: Results of experiment 4. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 4. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation. Sea-sea containers were not mixed with others.

C.9 Experiment 3 vs 4

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.92%	81.10%	99.60%	59.44%	0.48	0.33	1.30	0.76
Random stacking	69.44%	107.69%	89.56%	62.70%	0.73	0.47	2.09	1.23
TPRL	62.82%	95.35%	96.41%	57.78%	0.43	0.29	1.13	0.66
TVR (penalty 0.0)	62.06%	95.47%	99.28%	57.02%	0.40	0.27	1.03	0.62
TVR-PA (mixed, 0.0)	63.23%	95.49%	98.66%	57.13%	0.41	0.28	1.07	0.63
TVR-PA (unmixed, 0.0)	63.41%	95.58%	98.70%	57.11%	0.41	0.28	1.07	0.62
TVR (penalty 0.03)	63.63%	86.83%	99.65%	57.60%	0.40	0.28	1.02	0.61
TVR-PA (mixed, 0.03)	63.52%	90.60%	99.50%	57.69%	0.42	0.28	1.08	0.63
TVR-PA (unmixed, 0.03)	64.80%	90.13%	99.48%	57.43%	0.40	0.28	1.03	0.62

Table C.35: Results of experiment 3 vs 4. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 3. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.20%	80.60%	99.35%	63.17%	0.84	0.53	2.52	1.44
Random stacking	69.32%	134.11%	84.68%	69.41%	1.83	1.15	5.48	3.64
TPRL	58.91%	116.67%	90.21%	61.89%	0.77	0.49	2.26	1.33
TVR (penalty 0.0)	55.31%	112.72%	97.99%	59.50%	0.71	0.44	2.08	1.15
TVR-PA (mixed, 0.0)	56.09%	106.32%	97.39%	60.54%	0.70	0.43	2.02	1.14
TVR-PA (unmixed, 0.0)	56.28%	106.58%	97.45%	60.56%	0.69	0.43	1.99	1.14
TVR (penalty 0.03)	58.21%	94.02%	99.47%	60.74%	0.64	0.40	1.80	1.04
TVR-PA (mixed, 0.03)	57.64%	95.29%	99.31%	61.09%	0.69	0.43	1.98	1.12
TVR-PA (unmixed, 0.03)	58.39%	95.22%	99.29%	60.75%	0.66	0.41	1.89	1.06

Table C.36: Results of experiment 3 vs 4. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 4. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.92%	81.10%	99.60%	59.44%	0.48	0.33	1.30	0.76
Random stacking	69.44%	107.69%	89.56%	62.70%	0.73	0.47	2.09	1.23
TPRL	62.82%	95.35%	96.41%	57.78%	0.43	0.29	1.13	0.66
TVR (penalty 0.0)	45.91%	70.53%	94.42%	43.98%	0.16	0.14	0.26	0.22
TVR-PA (mixed, 0.0)	50.14%	74.29%	89.57%	44.28%	0.17	0.15	0.27	0.23
TVR-PA (unmixed, 0.0)	50.12%	74.23%	89.54%	44.25%	0.17	0.15	0.27	0.23
TVR (penalty 0.03)	47.86%	60.89%	99.55%	44.86%	0.16	0.15	0.26	0.22
TVR-PA (mixed, 0.03)	47.47%	61.33%	98.64%	44.90%	0.16	0.15	0.27	0.22
TVR-PA (unmixed, 0.03)	47.33%	61.80%	98.71%	44.67%	0.16	0.15	0.26	0.22

Table C.37: Results of experiment 3 vs 4. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 3. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.20%	80.60%	99.35%	63.17%	0.84	0.53	2.52	1.44
Random stacking	69.32%	134.11%	84.68%	69.41%	1.83	1.15	5.48	3.64
TPRL	58.91%	116.67%	90.21%	61.89%	0.77	0.49	2.26	1.33
TVR (penalty 0.0)	47.68%	96.34%	85.22%	45.85%	0.21	0.18	0.40	0.30
TVR-PA (mixed, 0.0)	46.07%	85.99%	86.98%	46.89%	0.21	0.18	0.40	0.29
TVR-PA (unmixed, 0.0)	46.07%	85.99%	86.98%	46.89%	0.21	0.18	0.40	0.29
TVR (penalty 0.03)	45.02%	64.37%	99.52%	47.23%	0.19	0.16	0.34	0.26
TVR-PA (mixed, 0.03)	44.42%	64.36%	98.72%	47.43%	0.20	0.17	0.37	0.28
TVR-PA (unmixed, 0.03)	43.89%	64.91%	98.74%	47.14%	0.20	0.17	0.35	0.27

Table C.38: Results of experiment 3 vs 4. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 4. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

