Bachelor Thesis

Discrete event simulation of a container stack with a twin-automatic stacking cranes setup.

Ho Kai Man
Student number: 253119

Supervisor: Prof. Dr. Ir. Rommert Dekker
Second reader: J. Mulder, MSc
Erasmus University Rotterdam
Erasmus School of Economics
Economics & Informatics programme

Index

1. Introduction

In the past decades world trade have developed rapidly. Especially the Asia-Europe and Asia-US trade. Because of this development the use of containers has increased rapidly as well, which in turn has led to bigger containerships to cope with the demand. With bigger ships, the speed in which a maritime container terminal can load and/or unload a container, can become a bottleneck in the process. To increase the productivity of a terminal, we have to look for ways to improve the process.

The process at a maritime terminal can be described as follows(see fig. 1). The container ship arrives at the terminal and berth at a predetermined location. After berthing the containers are unloaded from the ship through quay cranes. The unloaded containers are then transported either by human driven vehicles or automatic guided vehicles (AGV) to the stack. The stack is a big yard reserved for storage of the containers, divided in different lanes in this situation. The storage of the containers into the stack is done either by human driven cranes or by automatic stacking cranes (ASC). When it is time for a container to be transported further, the human driven crane or the ASC will retrieve the crane from the stack and place it on the landside or the seaside of the stack, depending on the mode of transportation. The landside generally includes trucks and trains . The seaside generally includes barges, feeders, short sea ships, deep sea ships and jumbo ships.

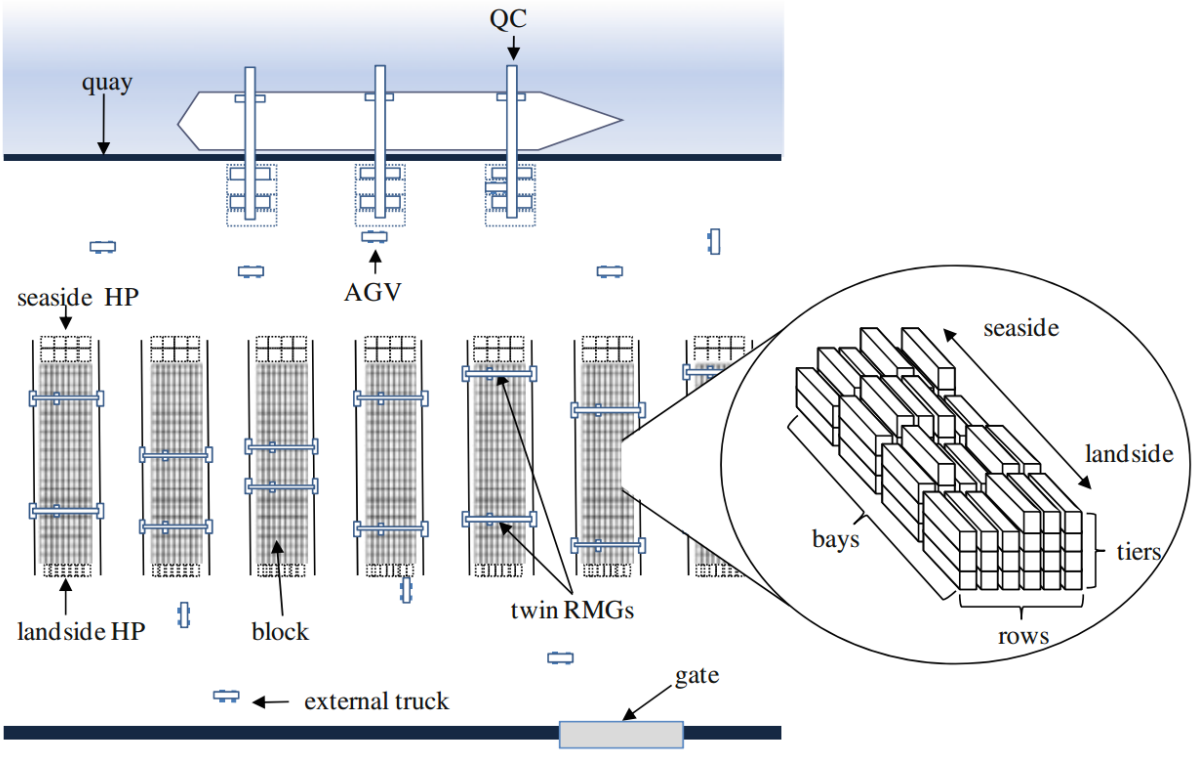


Fig. 1 example automated terminal (source: Park et al. , 2010)

Figure 1 is an example of an automated container terminal. The container stack is divided in multiple lanes (also known as blocks) and each block has one or two rail mounted gantries, referred here as ASC, servicing it. Each lane consists of multiple bays, which in turn consist of multiple piles of containers, that are multiple rows wide.

One of the main problems of a terminal is stacking the containers optimally in the stack. That is which container should be stacked on which other container. The way the containers are stacked has an impact on the time required to retrieve them from the stack, e.g. if a container to be retrieved is stacked below another, first the other container has to be retrieved, before it can be handled. This other retrieval is a kind of improductive move, which should be avoided, but that is not always possible. For this reason stacking strategies are employed.

Stacking cranes are often a bottleneck in handling, as they have to export containers from the stack to a ship and to import containers brought from a landside for a next ship. Another way to improve the productivity of a terminal is using more than one ASC per lane. Currently there are three kinds of setups of multiple ASC. A setup where each stack has two non-passing ASC, also called Twin ASC or Twin RMG(rail mounted gantry).They are non-passing as in they can't pass each other and thus can service only one side of the lane. One example where it is employed currently is in the Euromax Terminal located in Rotterdam in the Netherlands by ECT. The Container Terminal Altenwerder in Hamburg, Germany, is currently using the second type of setup(also known as Dual ASC). It also uses two ASC, but they can pass each other. One is larger than the other. More recently a third setup has been also been deployed. The triple cross-over stacking cranes layout has been implemented at the Container Terminal Burchardkai (CTB) in Hamburg in an effort to double the terminal throughput capacity(Dorndorf and Schneider, 2012).

1.1 Problem description

Most literature on stacking deals with just one ASC per stack lane. The literature about multiple ASC in one lane is relatively small. As mentioned before currently containers terminals are employing different kinds of setups with multiple ASCs, in an effort to increase their throughput. In the literature available, one focuses either on analytical calculations or simulations with less focus on stacking strategies with two ASCs. In this thesis I will investigate stacking strategies with two ASCs by developing a discrete event simulation using a random stacking strategy as base strategy and a leveling stacking strategy and compare them. In addition I will also use a variant of the two stacking strategies: random stacking with departure times and leveling with departure times. In combination with these stacking strategies, I will use an ASC scheduling strategy which is not based on segments. Since the focus in on the twin-ASC setup, this thesis will focused on just one lane (block) with two ASC.

1.2 Outline

This thesis has the following outline. First it will describe in short the relevant literature about stacking strategies and the use of multiple cranes. Secondly, I will discuss the stacking strategies that I will use in the simulation and also an ASC workload strategy. After that, the way the data, used as input for the simulator, is being generated. A description of the simulator will follow .In the next chapter a short description of the applications and the development of it. In the chapter after that, I will show the result of the simulation and discuss the results. I will conclude this thesis with my conclusions and discuss research ideas that could be done in the future.

2. Literature

The way containers are stored in the stack will have significant effect on their retrieval time. Therefore using an efficient stacking strategy is important. The main objectives of a stacking strategy are: the minimization of the amount of reshuffles, minimization of transportation time within the terminal and efficient use of the available storage space. A reshuffle is the relocation of a container on top of the target container.

There are two main ways to approach the stacking problem (Dekker et al, 2006). Through analytical calculations and with detailed simulation studies. Kim and Hong (2006) make use of analytical calculations to tackle the stacking problem. They focused on the relocation of already stacked containers and the pickup sequence of the containers using a branch and bound algorithm and a probabilistic heuristic. The disadvantage of the proposed branch and bound method is the high computational cost. It could take up to 4 minutes to optimally solve a 5 piles x 5 tiers. That is too long a time to be useful in practical use. Zhu et al (2010) improved this algorithm by combining it with Iterative Deepening A-star and probing heuristics. It can now solve a 7 piles x 7 tiers near optimal within 1 second.