C.10 Experiment 5 - all containers

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.92%	81.10%	99.60%	59.44%	0.48	0.33	1.30	0.76
Random stacking	69.44%	107.69%	89.56%	62.70%	0.73	0.47	2.09	1.23
TVR (penalty 0.0)	30.91%	44.28%	96.74%	51.86%	0.25	0.19	0.56	0.35
TPRL	62.82%	95.35%	96.41%	57.78%	0.43	0.29	1.13	0.66
-0.03	81.52%	139.52%	75.83%	58.83%	0.73	0.47	2.09	1.23
-0.01	46.00%	66.79%	88.40%	52.55%	0.28	0.21	0.66	0.44
0	30.91%	44.28%	96.74%	51.86%	0.25	0.19	0.56	0.35
0.003	30.38%	43.36%	97.40%	51.94%	0.25	0.19	0.57	0.35
0.01	30.16%	42.77%	97.95%	52.13%	0.26	0.20	0.61	0.36
0.02	30.65%	43.25%	98.05%	52.33%	0.28	0.20	0.68	0.38
0.03	31.03%	43.70%	98.04%	52.45%	0.29	0.21	0.70	0.39
0.04	31.00%	43.66%	98.01%	52.51%	0.30	0.21	0.72	0.40
0.05	31.27%	44.05%	98.01%	52.54%	0.29	0.21	0.73	0.41
0.1	30.84%	43.39%	97.99%	52.52%	0.30	0.21	0.73	0.40

Table C.39: Results of experiment 5. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 3. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	62.20%	80.60%	99.35%	63.17%	0.84	0.53	2.52	1.44
Random stacking	69.32%	134.11%	84.68%	69.41%	1.83	1.15	5.48	3.64
TVR (penalty 0.003)	18.83%	34.42%	94.75%	53.25%	0.30	0.21	0.72	0.44
TPRL	58.91%	116.67%	90.21%	61.89%	0.77	0.49	2.26	1.33
-0.03	82.06%	193.51%	58.43%	66.55%	1.91	1.20	5.74	3.84
-0.01	53.88%	103.24%	73.16%	56.63%	0.59	0.37	1.71	0.94
0	21.38%	39.70%	92.56%	53.31%	0.30	0.22	0.72	0.45
0.003	18.83%	34.42%	94.75%	53.25%	0.30	0.21	0.72	0.44
0.01	16.60%	29.07%	95.95%	53.29%	0.29	0.21	0.72	0.43
0.02	15.83%	26.89%	95.86%	53.32%	0.30	0.22	0.74	0.44
0.03	15.07%	24.79%	95.67%	53.24%	0.31	0.22	0.76	0.44
0.04	14.64%	23.71%	95.47%	53.17%	0.30	0.22	0.76	0.44
0.05	14.45%	23.29%	95.37%	53.16%	0.30	0.22	0.74	0.44
0.1	14.29%	23.08%	95.31%	53.14%	0.30	0.22	0.76	0.44

Table C.40: Results of experiment 5. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 4. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	52.73%	52.74%	99.57%	46.15%	0.17	0.16	0.28	0.25
Random stacking	65.11%	97.37%	79.12%	49.84%	0.24	0.20	0.46	0.34
TVR (penalty 0.003)	6.32%	8.39%	82.34%	39.73%	0.12	0.12	0.19	0.18
TPRL	54.74%	83.28%	81.11%	44.99%	0.17	0.15	0.27	0.24
-0.03	83.33%	142.43%	56.77%	44.98%	0.24	0.20	0.45	0.34
-0.01	39.62%	56.79%	68.62%	40.88%	0.15	0.13	0.23	0.21
0	10.63%	14.73%	80.24%	39.84%	0.13	0.12	0.19	0.18
0.003	6.32%	8.39%	82.34%	39.73%	0.12	0.12	0.19	0.18
0.01	1.92%	2.09%	84.14%	39.61%	0.12	0.12	0.18	0.18
0.02	0.48%	0.53%	84.34%	39.59%	0.12	0.12	0.18	0.18
0.03	0.24%	0.31%	84.43%	39.59%	0.12	0.12	0.18	0.18
0.04	0.28%	0.34%	84.46%	39.59%	0.12	0.12	0.18	0.18
0.05	0.28%	0.35%	84.48%	39.58%	0.12	0.12	0.18	0.18
0.1	0.26%	0.32%	84.43%	39.59%	0.12	0.12	0.18	0.18

Table C.41: Results of experiment 5. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 3. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	ETL	90% ETQ	90% ETL
Levelling	52.63%	52.64%	99.54%	48.53%	0.21	0.19	0.38	0.31
Random stacking	65.77%	118.35%	76.14%	54.38%	0.41	0.30	0.99	0.61
TVR (penalty 0.01)	1.16%	1.25%	81.61%	41.22%	0.13	0.13	0.20	0.19
TPRL	58.52%	115.71%	68.55%	48.81%	0.24	0.20	0.47	0.35
-0.03	86.82%	204.44%	43.40%	50.66%	0.45	0.32	1.11	0.68
-0.01	48.54%	90.75%	57.08%	43.49%	0.19	0.16	0.36	0.27
0	11.74%	20.35%	74.96%	41.51%	0.14	0.13	0.22	0.20
0.003	6.52%	10.51%	78.26%	41.36%	0.13	0.13	0.21	0.20
0.01	1.16%	1.25%	81.61%	41.22%	0.13	0.13	0.20	0.19
0.02	0.18%	0.19%	82.21%	41.23%	0.13	0.12	0.20	0.19
0.03	0.02%	0.02%	82.22%	41.24%	0.13	0.13	0.20	0.19
0.04	0.02%	0.02%	82.21%	41.23%	0.13	0.12	0.20	0.19
0.05	0.02%	0.02%	82.21%	41.23%	0.13	0.12	0.20	0.19
0.1	0.02%	0.02%	82.21%	41.23%	0.13	0.12	0.20	0.19