Since analytical calculations can be very costly in terms of time, other methods has been proposed. In Dekker et al (2006) and Borgman et al (2010) online stacking strategies are used in detailed simulations studies. The strategies attempts to optimize the placement of the containers as they arrive to the stack. The main focus of the strategies are minimization of reshuffles and travel time to the transport point. The advantage of online stacking strategies is that they are light in computational cost and can be used immediately by a stacking crane to find the optimal place to stack a container. The algorithms used in Dekker et al(2006) and Borgman et al(2010) perform significantly better than the baseline stacking strategy of random stacking.

There are relative few papers on the subject of using twin ASC on the same lane specifically. But there are more papers on the subject of multiple ASC on one lane. Saanen and Valkengoed (2005) compares the performance of three different of setups of ASC. The three setups are: one ASC per lane, two non-passing ASC (twin ASC) per lane and two passing ASC (dual ASC) per lane. Detailed simulations were run with those setups using nearest neighbor heuristic. As expected the two ASC per lane performed better than the one ASC per lane. Although the passing ASC performed better than the non-passable ASC, the benefit is relatively small. It seems the benefits of a passing ASC, doesn't outweigh the cost when compared to twin ASC. Iris and Hector (2010) used a mathematical model and simulated annealing based heuristic to efficiently determine the scheduling of two passing ASC. Saanen(2011) discusses the advantages of using twin ASC over the dual ASC. Park et al(2010) propose a real-time scheduling algorithm for the twin ASC setup as a ASC scheduling strategy. They use a heuristic based dispatching of jobs to ASCs with dividing the stack into three segments. Zyngriridis(2005) uses Integer Linear Programs to schedule routes for one and two ASC in a single lane. Zhou and Wu(2009) focuses on a solution to a very specific case of the two-crane scheduling problem. The minimization of the time used when a containers are transported from one side of the lane to the other side using two non-passing ASC. Ng (2005) uses dynamic programming-based heuristic to find an effective schedule. It involves multiple cranes but is applied to a Asian type container terminal. In an Asian type container terminal, the whole stack is divided into multiple blocks between which vehicles can drive through to pickup containers.

3. Stacking strategies and ASC scheduling strategy

As mentioned before the way containers are placed a stack can have big impact on the retrieval time of said containers. One of the important factors to look for is minimizing the amount of reshuffles. According to Kim and Hong (2006) there are three phases where we can minimize the amount of reshuffles. First the storage location of incoming blocks must be efficiently determined. Second, the relocation of containers must also be efficiently determined and third is the pickup sequence must be determined in a way that the amount of relocations is minimized.

The stacking strategies used by Dekker et al(2006) and Borgman et al(2010) are of the first category. They attempt to reduce the amount of reshuffles by optimally placing the incoming containers. In this thesis I will use the random stacking and leveling stacking strategy in combination with the ASC scheduling lock strategy.

3.1 Stacking strategies

3.1.1 Random Stacking(RS)

The random stacking strategy is the most basic stacking strategy. Generally used as a benchmark algorithm. Each time a container arrives, it randomly selects a pile not at the maximum tier to place the container. When a reshuffle has to take place, the place to stack the to be reshuffled container is also determined by the same method.

3.1.2 Leveling (LEV)

Leveling is a stacking algorithm that seeks to minimize the amount of reshuffling. It tries to accomplish this by stacking in levels. First all the empty ground places are filled. Then the second layer is build on top and so on. The layers are build from the seaside to the landside on. The reason for this is because most containers arrives from and leaves on the seaside. The thought behind this algorithm is that reshuffling is very costly compared to travel time. By keeping the average tier of the piles low, this algorithm hopes to reduce the amount of reshuffles.

3.1.3 Random Stacking with Departure Times (RSDT)

The previously mentioned algorithms don't make use of the available data about the containers. In some cases the estimated time for a container to depart is known. Using this data one can stack the containers better. An simple addition to a basic algorithm could improve the performance of an algorithm. By using departure times, containers can be stacked more optimally. A variant of the random stacking algorithm will be also used in the simulation. The random stacking departure times (RSDT) is basically the same as the random stacking, but with a preference of stacking on stackable piles with a top container with a departure time later than the container to be stacked. This will reduce the chances of reshuffles. If no such pile is found, the algorithm will behave the same as the parent random stacking algorithm. Even though departure dates are never precise, for this simulation I will use the actual departure times of the containers.

3.1.4 Leveling with Departure Times (LDT)

Although there is some thought in the leveling algorithm, it still doesn't use make good use of the available information. It treats every container the same. By using the departure times of the

container the amount of reshuffles can be reduced. Further increasing the performance of the algorithm. The algorithm will first search for piles whose top container departure time is later than the new container's departure time. Of these piles, the algorithm chooses the one with the smallest difference with the new container's departure time. If there are multiple candidates, the algorithm chooses the one closest to the side where the container will exit.

If there are no piles whose top container's departure time is later, the algorithm chooses an empty pile closes to the side where the container eventually will leave the lane. Like the parent algorithm this is an attempt to minimize reshuffling. When there are no more empty piles left in the lane, the highest stackable pile nearest to the container's exit side will be chosen. This is to minimize the reshuffling.

3.2 ASC scheduling strategy

With more than one ASC in one lane of the non-passing kind, a strategy is needed to ensure that the ASCs don't collide during operation.

3.2.1 Segmentation strategy

One way to accomplish this, is to divide the lane into 3 segments: landside, middle and seaside. Each ASC has only access to its own segment and the middle segment, if necessary. The middle segment also would serve as a inter-segment transfer point of containers for both of the ASCs. This will guaranty no collision will occur. This will ensure that the performance is raised by presence of an extra ASC, without too much extra cost in scheduling. The disadvantage of such a strategy is that the segments might not be divided appropriately. The problem arises when a segments is reaching its capacity, while the other segment is not. This could lead to underutilization of the lane, which is very costly for a terminal. Even when the segments are divided appropriately, it might not be able to cope with a temporary rise in amount of arrivals.

3.2.2 Locking strategy

Because of this disadvantage, I have chosen an ASC scheduling strategy that doesn't make use of segments. Every ASC is allowed to stack containers anywhere it is capable to go. To avoid collision, the strategy uses a lock mechanism. When an ASC A have to place a container at bay z, it will lock the whole lane segment till that bay z. ASC B is not allowed to stack anything beyond bay z until the ASC A has released its lock. ASC B will wait patiently and regularly check the lock whether it has been repositioned in a way ASC B can resume performing its task. The wait time is determined by the estimated time ASC A will need to complete its task and potentially shift the lock to a more favorable position. The chances that an ASC will have to wait long is pretty low. Every job of the ASCs involves going to the transfer point of at some point. The lock will be release as soon as ASC A heads towards the transfer point. When an ASC is going idle, it will return to its transfer point to dwell. This will ensure the whole lane will be accessible for the other ASC. This strategy is very flexible and can adapt quickly to changing circumstances, like higher amount of land containers than usual. It can stack anywhere in the whole lane without worrying about segmented capacity problems.

3.2.3 Waiting Deadlock

While it flexible in its operation, there are some unique problems that can occur in this strategy. Because of its flexibility, a deadlock situation can occur. In this situation both ASC will be waiting for the other ASC to release its lock, in order to complete its own job.