Table C.42: Results of experiment 5. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 4. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

C.11 Experiment 5 - sea-sea containers

	ROC	RDC	GPU	ASC	ETQ	90% ETQ
Levelling	55.17%	55.47%	99.43%	44.35%	0.18	0.31
Random stacking	64.68%	96.63%	78.55%	47.57%	0.26	0.53
TVR (penalty 0.03)	0.67%	0.72%	83.58%	38.16%	0.12	0.19
TPRL	55.73%	84.75%	81.03%	43.35%	0.18	0.31
-0.03	81.69%	138.84%	55.95%	42.78%	0.25	0.50
-0.01	51.79%	75.73%	63.82%	39.59%	0.16	0.26
0	23.61%	33.96%	74.17%	38.65%	0.13	0.21
0.003	17.51%	24.76%	76.59%	38.50%	0.13	0.20
0.01	6.83%	8.83%	81.70%	38.24%	0.12	0.19
0.02	2.22%	2.36%	83.34%	38.16%	0.12	0.19
0.03	0.67%	0.72%	83.58%	38.16%	0.12	0.19
0.04	0.22%	0.29%	83.72%	38.16%	0.12	0.19
0.05	0.28%	0.38%	83.81%	38.17%	0.12	0.19
0.1	0.28%	0.38%	83.75%	38.16%	0.12	0.19

Table C.43: Results of experiment 5. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 3. The "90%" values are the average 90% percentile values. All normal 20 ft. sea-sea containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	90% ETQ
Levelling	55.09%	55.40%	99.39%	46.68%	0.23	0.45
Random stacking	65.32%	117.36%	75.68%	51.93%	0.46	1.14
TVR (penalty 0.03)	0.31%	0.31%	81.64%	39.60%	0.13	0.20
TPRL	59.29%	117.48%	68.60%	46.54%	0.26	0.53
-0.03	84.99%	199.23%	42.41%	47.85%	0.47	1.17
-0.01	60.33%	117.63%	51.89%	42.23%	0.23	0.45
0	26.27%	49.24%	66.55%	40.26%	0.16	0.28
0.003	18.84%	34.41%	70.47%	39.99%	0.15	0.25
0.01	6.31%	9.78%	78.24%	39.67%	0.14	0.21
0.02	1.34%	1.37%	81.27%	39.59%	0.13	0.20
0.03	0.31%	0.31%	81.64%	39.60%	0.13	0.20
0.04	0.03%	0.03%	81.81%	39.60%	0.13	0.20
0.05	0.01%	0.01%	81.96%	39.64%	0.13	0.20
0.1	0.01%	0.01%	81.95%	39.64%	0.13	0.20

Table C.44: Results of experiment 5. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 4. The "90%" values are the average 90% percentile values. All normal 20 ft. sea-sea containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	90% ETQ
Levelling	31.78%	31.78%	97.60%	33.76%	0.13	0.19
Random stacking	58.73%	83.74%	68.41%	37.47%	0.17	0.26
TVR (penalty 0.03)	0.03%	0.03%	62.63%	29.60%	0.11	0.16
TPRL	55.28%	84.03%	60.62%	33.65%	0.14	0.20
-0.03	82.87%	141.20%	41.99%	32.37%	0.16	0.25
-0.01	45.63%	66.16%	49.29%	30.28%	0.12	0.18
0	14.16%	19.70%	58.25%	29.79%	0.11	0.16
0.003	8.42%	11.21%	60.31%	29.72%	0.11	0.16
0.01	2.51%	2.71%	62.20%	29.62%	0.11	0.16
0.02	0.54%	0.54%	62.47%	29.60%	0.11	0.16
0.03	0.03%	0.03%	62.63%	29.60%	0.11	0.16
0.04	0.00%	0.00%	62.61%	29.58%	0.11	0.16
0.05	0.00%	0.00%	62.61%	29.58%	0.11	0.16
0.1	0.00%	0.00%	62.61%	29.58%	0.11	0.16

Table C.45: Results of experiment 5. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 3. The "90%" values are the average 90% percentile values. All normal 20 ft. sea-sea containers were included in the simulation.

	ROC	RDC	GPU	ASC	ETQ	90% ETQ
Levelling	31.66%	31.66%	97.58%	35.41%	0.14	0.21
Random stacking	59.11%	97.31%	66.72%	40.38%	0.21	0.36
TVR (penalty 0.03)	0.01%	0.01%	61.09%	30.80%	0.11	0.17
TPRL	58.79%	116.26%	51.24%	36.29%	0.16	0.26
-0.03	86.95%	204.00%	31.85%	36.22%	0.23	0.41
-0.01	54.93%	105.32%	40.47%	32.08%	0.15	0.23
0	15.65%	27.40%	53.89%	30.92%	0.12	0.18
0.003	8.73%	14.11%	57.04%	30.82%	0.12	0.17
0.01	1.53%	1.62%	60.44%	30.77%	0.11	0.17
0.02	0.25%	0.25%	60.96%	30.78%	0.11	0.17
0.03	0.01%	0.01%	61.09%	30.80%	0.11	0.17
0.04	0.00%	0.00%	61.09%	30.81%	0.11	0.17
0.05	0.00%	0.00%	61.09%	30.81%	0.11	0.17
0.1	0.00%	0.00%	61.09%	30.81%	0.11	0.17

Table C.46: Results of experiment 5. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 4. The "90%" values are the average 90% percentile values. All normal 20 ft. sea-sea containers were included in the simulation.

C.12 Experiment 6 - all containers

	ROC	RDC	GPU	STU	ASC	ETQ	ETL	90% ETQ	90% ETL	NTQ	NTL	90% NTQ	90% NTL
Levelling	62.92%	81.10%	99.60%	74.92%	59.44%	0.48	0.33	1.30	0.76	0.30	0.24	3.61	0.72
Random stacking	69.44%	107.69%	89.56%	75.00%	62.70%	0.73	0.47	2.09	1.23	0.43	0.34	4.84	1.16
TPRL	62.82%	95.35%	96.41%	74.91%	57.78%	0.43	0.29	1.13	0.66	0.27	0.20	3.30	0.64
TVR-DTC													
0.01	30.16%	42.77%	97.95%	74.84%	52.13%	0.26	0.20	0.61	0.36	0.19	0.15	2.15	0.37
0.02	30.65%	43.25%	98.05%	74.85%	52.33%	0.28	0.20	0.68	0.38	0.19	0.16	2.48	0.39
0.03	31.03%	43.70%	98.04%	74.85%	52.45%	0.29	0.21	0.70	0.39	0.20	0.16	2.54	0.40
0.04	31.00%	43.66%	98.01%	74.86%	52.51%	0.30	0.21	0.72	0.40	0.20	0.16	2.47	0.41
LDT-DTC	12.20%	18.08%	88.67%	74.82%	50.42%	0.22	0.18	0.44	0.32	0.17	0.14	2.17	0.34
TVR-DTC-MD													
0.01rm, 0.003 cd	23.00%	31.50%	95.86%	74.82%	51.15%	0.22	0.18	0.47	0.31	0.17	0.14	1.64	0.33
0.01rm, 0.01 cd	22.78%	29.88%	91.60%	74.81%	50.65%	0.19	0.17	0.38	0.30	0.17	0.13	1.37	0.32
0.01rm, 0.03 cd	45.18%	63.55%	84.30%	74.86%	53.72%	0.33	0.24	0.85	0.49	0.22	0.17	2.85	0.50
0.01rm, 0.04 cd	46.71%	66.20%	83.82%	74.87%	54.18%	0.34	0.25	0.92	0.52	0.23	0.18	2.83	0.53
0.02rm, 0.003 cd	24.83%	34.33%	96.59%	74.84%	51.53%	0.25	0.19	0.57	0.34	0.18	0.15	2.14	0.36
0.02rm, 0.01 cd	19.71%	25.27%	93.33%	74.81%	50.54%	0.20	0.17	0.39	0.30	0.17	0.13	1.43	0.32
0.02rm, 0.03 cd	37.58%	50.61%	86.36%	74.84%	52.38%	0.26	0.20	0.61	0.40	0.19	0.15	2.17	0.41
0.02rm, 0.04 cd	44.16%	61.83%	84.35%	74.86%	53.68%	0.32	0.24	0.83	0.49	0.22	0.17	2.82	0.50
TVR-DTC-MD													
0.03rm, 0.003 cd	26.33%	36.63%	96.86%	74.85%	51.79%	0.27	0.20	0.63	0.36	0.19	0.15	2.34	0.37
0.03rm, 0.01 cd	19.50%	25.09%	94.09%	74.82%	50.71%	0.21	0.17	0.43	0.31	0.17	0.14	1.65	0.33
0.03rm, 0.03 cd	32.32%	41.57%	87.85%	74.82%	51.60%	0.23	0.19	0.51	0.36	0.18	0.14	1.98	0.37
0.03rm, 0.04 cd	37.77%	50.31%	86.19%	74.84%	52.49%	0.27	0.21	0.63	0.40	0.20	0.16	2.49	0.42
0.04rm, 0.003 cd	27.21%	37.98%	96.94%	74.85%	52.00%	0.28	0.20	0.68	0.37	0.19	0.16	2.39	0.39
0.04rm, 0.01 cd	20.37%	26.63%	94.69%	74.82%	50.92%	0.22	0.18	0.48	0.32	0.18	0.14	1.82	0.35
0.04rm, 0.03 cd	28.84%	35.58%	88.97%	74.82%	51.16%	0.22	0.18	0.45	0.34	0.18	0.14	1.62	0.36
0.04rm, 0.04 cd	33.47%	42.71%	87.45%	74.83%	51.76%	0.24	0.19	0.53	0.37	0.19	0.15	2.33	0.38