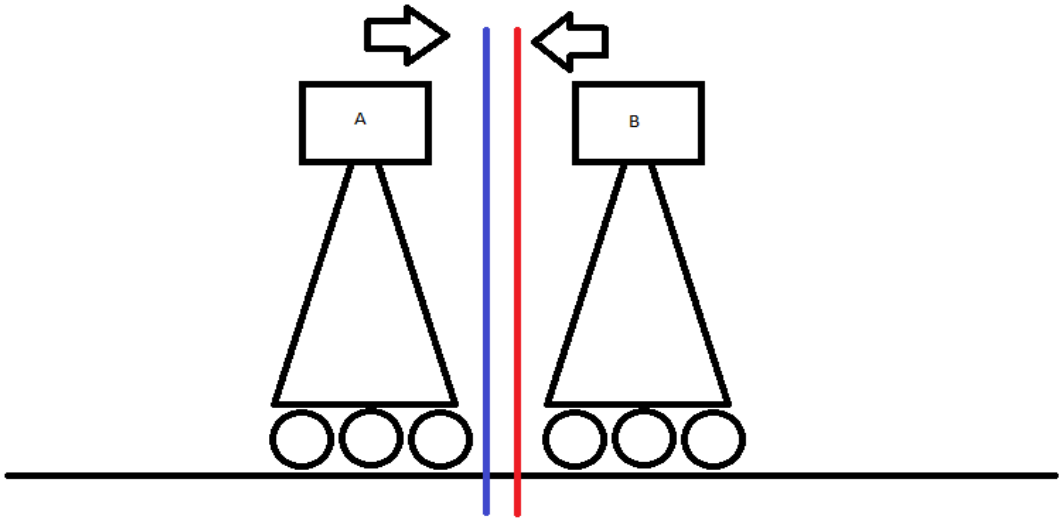


Fig.2 Deadlock situation with ASC A and B

In figure 2 the ASC A has locked the lane at the blue line. And ASC B has locked the lane at the red line. ASC A want to go to bay z for its job. The problem is that it lies within the segment blocked off by B. ASC B has the same situation. It can't complete its job because ASC A has locked its segment. The solution I have implemented for this situation is to have one of the ASC retreat to a position that will allow the other ASC to accomplish its job. This in turn will lead the release of the lock imposed by the other ASC when it has completed its job. The first ASC will be able to completes its job also after that. The first one to wait will get the priority over the second one. Thus the second one will retreat to a favorable position for the first one.

3.2.4 Retrieval Deadlock

To ensure that the other ASC will not interfere with the job of an ASC, the target pile of an ASC is locked. When a pile is locked no other ASC can stack or retrieve containers from it. This is to ensure that the conditions of the pile of containers don't change while travelling there. This could create a conflict however. This happens when a reshuffle occasion is happening on a pile by ASC A, while ASC B wants to retrieve a container in that same pile. The reshuffle operation of ASC A could lead to a repositioning of the target container. There is no guaranty that the container is still in the pile, while ASC A has moved away from the pile doing a reshuffling move. To ensure no such conflict will occur, the ASC, whose target container is in the pile that is being reshuffled, will wait till the pile is unlocked when the reshuffle operation has been completed. The ASC will wait at its own transfer point to make sure the reshuffle operation will proceed smoothly. The ASC will then attempt to located the target container again and retrieve it. Although this might seem very inefficient, it is best to avoid collisions or wasted movement.

3.3.5 No co-operation between ASCs

In this strategy there is no co-operation with the accomplishment of jobs. This might be something to implement in the future for more efficient use of the twin ASC. The current strategy is that each ASC handles only the containers from and destined to its transfer point. The reshuffles will also be done by the concerning ASC alone.

4. Data Generator

To be able to run a semi-realistic simulation, a good data set is needed. In this thesis I have used the data available in Voogd et al (1999. The paper contains a modal split for a realistic model and the amount of containers that comes with it.

4.1 Modes of transport

In a container terminal there are lots of different kinds of transport modes. On the landside there are the trucks and rails. On the seaside there are the barges, feeders/short sea ships, deep sea ships and the jumbo ships. In this simulation however, I will only simulate 3 modes of transport. On the landside the trucks and on the seaside the deep sea ships and jumbo ships.

4.2 Duration

The duration of the simulation will be 15 weeks, including 3 weeks warm-up. The first 3 weeks are to fill an empty lane. The data generated will therefore also be 15 weeks' worth.

4.3 Amount of containers

In Voogd et al(1999) a total of 186000 containers were generated for a 3-week period for a whole terminal. Since I only simulate one lane of a terminal the amount of containers generated must reflect this also. Assuming that the containers will have a average dwell time of 4.6 days with standard deviation of 1.8 days and the stack has an average stack utilization of 67%. (Van Asperen et al(2013). The total amount of containers for the duration of 15-weeks is calculated as follows:

$$\frac{\text{A} * \text{average residence time containers in days}}{\text{max capacity stack} * \text{days}} = \text{stack utilization}$$

with A = the amount of containers to be generated.

Using that formula I will get a total of 11256 containers total≈ 750 containers per week. The actual amount produced by the generator will be 832 containers per week and therefore 12480 containers for the whole period. This is because of the relative low amount of containers and rounding up some numbers led to this new actual number.
Max containers in stack = 41 TEU * 3 Tiers * 6 lanes -2 containers = 736 containers. The minus 2 is to prevent deadlocks in the stack. With a stack with a capacity of 736 and a stack utilization of 67% I will calculate the amount containers to be generated for a 15 week period using the following formula:

4.4 Container format

Each generated container will have the following data

- Container ID (nr.)
- arrival mode (trucks, jumbo, deep sea)
- departure side (land or sea)
- arrival time
- residential time of the container

Containers will have a average residential time of 4.6 days with standard deviation of 1.8 days and the average stack utilization will be 67%. (Van Asperen et al, 2013).

4.5 Modal split arrival and destination of containers

4.5.1 Arrival modal split

The split in Voogd(1999) for arriving containers is 86.9 % (161630)-13.1% (24370) in favor for the seaside. According to Van Asperen et al(2013) this modal split has changed. Trucks process a higher percentage of containers now. To account for the higher trucks arrivals I have chosen a 70% - 30% split for the seaside. This means about 30% of the generated containers will be coming from the trucks . Because of the relative low number of containers, the actual split will be 73%-27% in favor of the seaside. This means of the total of 832 containers per week, will be split in 220 containers arriving from the trucks and 612 containers will be arriving from the seaside.

4.5.2 Destination modal split

All the containers arrived on trucks will leave the stack through the seaside. This is not the case with the containers arrived on the seaside. Of these containers a modal split of 80%-20% in favor for the seaside has been chosen. The reason is that a lot more containers leaves through the seaside than landside. In the generator this splits holds. This means of all the containers arrived from the seaside 80% will leave the stack again through the seaside.

4.6 Schedules and delivery amounts

For the simulation a schedule of arrivals of the different transport modes is required. The arrival scheduled used in this simulation is based on the schedules used in Voogd(1999).

4.6.1 Jumbo and deep sea ship schedule and delivery size

On the seaside each transport mode will have their own schedule. The Jumbo ship will have an schedule that consist 5 arrivals a week equally distributed throughout the week. The arrival times are: Mon 00:00 Tue 09:36 Wed 19:12 Fri 04:48 and Sunday 07:12.

Deep sea ship will arrive 7 times a week equally distributed throughout the week. The arrival times are: Mon 00:00 Tue 01:12 Wed 02:24 Thu 03:36 Fri 04:48 Sat 06:00 and Sun 07:12.

I assume that a jumbo ship will deliver twice as much containers compared to deep sea. Therefore the amount of containers per deep sea shipment will be:

(832 containers /week*73%)/(2*5 + 7)= 36 containers per arrival.

The jumbo will thus deliver 72 containers per arrival. This of course doesn't represent the real capacity of a jumbo ship. I assume the 72 containers is just a part of the total load that is destined for the simulated stack in question.

4.6.2. Truck schedule and delivery size

The arrival of trucks during the week is a bit more difficult to make, since I don't have access to the real data. I have chosen for a simplified approximation of the data.