Table C.47: Results of experiment 6. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 3. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

	ROC	RDC	GPU	STU	ASC	ETQ	ETL	90% ETQ	90% ETL	NTQ	NTL	90% NTQ	90% NTL
Levelling	62.20%	80.60%	99.35%	56.26%	63.17%	0.84	0.53	2.52	1.44	0.53	0.41	5.63	1.45
Random stacking	69.32%	134.11%	84.68%	56.42%	69.41%	1.83	1.15	5.48	3.64	1.21	0.96	9.34	3.56
TPRL	58.91%	116.67%	90.21%	56.24%	61.89%	0.77	0.49	2.26	1.33	0.49	0.36	5.27	1.36
TVR-DTC													
0.01	16.60%	29.07%	95.95%	56.13%	53.29%	0.29	0.21	0.72	0.43	0.22	0.17	2.73	0.47
0.02	15.83%	26.89%	95.86%	56.13%	53.32%	0.30	0.22	0.74	0.44	0.22	0.17	2.65	0.48
0.03	15.07%	24.79%	95.67%	56.13%	53.24%	0.31	0.22	0.76	0.44	0.23	0.17	2.81	0.50
0.04	14.64%	23.71%	95.47%	56.13%	53.17%	0.30	0.22	0.76	0.44	0.23	0.17	2.78	0.50
LDT-DTC	1.51%	3.05%	75.39%	56.10%	50.64%	0.20	0.17	0.40	0.31	0.19	0.14	1.77	0.38
TVR-DTC-MD													
0.01rm, 0.003 cd	9.69%	14.99%	93.34%	56.11%	52.17%	0.22	0.18	0.47	0.35	0.19	0.15	1.70	0.40
0.01rm, 0.01 cd	10.69%	15.03%	87.51%	56.10%	51.75%	0.20	0.18	0.42	0.33	0.18	0.14	1.41	0.38
0.01rm, 0.03 cd	25.69%	41.79%	80.47%	56.13%	53.76%	0.30	0.23	0.75	0.51	0.23	0.18	2.67	0.55
0.01rm, 0.04 cd	26.09%	42.73%	80.34%	56.13%	53.85%	0.30	0.23	0.75	0.51	0.23	0.17	2.67	0.54
0.02rm, 0.003 cd	10.82%	16.91%	93.80%	56.12%	52.41%	0.25	0.19	0.55	0.38	0.20	0.15	2.12	0.43
0.02rm, 0.01 cd	6.77%	8.22%	90.02%	56.10%	51.54%	0.20	0.17	0.39	0.32	0.18	0.14	1.44	0.37
0.02rm, 0.03 cd	18.96%	26.71%	82.45%	56.11%	52.59%	0.25	0.20	0.57	0.41	0.20	0.16	1.90	0.44
0.02rm, 0.04 cd	24.29%	36.63%	80.81%	56.12%	53.47%	0.29	0.22	0.71	0.48	0.23	0.17	2.59	0.52
TVR-DTC-MD													
0.03rm, 0.003 cd	10.98%	17.03%	93.73%	56.12%	52.46%	0.26	0.20	0.61	0.38	0.21	0.16	2.34	0.45
0.03rm, 0.01 cd	6.21%	7.43%	90.77%	56.10%	51.65%	0.20	0.17	0.41	0.32	0.18	0.14	1.60	0.38
0.03rm, 0.03 cd	14.85%	18.20%	83.70%	56.11%	52.03%	0.23	0.19	0.51	0.37	0.20	0.15	1.64	0.42
0.03rm, 0.04 cd	19.05%	24.94%	82.19%	56.12%	52.58%	0.25	0.20	0.58	0.41	0.21	0.16	2.05	0.45
0.04rm, 0.003 cd	11.39%	17.71%	93.66%	56.12%	52.55%	0.27	0.21	0.65	0.41	0.21	0.16	2.36	0.46
0.04rm, 0.01 cd	6.92%	8.82%	91.35%	56.10%	51.80%	0.21	0.18	0.45	0.34	0.19	0.15	1.88	0.39
0.04rm, 0.03 cd	12.40%	13.94%	84.57%	56.11%	51.80%	0.22	0.19	0.47	0.35	0.19	0.15	1.74	0.40
0.04rm, 0.04 cd	15.41%	18.19%	83.36%	56.11%	52.09%	0.23	0.19	0.51	0.37	0.20	0.15	1.79	0.41

Table C.48: Results of experiment 6. A terminal of 6 lanes of 34x6 positions each was used, with a maximum stacking height of 4. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