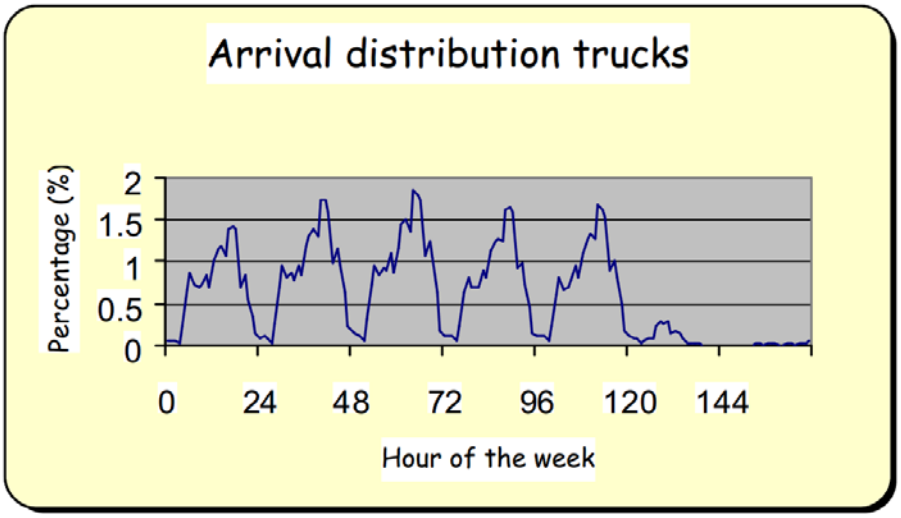


Fig 3. Arrival distribution trucks (source:Voogd et al 1999)

Using the truck arrival distribution graph of Voogd(1999), see figure 3. I have made the following daily distribution.

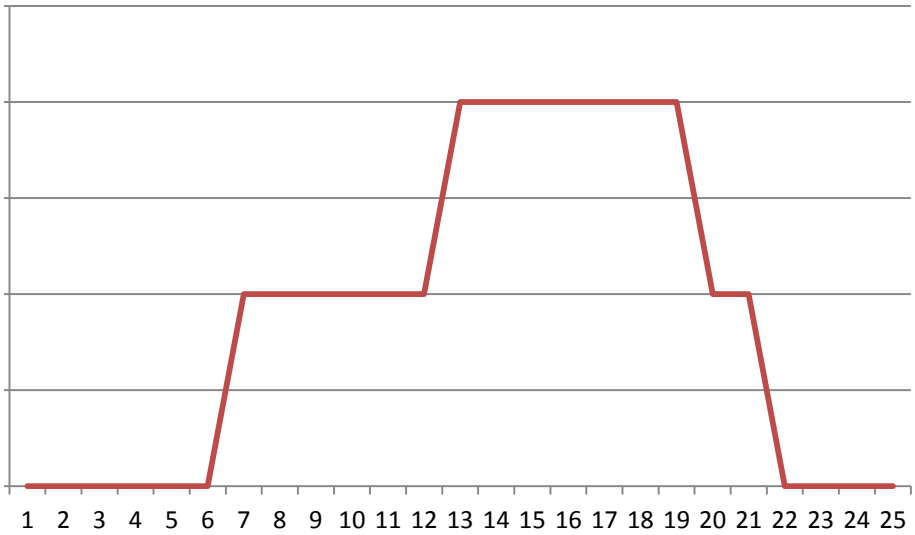


Fig. 4 Simplified daily truck arrival (hours)

The daily schedule is build using the following data:

- 0000-0600 no arrival
- 0600-1200 medium arrival rate
- 1200-1900 high arrival rate
- 1900-2100 medium arrival rate
- 2100-2400 no arrival

In summary there are 8 hour of medium arrival rate, 7 hours of high arrival rate and 9 hours of no arrivals per day. Assuming there will be no truck arrivals at all during the weekends. 40 hours of medium and 35 hours of high arrival rate per week.

Assuming twice as many trucks during high arrival rate compared to medium, a total of 220 containers per week through trucks and that each truck only delivers one container, the medium rate of arrival will be 2 trucks per hour. The high rate of arrival will thus in turn be 4 trucks per hour.

4.6.3 Random Element to the schedules

Since arrivals are never that precise, I have add an random time element to each of the arrival times.

For jumbo and deep sea ships, the arrival time will be "disturbed" with a random drawing from a uniform probability distribution with an interval from -12 hours to +12 hours. The arrival times of the trucks will also be disturbed by a random drawing from a uniform probability distribution, but the interval will be -6 minutes (-0.1 hour) to +6 minutes(+0.1 hour).

The time that a container will stay in the stack after arrival, called residence time, will be determined by a random drawing from a triangular probability distribution with a mean of 4.6 days (110.4 hours) and a standard deviation of 1.8 days(43.2 hours). The lower limit will be 0 days (0 hours) and the upper limit 9.2 days(220.8 hours).

4.7 Random Generator and Java package SSJ: Stochastic Simulation

The generator program has been programmed in the Java programming language. To aid me in developing the application, I have made use of the SSJ Stochastic simulation java package . (available at: http://www.iro.umontreal.ca/~simardr/ssj/indexe.html) .Of this package I used the Uniform Probability Distribution, the Triangular Probability Distribution, the random generator using MRG32k3a and the statistics classes of the package to build the generation application.

4.8 Data generated and input file

Using the generator application, 20 generated arrival list have been created. This will be used in the simulator described in the next chapter.

Using the generator I create the following type of files:

- *Containergen<generationnumber>.txt* : this file contains the arrival schedules of all the transport modes with the containers sorted by arrival time, written in csv format. Of each container the following data is created:
    - Container ID
    - Arrival mode: jumbo ship, deep sea ship or truck (resp. 0,1 and 2)
    - Arrival time: in hours from the start
    - Departure mode: landside or seaside (0 and 1 resp.)
    - Residential time: the amount of time the container will reside in the terminal after arrival (using this you can calculate the departure time.)
- *Datagen<generationnumber>.txt*: this text contains all the generated lists used to create the final arrival list. It contains the unaltered default arrival times of the transport modes. The random time change list (disturbance) used on the default list, the sum of both lists and the residence list of the containers.
- *Statgen<generationnumber>.txt:* contains the used parameters for the generation and summarizes the important information about the generation, like amount of containers generated, how much of it is from trucks , the interval used for the uniform probability distribution. It also contains a histogram and a short report with statistics of each list. It gives the average, standard deviation, number of observations and the min and max value.
- *parametersGenerator.txt:* this is the input file used for the generator.

4.9 Verification

To ensure the data generated is according to specification, I used some statistical analysis on the data. (See 6.5 for a more detailed explanation)

4.9.1 Random drawing

The disturbance list for the trucks, deep sea ships and jumbo ship were generated using an uniform probability distribution, which means the data generated should display the same properties. I divided the range of the these list into several intervals of equal size (20 for jumbo and deep sea, 50 for trucks) and made a histogram. The resulting histogram was shaped as a uniform probability distribution. This means that all the generated numbers are close to equally distributed throughout the range.

The disturbance list of the residence time of the containers was generated using a triangular probability distribution. As with the trucks, I have divided the range into 50 intervals and made a histogram. It had a shape close to a triangle. It has a mean 4.6 days and a standard deviation of 1.89 days, which is about the same as what Van Asperen et al (2013) used for their simulation.

In table 1 is shown the statistical numbers of 10 of the 20 generated arrival list. As you can see in the tables, the standard deviation doesn't differ too much each generation. The only one that could be considered significant is the jumbo distribution. It differs the most each generation most likely because of the low amount of observations. (see table 2) The other lists can be viewed in appendix A.

Truck Disturbance Distribution				
num. obs.	Min. (hours)	Max. (hours)	Average (hours)	standard dev.(hours)
3300	-0,1	0,1	2,30E-04	0,058
3300	-0,1	0,1	-7,70E-04	0,058
3300	-0,1	0,1	3,60E-04	0,059
3300	-0,1	0,1	3,70E-04	0,057
3300	-0,1	0,1	-4,70E-04	0,058
3300	-0,1	0,1	7,40E-04	0,057
3300	-0,1	0,1	1,20E-03	0,058
3300	-0,1	0,1	4,30E-04	0,058
3300	-0,1	0,1	1,70E-03	0,058
3300	-0,1	0,1	-1,40E-03	0,058

Table 1. Truck disturbance distribution (interval -/+0.1hours with 0 hours as mean)

Jumbo Disturbance Distribution				
num. obs.	Min. (hours)	Max. (hours)	Average (hours)	standard dev. (hours)
75	-11,952	11,466	0,246	7,006
75	-11,827	11,629	0,059	7,818
75	-10,9	11,908	1,187	7,311
75	-11,907	11,934	0,518	6,506
75	-11,862	11,527	1,12	6,328
75	-11,879	11,786	-0,791	6,946
75	-11,968	11,818	-0,969	8,058
75	-11,988	10,95	0,335	6,532
75	-11,763	11,926	-0,554	7,417
75	-11,916	11,937	0,495	7,228

Table 2. Jumbo disturbance distribution (interval -/+ 12 mean 0)

The generated schedules without disturbance are exactly like programmed it. Combining it with the random time deviation list (disturbance) has posed no problems.

4.9.2 Modal split

From/To transport side	Land	Seaside
Jumbo	1038	4312
Deep Sea	748	3032
Truck		3300

Table 3. modal split of outgoing containers

I also checked the modal split of the outgoing containers.(see table 3.) The outgoing modal split is exactly like configured. I have set the split on 80-20 in favor of seaside on the containers arriving from the seaside. Of containers that came from the seaside (4312+3032) / (1038+748+4312+3032) * 100 = 80% has been assigned to the seaside. While the rest has been assigned to the landside.

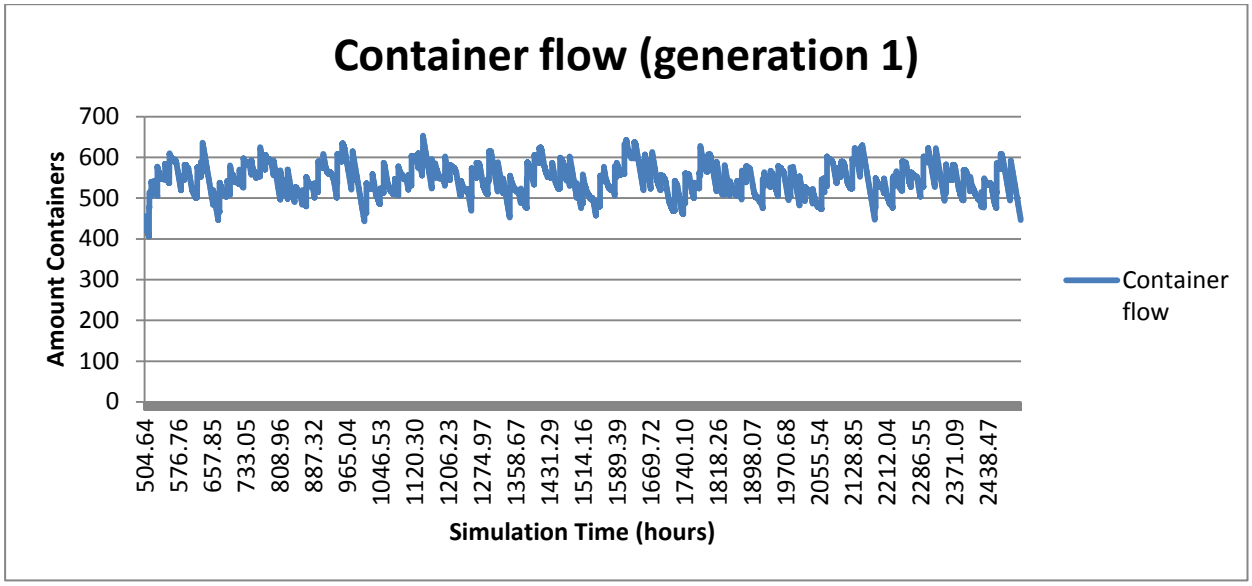
As mentioned before the modal split of the incoming containers is actually 73%-27% in favor of the seaside, because of the way the schedules has been setup ,rounding up of numbers and the relative low amount of containers generated.

Having these observations about the generated data, I feel confident in using them for my simulation.

4.10 Data Analysis

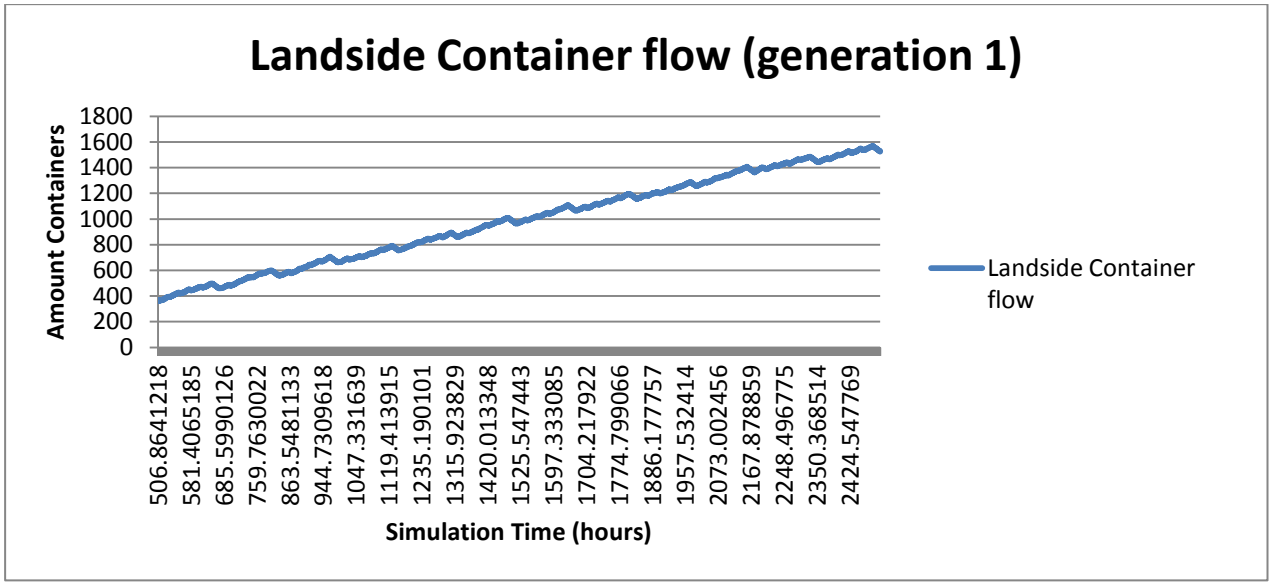
To get an idea how the containers flow through the stack, I have generated three graphs. The first one shows you the amount of containers in the stack over the time. The highest amount of

containers in a stack is 653. It means that the current capacity of 736 should be enough to process the containers. There is also lot



Graph 1. Amount of containers in stack over time.

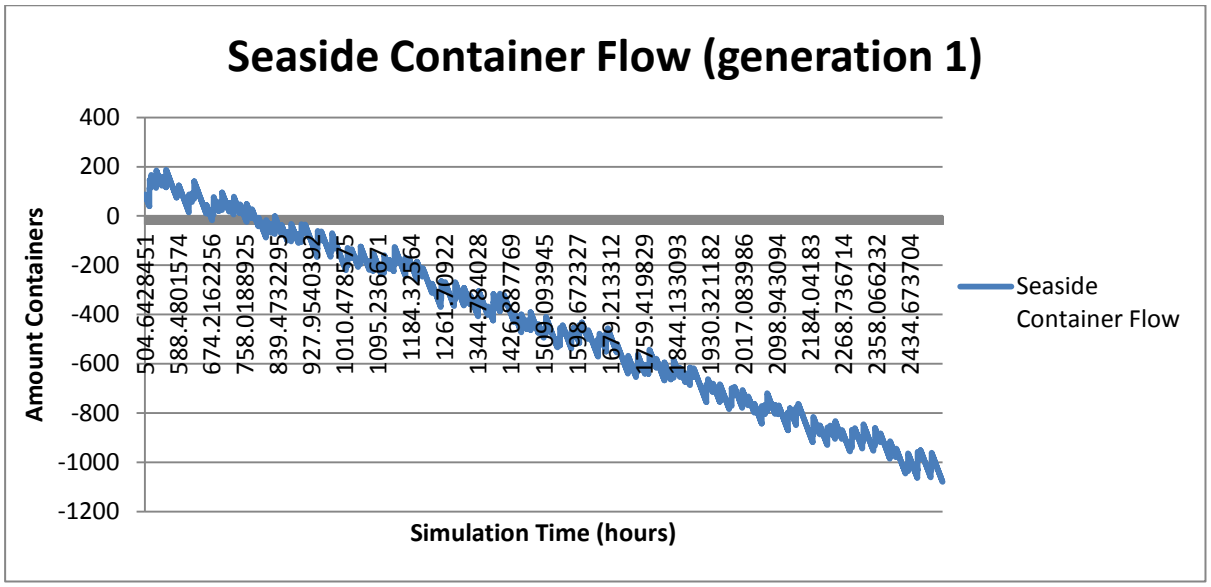
The next graph show the in and outgoing containers of the landside of the stack. If containers come in it goes up and when containers go out through the landside it goes down.