	ROC	RDC	GPU	STU	ASC	ETQ	ETL	90% ETQ	90% ETL	NTQ	NTL	90% NTQ	90% NTL
Levelling	52.73%	52.74%	99.57%	56.11%	46.15%	0.17	0.16	0.28	0.25	0.13	0.12	0.90	0.20
Random stacking	65.11%	97.37%	79.12%	56.13%	49.84%	0.24	0.20	0.46	0.34	0.15	0.14	1.70	0.26
TPRL	54.74%	83.28%	81.11%	56.11%	44.99%	0.17	0.15	0.27	0.24	0.12	0.11	0.95	0.19
TVR-DTC													
0.01	1.92%	2.09%	84.14%	56.09%	39.61%	0.12	0.12	0.18	0.18	0.11	0.10	0.53	0.16
0.02	0.48%	0.53%	84.34%	56.09%	39.59%	0.12	0.12	0.18	0.18	0.11	0.10	0.54	0.16
0.03	0.24%	0.31%	84.43%	56.09%	39.59%	0.12	0.12	0.18	0.18	0.11	0.10	0.59	0.16
0.04	0.28%	0.34%	84.46%	56.09%	39.59%	0.12	0.12	0.18	0.18	0.11	0.10	0.60	0.16
LDT-DTC	0.63%	0.94%	67.83%	56.09%	38.43%	0.12	0.12	0.19	0.18	0.12	0.11	0.69	0.18
TVR-DTC-MD													
0.01rm, 0.003 cd	3.75%	4.18%	78.43%	56.09%	39.49%	0.12	0.12	0.18	0.18	0.11	0.10	0.54	0.16
0.01rm, 0.01 cd	15.02%	18.70%	71.33%	56.10%	39.80%	0.13	0.12	0.19	0.19	0.12	0.11	0.58	0.17
0.01rm, 0.03 cd	44.76%	62.52%	63.70%	56.11%	42.63%	0.16	0.15	0.27	0.23	0.13	0.11	1.16	0.20
0.01rm, 0.04 cd	46.08%	64.75%	63.41%	56.11%	42.92%	0.17	0.15	0.28	0.24	0.13	0.12	1.29	0.20
0.02rm, 0.003 cd	1.49%	1.50%	79.57%	56.09%	39.44%	0.12	0.12	0.18	0.18	0.11	0.10	0.58	0.16
0.02rm, 0.01 cd	8.38%	9.14%	73.45%	56.09%	39.43%	0.12	0.12	0.19	0.19	0.12	0.11	0.57	0.17
0.02rm, 0.03 cd	35.56%	47.50%	65.70%	56.10%	41.47%	0.15	0.14	0.23	0.21	0.12	0.11	0.92	0.19
0.02rm, 0.04 cd	43.51%	60.35%	63.83%	56.11%	42.56%	0.16	0.15	0.27	0.23	0.13	0.11	1.18	0.20
TVR-DTC-MD													
0.03rm, 0.003 cd	0.57%	0.57%	80.08%	56.09%	39.41%	0.12	0.12	0.18	0.18	0.11	0.10	0.56	0.16
0.03rm, 0.01 cd	5.09%	5.18%	74.97%	56.09%	39.41%	0.12	0.12	0.18	0.18	0.12	0.11	0.58	0.16
0.03rm, 0.03 cd	29.41%	37.05%	67.16%	56.10%	40.75%	0.14	0.13	0.22	0.20	0.12	0.11	0.76	0.18
0.03rm, 0.04 cd	35.48%	46.73%	65.66%	56.10%	41.46%	0.15	0.14	0.23	0.21	0.12	0.11	0.88	0.19
0.04rm, 0.003 cd	0.10%	0.10%	80.26%	56.09%	39.43%	0.12	0.12	0.18	0.18	0.11	0.10	0.54	0.16
0.04rm, 0.01 cd	3.37%	3.38%	75.98%	56.09%	39.35%	0.12	0.12	0.18	0.18	0.12	0.11	0.56	0.16
0.04rm, 0.03 cd	25.33%	30.24%	68.22%	56.10%	40.31%	0.13	0.13	0.21	0.20	0.12	0.11	0.72	0.18
0.04rm, 0.04 cd	30.44%	38.05%	66.89%	56.10%	40.85%	0.14	0.13	0.22	0.21	0.12	0.11	0.73	0.18

Table C.49: Results of experiment 6. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 3. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