Graph 2. Landside Containers In- and outflow.

It seems that one the landside not many containers will leave the stack. Which is not strange. All the containers from the trucks are destined for the seaside. The only containers destined for the seaside

are a part of the containers arriving via the ships. An interesting pattern seems to emerge. The outflow of containers on the landside seems to have a relation with the arrival of the ships. Most likely because the containers destined for the landside comes in big batches through the ships.



Graph 3. Seaside Containers In- and outflow

The next graph shows the in- and outflow of the sea containers. The line goes below zero, which means there are more containers leaving the stack via the seaside than coming in. The negative net-flow of the seaside is compensated by the positive net-flow on the landside.

5. Simulation

There are different ways to simulate a stack with two non-passing ASCs. Using a simulation model that simulate every passing minute/second. Or use a discrete event model. This model only simulate only the events where something noteworthy is happening. Severely decreasing the computation cost.

I have chosen to use a discrete event model for my simulation. This application is build in the java programming language and I have used the SSJ: Stochastic Simulation in Java package(available at: http://www.iro.umontreal.ca/~simardr/ssj/indexe.html) , as mentioned before, to help me with the development of the application. The SSJ: Stochastic Simulation package contains a class Sim that forms the framework upon which the model is build. It will process events according to their scheduled time. I used the simevents from the package and the stats classes.

5.1 The Discrete Event Model

As explained before the discrete event simulation only simulates noteworthy events at specified times. This reduced the computation load severely, because it only simulate the relevant events. This accomplished by having a timeline. On this timeline events are scheduled to occur. The simulator starts at time = 0 and end at a pre-determined time. The simulator goes from one event to the other according to the scheduled timeline. Each event can generate more events that are in turn scheduled on the timeline for them to be activated when the simulator has reached the specified time. Because of the way it is build, it can simulate all the relevant events without simulating all the time between it. Chapter 6 will give a brief overview of the implementation of this model.

5.2 Simulation parameters

To build this simulation program, I based my program on a simplified design of the one described in Borgman(2009). The simulation will use the imperial feet as the standard unit, because the containers are also measured in feet. An hour has been chosen for one standard time unit.

The total time the simulation will run is a total of 15 weeks(2520 hours), that is including 3 weeks (504 hours) of running to fill the stack. This means I will only gather data of 12 weeks of simulation.

As I mentioned before I will be using a simplified model of the stack. On the seaside only two modes of transport are simulated: Jumbo ships and deep sea ships. And for the landside only the trucks. Every other mode of transportation will be left out.

The simulation of the interaction of quay cranes and AGV between the ships and the stack has also been left out. In this simulation, I want focuses on simulating the stack with the two ASCs. The consequence of this, is that when a ship arrives the whole load will be offered to the stack to be processed at once.

At the transfer points will be no delays due to shortage of AGVs or straddle carriers. It is assumed there is enough of both types to prevent these delays from happening. The process time required to (un)load a container at the transfer points is constant. Which are 0.02 hours (72 sec.) on the seaside and 0.03 hours (108 sec.) on the landside.

5.3 The stack (lane)

In this simulation I will only use one lane of the multiple lanes in an automated container terminal. The dimensions of the stack are 41 TEU long, 6 rows wide and have a maximum of 3 containers height (3 tiers) (Borgman, 2009). Each ground position where containers can be stacked will be called a pile. There are a total of 41 x 6 = 246 piles in the lane.

There will be situations where the ASC need to retrieve the bottom container of a 3 tier high pile. In that case the ASC will need to relocate the unneeded containers to other positions in the lane. This is called a reshuffle of a container. A reshuffle occasion is the event where a reshuffle is needed. This could mean multiple reshuffles. To ensure that there is always the possibility to reach the lowest container with reshuffling, a minimum of the maximum tier minus one positions should be open. If that is not enforced, a deadlock situation might arise. In this situation the ASC can't reach the target container because there are no positions in the lane to place the unwanted containers. For that reason the maximum capacity of a stack the total amount of containers possible in a stack minus the one less of the maximum tier. In this case the maximum containers in a stack is equal to 41 *6 *3 - (3-1)= 736 containers.

If the stack is at capacity every incoming container will be rejected. I will assume that the rejected containers are being moved to other lanes of the terminal for processing. But as seen in the previous chapter the maximum would likely be below 660.

In this simulation the 20 feet container is the only one type of containers that will be used. The dimensions of a 20 feet container are 20 feet in length x 8 feet in width x 8.5 feet in height. Every container will occupy exactly 1TEU.

As mentioned before it is assumed that AGV and straddle carriers will not cause a delay for the operation of the stack and its ASCs.

5.4 ASC movement

5.4.1 ASC data

To somewhat accurately simulate the movement of the ASC I will use the following data from Borgman (2009)Note that the figures are expressed in hours because the standard unit used in this simulation are hours. The unit has been chosen, because of the 15 week period of simulation and the relative infrequency the ships will arrive at the terminal.

ASC lengthwise speed : 47244.10 ft/hour (14400 m/ hour =4 m/s)

ASC lengthwise acceleration :12755905.51 ft/hour^2 (388800 m/hour^2 =0.3 m/s^2)

ASC widthwise speed : 9448.82 ft/hour (2880 m/hour = 0.8 m/s)

ASC widthwise acceleration :12755905.51 ft/hour^2

ASC lifting speed (empty) :11811.025 ft/hour (3600 m/hour =1.0 m/s)

ASC lifting acceleration(empty):12755905.51 ft/hour^2

ASC lifting speed (full) :7086.615 ft/hour (2160 m/hour = 0.6 m/s)

ASC lifting acceleration (full) :12755905.51 ft/hour^2

ASC lane pickup/set down time :0.001 (3.6 seconds)

The time to process a container, either loading or unloading, at a transfer point has been set to a fixed amount. This value includes the lifting time and the pick/set down time for that transfer point.

For the landside transfer point, I have set it at 0.03 hours(108 sec.). And for the seaside transfer point the process time has been set at 0.02 hours (72 sec.)

5.4.2 Acceleration model

These are the speed numbers I use for the movement and processing time of the ASC in the simulation. To account for the time needed for acceleration I use the following formula to calculate the amount of time needed to travel a distance.

$$(1) \quad t = 2 * \sqrt{\frac{S}{a}}$$

with t = time in seconds, S is the distance in feet to be travelled and a is the acceleration in feet/second. This formula is derived from the standard formula to calculate the distance travelled when a object accelerate.

$$(2) \quad S = v_0 t * \frac{a*t^3}{2}$$

where v_0 is the initial speed (which is 0 in our case). This in combination with the assumption that the time needed to accelerate is the same as the time need de decelerate to travel a distance will result in (1). But when the ASC reached its maximum speed and the distance is bigger than 87.0522 ft, a different formula is used.

$$(3) \quad t = 26.7 + \frac{(D - 87.0522\ ft)}{a}$$

with D is the distance that is bigger that is 87.0522 ft. The same formula's are derived for the trolley of the ASC widthwise and lifting wise. 87.0522 ft. is the distance required to speed up to max speed and decelerate immediately to a stop. 26.7 seconds is the time needed to perform such action. Using this knowledge, it will become easier to calculate the time need to travel for a distance.

The movement speed of the ASC is assumed to be the same whether it is empty or full. There is also no delay to process a job, when a job is received it will move immediately to carry it out.

5.4.3 Safety distance

In some papers a safety distance has been suggested between the ASC to prevent damage when malfunctioning. The safety distance employed in this simulation is exactly 20 ft. The distance between two ASC is always a minimum of a container length.

5.4.4 Job Assignment ASC

When retrieving containers, only the ASC that belong to the departure side will retrieve it. All containers that will depart through the seaside transfer point will only be handled by the seaside ASC. All the containers destined for landside transfer point will only be processed by the landside ASC. Due to long queue there could be a situation where the departure time of a un-stacked

container has past. In the simulation these containers will be stacked and be retrieved to the corresponding transfer point.

5.4.5 Stacking strategies and ASC scheduling strategies

As described in chapter 3, this simulation will stack the incoming containers using the random stacking strategy and the leveling strategy. The ASC scheduling strategy will be implemented as is described in previously mentioned chapter. When the ASCs are idle, they will not perform any re-marshalling of the containers. Although it most likely will improve the efficiency of the stack, it is beyond the scope of this simulation.

5.5 Input data

As input data I use the generated data from the previously mention generator. ContainersGen1.txt contains the arrival data of the containers. This includes the container ID, arrival mode, arrival time, departure mode and the residence time in stack of the container. Every entry will be converted to an event to be entered into the timeline of the discrete event simulation. A retrieve container job event will be created after a new container has been stacked. The time this retrieve container job will be scheduled is equal to the departure time of the container itself. If the departure time is earlier than the current simulation time, the event will be scheduled at the current simulation time.

Although in real life the actual arrival date is not certain, in this simulation I will use the arrival date without adding or subtracting a random time to it. In this simulation I want to use the arrival and real departure time. Any difference in performance of the stacking algorithm can subsequently be attributed to the algorithm itself and its performance will not influenced by a random element.

5.6 Performance measures

To measure the performance of a stacking strategy the following numbers are tracked:

- *Reshuffles(**RDC%**) and reshuffle occasions(**ROC%**)*: Measured as a percentage of the amount of containers leaving the stack. A reshuffle occasions is an event where one or more reshuffles are needed to reach the target container.
- *Exit Time Seaside (**ETS**)* and *Exit Time Landside (**ETL**)*: The time it takes to retrieve a container from the stack to the sea side or land side respectively. The time is calculated as the difference between the time the ASC has arrived at the pile and started the job and the time the container has been delivered to the respective side . A 90-percentile value of both exit times will also be tracked since it will give me a bit more information of the distribution of the exit times.(**90%ETS** and **90% ETL**)
- *Workload of the ASC*: The workload is measured as in the percentage of time the ASC is busy. The variable **ASCSS%** is for the ASC seaside and the variable **ASCLS%** is for the ASC on the landside.
- *Ground piles usage(**GPU%**)*: The amount of non-zero container piles are recorded as a percentage of the total amount of storage piles on a hourly schedule.
- *Total Time Job Assignment(***TTJLS** for jobs on the landside and **TTJSS** for jobs on the seaside*):* Although ETS and ETL are generally used as the main performance measure, it doesn't incorporate the time a job has to wait before actual delivery. This measures the total time it from supposed departure time to the actual simulated departure time. The 90-percentile

values are also kept for these performance numbers(**90%TTJLS** for landside and **90%TTJSS** for the seaside)

5.7 Data generated

5.7.1 Amount of simulations

The simulation has been run 20 times for each stacking algorithm, Random stacking and Leveling. I believe 20 runs for each should be enough to give me reliable results about the performance of the stacking algorithm with the ASC locking strategy. As seen in 5.8.2 the differences between each generation are very small and well within the 95% confidence interval. This is especially the case with the random stacking. The simulation itself is deterministic. It will generate the same results every time if you give it the same input data. The ASCs follow a specific rule set and don't deviate from it. The only stochastic element is build in the arrival list of the containers, generated by the generator program. With the random stacking algorithm however another random element has been added to the equation. This will lead to increased variability in the output with each simulation. As can be seen in the averages in 5.8.2, the random stacking does indeed have bigger standard deviations. The average means of each however stays within the 95% confidence interval and are very close to each other. Because the difference in output is small with each simulation, I believe the 20 runs, I have chosen, will give me a reliable information about the performance of the stacking algorithms and ASC strategy within the specified constraints of the simulation.

5.7.2 Generated data files and input files

Each simulation will generate a certain amount of files that describe the simulation and its result. They are the following:

- *Activitylog<Simulationnumber>.txt:* This file contains the log of every event occurrences happening within the simulation. It contains the time stamp of the event, the type of event and a brief description of what the event entails.(Note: this log has helped me with the debugging process). An example log entry:

0.6811967243424157 Event job new container 20 has been accepted from queue
0.6811967243424157 ASCSS job accepted: stack new container 20
0.6811967243424157 ASCSS moving to Seaside Transfer Point
0.6862053588625867 ASCSS pickup container 20
0.7062053588625867 ASCSS moving to bay 38 row 5
0.7112139933827577 ASCSS stacking container 20 at bay 38 row 5
0.7194528076989335 ASCSS Starting new job

- *SimulationStats<StackingAlgorithmname><Simulationnumber>.txt:* this text file contains a summary of the performance measurements of the simulation. Performance measurements like those described in 5.6. Each simulation will generate such a file.
- *CombinedStats<StackingAlgorithmname>.txt:* this file contains the all important performance measurements of every generations done with a specific stacking algorithm in csv format. This format allows easy use in spreadsheet programs like MS Excel.

- *parametersSimulator.txt:* this is the variables input file. It contains the values for the input variables for the simulator. Every time the program is start with this filename as parameter, the program will read the input variables and use them for the simulation.

5.8 Verification

5.8.1 Software testing

During the development of the application, many parts have been tested individually and combined throughout before incorporating it in the main application. And small test cases were created to test for known possible conflicts and the conflicts that surfaced during development of the application. Like the waiting deadlock (see 3.3.2.1) or the retrieval deadlock(see 3.3.2.2). The basic functions and the strategy used by the ASC were al tested and have passed.(See chapter 6.5) Although I can't guaranty this application is bug-free, I believe it is reliable enough to be used for simulation.

5.8.2 Statistical comparison

Since I don't have any data of other simulations using a twin ASC setup. I have decided to run the simulation for each algorithm twenty times each. Take the average of the first 10 and compare it with the last 10.

ROC%	RDC%	GPU%	ASCSS%	ASCLS%	ETS (h)	StDev ETS(h)	ETL(h)	StDev ETL(h)
72,280	92,092	99,760	53,474	15,957	0,072	0,024	0,084	0,024
71,973	91,994	99,754	53,468	15,924	0,072	0,024	0,083	0,024

90% ETS(h)	90% ETL(h)	TTJSS(h)	StDev TTJSS (h)	TTJLS(h)	StDev TTJLS(h)	90% TTJSS (h)	90% TTJLS (h)
0,102	0,122	0,554	0,908	0,106	0,908	1,952	0,158
0,102	0,122	0,561	0,916	0,107	0,916	1,972	0,160

Table 4. Averages first 10 and the last 10 simulation Leveling

The first two are the averages of the 20 simulations using the Leveling algorithm.(table 4) As you can see the averages are pretty close together even the 90% for the ETS, ETL and the total time Job (TTJ) for the landside and seaside are close together. They are all within the 95 % confidence interval.