	ROC	RDC	GPU	STU	ASC	ETQ	ETL	90% ETQ	90% ETL	NTQ	NTL	90% NTQ	90% NTL
Levelling	52.63%	52.64%	99.54%	42.09%	48.53%	0.21	0.19	0.38	0.31	0.15	0.14	1.40	0.26
Random stacking	65.77%	118.35%	76.14%	42.13%	54.38%	0.41	0.30	0.99	0.61	0.24	0.19	3.23	0.52
TPRL	58.52%	115.71%	68.55%	42.10%	48.81%	0.24	0.20	0.47	0.35	0.16	0.14	1.59	0.29
TVR-DTC													
0.01	1.16%	1.25%	81.61%	42.07%	41.22%	0.13	0.13	0.20	0.19	0.12	0.11	0.78	0.18
0.02	0.18%	0.19%	82.21%	42.07%	41.23%	0.13	0.12	0.20	0.19	0.12	0.11	0.66	0.18
0.03	0.02%	0.02%	82.22%	42.07%	41.24%	0.13	0.13	0.20	0.19	0.12	0.11	0.73	0.18
0.04	0.02%	0.02%	82.21%	42.07%	41.23%	0.13	0.12	0.20	0.19	0.12	0.11	0.72	0.18
LDT-DTC	0.06%	0.12%	56.70%	42.07%	39.96%	0.13	0.12	0.20	0.19	0.13	0.11	0.80	0.19
TVR-DTC-MD													
0.01rm, 0.003 cd	2.31%	2.53%	75.22%	42.07%	41.07%	0.13	0.12	0.20	0.19	0.12	0.11	0.71	0.18
0.01rm, 0.01 cd	8.21%	10.49%	67.59%	42.07%	41.09%	0.13	0.13	0.20	0.20	0.12	0.11	0.68	0.18
0.01rm, 0.03 cd	25.78%	40.93%	60.83%	42.07%	42.72%	0.15	0.15	0.25	0.23	0.13	0.12	1.01	0.21
0.01rm, 0.04 cd	25.93%	41.21%	60.79%	42.08%	42.73%	0.15	0.15	0.25	0.23	0.13	0.12	1.04	0.21
0.02rm, 0.003 cd	0.94%	0.95%	76.56%	42.07%	41.06%	0.13	0.12	0.19	0.19	0.12	0.11	0.67	0.18
0.02rm, 0.01 cd	3.99%	4.15%	69.68%	42.07%	40.92%	0.13	0.13	0.20	0.20	0.12	0.11	0.69	0.18
0.02rm, 0.03 cd	17.87%	23.53%	62.59%	42.07%	41.69%	0.14	0.14	0.22	0.21	0.13	0.12	0.88	0.20
0.02rm, 0.04 cd	23.70%	34.46%	61.19%	42.07%	42.44%	0.15	0.14	0.24	0.22	0.13	0.12	0.89	0.21
TVR-DTC-MD													
0.03rm, 0.003 cd	0.35%	0.35%	77.07%	42.07%	41.05%	0.13	0.12	0.19	0.19	0.12	0.11	0.75	0.18
0.03rm, 0.01 cd	2.90%	2.92%	70.76%	42.07%	40.98%	0.13	0.13	0.20	0.19	0.12	0.11	0.69	0.18
0.03rm, 0.03 cd	13.78%	15.78%	63.76%	42.07%	41.28%	0.14	0.13	0.21	0.21	0.13	0.11	0.80	0.20
0.03rm, 0.04 cd	17.83%	22.18%	62.58%	42.07%	41.71%	0.14	0.14	0.22	0.21	0.13	0.12	0.89	0.20
0.04rm, 0.003 cd	0.04%	0.04%	77.31%	42.07%	41.08%	0.13	0.12	0.19	0.19	0.12	0.11	0.74	0.18
0.04rm, 0.01 cd	2.02%	2.02%	71.90%	42.07%	40.99%	0.13	0.13	0.20	0.19	0.12	0.11	0.73	0.18
0.04rm, 0.03 cd	11.61%	12.41%	64.41%	42.07%	41.14%	0.13	0.13	0.21	0.20	0.13	0.11	0.76	0.19
0.04rm, 0.04 cd	14.18%	15.83%	63.56%	42.07%	41.34%	0.14	0.13	0.22	0.21	0.13	0.12	0.79	0.20

Table C.50: Results of experiment 6. A terminal of 8 lanes of 34x6 positions each was used, with a maximum stacking height of 4. The "90%" values are the average 90% percentile values. All normal 20 ft. containers were included in the simulation.

Bibliography

[1] Apache POI project. Apache POI - Java API To Access Microsoft Format Files. Website, 2002-2008. See http://poi.apache.org/; retrieved October 30, 2008.

[2] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol. *Discrete-Event System Simulation*. Prentice Hall, London, third edition, 2001. Cited by[8].

[3] B. Borgman. Improving a scheduled short sea shipping network using simulation, July 2007. Erasmus University Rotterdam, School of Economics. Bachelor's thesis Informatics & Economics.

[4] B. Borgman, H. van der Goot, and D. Jekkers. Werkcollege simulatie: Simulating dynamic re-planning for security service companies, 2007. Erasmus University Rotterdam, School of Economics.

[5] R. Dekker, P. Voogd, and E. van Asperen. Advanced methods for container stacking. *OR Spectrum*, 28:563–586, 2006.

[6] E. W. Dijkstra. The humble programmer, 1972. ACM Turing Lecture.

[7] M. B. Duinkerken, J. J. Evers, and J. A. Ottjes. A simulation model for integrating quay transport and stacking policies in automated terminals. In *Proceedings of the 15th European Simulation Multiconference (ESM2001)*, Prague, 2001. SCS.

[8] T. Eldabi, M. W. Lee, and R. J. Paul. A framework for business process simulation: the grab and glue approach. In A. Verbraeck and V. Hlupic, editors, *Proceedings 15th European Simulation Symposium*, 2003.

[9] G. Froyland, T. Koch, N. Megow, E. Duane, and H. Wren. Optimizing the landside operation of a container terminal. *OR Spectrum*, 30:53–75, 2008.

[10] Y. Han, L. H. Lee, E. P. Chew, and K. C. Tan. A yard storage strategy for minimizing traffic congestion in a marine container transshipment hub. *OR Spectrum*, 30:697–720, 2008.

[11] Y. Hirashima, K. Takeda, S. Harada, M. Deng, and A. Inoue. A Q-learning for group-based plan of container transfer scheduling. *JSME International Journal Series C*, 49(2):473–479, 2006.

[12] P. H. Jacobs, A. Verbraeck, and J. B. Mulder. Flight scheduling at KLM. In M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, editors, *Proceedings of the 2005 Winter Simulation Conference*, pages 299–306, 2005.

[13] J. Kang, M.-S. Oh, E. Y. Ahn, K. R. Ryu, and K. H. Kim. Planning for intra-block remarshalling in a container terminal. In M. Ali and R. Dapoigny, editors, *Advances in applied artificial intelligence*, pages 1211–1220. Springer, 2006.

[14] W. D. Kelton, R. P. Sadowski, and D. T. Sturrock. *Simulation with Arena*. McGraw-Hill, New York, third edition, 2003.

[15] K. H. Kim and G.-P. Hong. A heuristic rule for relocating blocks. *Computers & Operations Research*, 33:940–954, 2006.

[16] P. L'Ecuyer and E. Buist. Simulation in java with SSJ. In M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, editors, *Proceedings of the 2005 Winter Simulation Conference*, pages 611–620, 2005.

[17] P. L'Ecuyer, L. Meliani, and J. Vaucher. SSJ: A framework for stochastic simulation in java. In J. L. Snowdon and J. M. Charnes, editors, *Proceedings of the 2002 Winter Simulation Conference*, pages 234–242, 2002.

[18] A. Lim and Z. Xu. A critical-shaking neighborhood search for the yard allocation problem. *European Journal of Operational Research*, 174:1247–1259, 2006.

[19] D. K. Pace. Ideas about a simulation conceptual model development. *Johns Hopkins APL Technical Digest*, 21(3):327–336, 2000.

[20] B. J. Park, H. R. Choi, H. K. Kwon, and M. H. Kang. *AI 2006: Advances in Artificial Intelligence*, volume 4304 of *Lecture Notes in Computer Science*, chapter Simulation Analysis on Effective Operation of Handling Equipments in Automated Container Terminal, pages 1231–1238. Springer Verlag, Berlin/Heidelberg, 2006.

[21] S. Robinson. *Simulation: The Practice of Model Development and Use*. Wiley, Chichester, UK, 2004. Cited by [22].

[22] S. Robinson. Conceptual modeling for simulation: issues and research requirements. In L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, editors, *Proceedings of the 2006 Winter Simulation Conference*, pages 792–800, 2006.

[23] Y. A. Saanen and R. Dekker. Intelligent stacking as way out of congested yards? part 1. *Port Technology International*, 31:87–92, 2006.

[24] Y. A. Saanen and R. Dekker. Intelligent stacking as way out of congested yards? part 2. *Port Technology International*, 32:80–86, 2006.

[25] R. G. Sargent. Verification and validation of simulation models. In M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, editors, *Proceedings of the 2005 Winter Simulation Conference*, pages 130–143, 2005.

[26] S. Schlesinger. Terminology for model credibility. *Simulation*, 32(2):103–104, 1979. Cited by [25].

[27] R. E. Shannon. *Systems Simulation: The Art and Science*. Prentice-Hall, Englewood Cliffs, New Jersey, 1975. Cited by [22].

[28] R. Stahlbock and S. Voss. Operations research at container terminals: a literature update. *OR Spectrum*, 30:1–52, 2008.

[29] D. Steenken, S. Voss, and R. Stahlbock. Container terminal operation and operations research a classification and literature review. *OR Spectrum*, 26:3–49, 2004.

[30] M. Stopford. *Maritime economics*. Routledge, London, second edition, 2002.

[31] R. Thijs, P. Bovy, and T. J. Schoemaker. *Integrale rapportage FAMAS*. Connekt, 2000. TRAIL report.

[32] Various TRAIL members. NewCon definitiestudie. Technical report, Centrum voor transporttechnologie (CTT), 1998.

[33] P. Voogd, R. Dekker, and P. J. Meersmans. FAMAS-Newcon: a generator program for stacking in the reference case. Technical Report EI-9943/A, Erasmus University Rotterdam - Econometric Institute, 1999.

[34] W. Wang and R. J. Brooks. Empirical investigations of conceptual modeling and the modeling process. In S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, editors, *Proceedings of the 2007 Winter Simulation Conference*, pages 762–770, 2007.

[35] N. Warrens. Distilled FAMAS-Newcon data spreadsheet. Internal document, 2008.

[36] H. Weng. Container stacking rules with computational technology. Master's thesis, Informatics & Economics, Erasmus University Rotterdam, School of Economics, August 2007.

[37] R. B. Whitner and O. Balci. Guidelines for selecting and using simulation model verification techniques. In E. MacNair, K. Musselman, and P. Heidelberger, editors, *Proceedings of the 1989 Winter Simulation Conference*, pages 559–568, 1989.