ROC%	RDC%	GPU%	ASCSS%	ASCLS%	ETS (h)	StDev ETS(h)	ETL(h)	StDev ETL(h)
68,401	104,975	0,889	56,101	16,003	0,075	0,030	0,088	0,030
68,859	105,433	0,889	56,218	15,974	0,076	0,030	0,088	0,030

90% ETS(h)	90% ETL(h)	TTJSS (h)	StDev TTJSS (h)	TTJLS(h)	StDev TTJLS (h)	90% TTJSS (h)	90% TTJLS (h)
0,115	0,131	0,627	0,995	0,111	0,995	2,188	0,167
0,115	0,131	0,637	1,006	0,111	1,006	2,213	0,167

Table 5. Averages first 10 and the last 10 simulation Random Stacking

ROC%	RDC%	GPU%	ASCSS%	ASCLS%	ETS (h)	StDev ETS(h)	ETL(h)	StDev ETL(h)
37,259	52,369	81,364	48,146	14,264	0,057	0,026	0,068	0,026
36,857	51,630	81,376	48,053	14,206	0,057	0,026	0,068	0,026

90% ETS(h)	90% ETL(h)	TTJSS(h)	StDev TTJSS (h)	TTJLS(h)	StDev TTJLS(h)	90% TTJSS (h)	90% TTJLS (h)
0,099	0,112	0,522	0,924	0,088	0,924	1,893	0,136
0,099	0,111	0,528	0,933	0,088	0,933	1,911	0,136

Table 6. Averages first 10 and the last 10 simulation Random Stacking with Departure Times

The same could be said about the Random stacking without or with departure times (table 5 and 6 respectively)

ROC%	RDC%	GPU%	ASCSS%	ASCLS%	ETS (h)	StDev ETS(h)	ETL(h)	StDev ETL(h)
0,598	0,719	84,761	40,428	13,148	0,040	0,006	0,049	0,006
0,545	0,639	84,659	40,391	13,133	0,040	0,005	0,049	0,005

90% ETS(h)	90% ETL(h)	TTJSS(h)	StDev TTJSS (h)	TTJLS(h)	StDev TTJLS(h)	90% TTJSS (h)	90% TTJLS (h)
0,047	0,057	0,451	0,894	0,064	0,894	1,705	0,084
0,047	0,056	0,454	0,901	0,064	0,901	1,722	0,085

Table 7. Averages first 10 and the last 10 simulation Leveling with Departure Times

The differences in values with the Leveling with departure times are also small, except for the reshuffles(RDC%) and the reshuffle occasions(ROC%). This can be explained by the incredible low amount of reshuffles and reshuffle occasions compared to the amount of outgoing containers. I had to double check whether the results. Borgman et al (2010) also implemented the LDT for its simulation and had the similar results for LDT. It has extreme low amount of reshuffles and reshuffle occasions. Because other people have the same results I believe the results are reliable. The results will be discussed in Chapter 7.

6. The applications

6.1 Programming language

For building a simulation on a computer, there are different kinds of venues a person can turn to. There are programming languages specifically designed for simulations (e.g. Arena, GPSS and Modelica) or we can use general purpose programming languages(e.g. Java, C++ and C#).

The advantage of simulation languages is that most of the concepts of a simulation are already build into the language and has been tested by professionals and experts. And because it is a simulations language, it should be relative simple to build a simulation if you know how to program in the specific language. The advantage of a simulation language is also a disadvantage. A simulation language are generally used for specific kinds of simulations. You can only build simulations that the language supports. It constraints you in your choice of the kind of simulation you can build.

The freedom of general purpose languages can be considered an advantage. A simulation model can be build fully customize to the specifications of the developer. The developer is no longer constraint by the pre-programmed capabilities of a simulation language. Although the disadvantage of such freedom is that it is very difficult to build a good simulation program with a general purpose language. Every component of the simulation needs to be tested to make sure it does what was specified. And even if it pass the test , there is still the chance a bug has not been discovered yet. This is also true for the simulation languages, but in lower amounts, since they constraint you in your choices and therefore limit the amount of errors you can make.

Even though a simulation language would have been the logical choice to make for my simulation, I have chosen to build the simulation in a general purpose language. It is not because I want to build a very specific simulation not supported by the simulation languages. The main reason is that I'm just more comfortable using Java, because it is the only programming language where I'm proficient enough in to build the simulation. I have relative little experience with simulation languages and felt that with Java I can still get the job done. I also have used a Java package to help me to the simulation. The SSJ: Stochastic Simulation in Java package. It contains tools to build a discrete simulation and statistical classes like the uniform probability distribution.

To build this application I used the programming IDE Eclipse with Java 7. It is developed on a computer with a Intel Core2Duo E8600 processor with 7 GB of internal memory.

6.2 Application information

For this simulation, I have build two applications. The generator and the simulator.

The generator was build to produce the arrival schedules of the relevant transport modes and create the containers that arrives with the transport modes. It is about 900 lines of code and includes comments and sometimes duplicate code, like variable declaration for each class. As described in chapter 4, it also generated files with generated data used to create the resulting arrival list. This is for verification purposes and to examine the statistical properties of the generated data. It can generated multiple arrival list in one run, appending each generated file with the number of each run. Each run takes about a couple of seconds to generate an arrival list of 12480 containers.

The simulator is the main application of this thesis. Given an arrival list of containers, it will simulate how the stack will process a incoming container, the movement of the ASC, the decision making of the stacking algorithm and the time it takes to do everything in the process. A 'dirty' count of the amount of lines of code is stuck at about 3500 lines. Although it may seem a lot, this is includes function comments and duplicate code. Without that I believe could would be around 2000 lines. Each simulation takes about 15 seconds.

6.3 Difficulties

As mentioned before using a general purpose language pose the difficulty of building almost everything yourself. But with the help of the SSJ statistics package, part of the foundation of a discrete event simulation has already been made. With this package I was able to focus on building the simulation instead of building the model of a discrete event simulation itself.

This was the first time I have build a discrete event simulation in Java. One of the problems that I have encountered was that I misunderstood what an event means in the context of the simulation. I was under the impression that each activation meant that the event has just ended. The reality is that when a event is triggered by the simulator, it meant that the event still has to happen. This cost me a lot of time to redesign my simulator. Luckily I could salvage most of the code.

Another mistake I made was not testing an event chain during the development. After programming multiple events classes, it became very time consuming to find the error in the code. If I wasn't using an IDE (integrated development environment) , the chances are I wouldn't have found the error.

Designing a discrete event simulator was certainly a learning experience for me. It might sound easy in theory, but the implementation of it, is a bit more complex than I have envisioned. I severely underestimated the time needed to develop it.

Another difficulty that I encountered is that I need to build all the verification tools myself. An example of a simple tool is the class that will convert the time in hours into weeks, days, hours, minutes and seconds. This was make sure the generated time was more easily readable for verification. Of course the tools themselves had to be verified also, but luckily they were small applications where the input and output could be verified quickly.

Although unrelated to the simulation, it is not advisable to program multiple days with little sleep. I binged 3 days programming the simulation with just 9 hours of sleep in total. In the end I had to scrap the application because of a unsolvable bug. Although I could salvage a big part of the application, it was a big waste of time, since my productivity also went down with the lack of sleep. One of the disadvantages of a language with a lot of freedom. When your design is wrong, there are no warnings.

6.4 Implementation

6.4.1 Discrete event simulation

A discrete event simulator consist of a timeline on which events are scheduled. The simulator start at the beginning of the timeline and advanced to the next scheduled event. Everything between the events is skipped since it only simulate the most relevant events, saving a lot of computing cost. If it

is relevant for the simulation there will be an event scheduled for it. When the pre-determined end time has been reached the simulator stops.

6.4.2 General event chain

When the actions of an event are executed, the event ends or it will generate another follow up event that will be scheduled on the simulation timeline. If every follow event create another event, it creates an event chain. The event chain represent a process to be simulated. An example of an event chain is the stacking of a newly arrived container by an ASC. An new container event is triggered, this in turn schedules a job event for the ASC. When the ASC accept the job, it will in turn start the event moving to transfer point. After reaching the transfer point, the event pick up new container is triggered and the ASC will pick up the container. When done with that action, the ASC have to decide where to place the container, triggering a decision event. After getting a location to place container, the movement event is executed. The movement event represent the time needed for an ASC to move to a specific location. Upon reaching the destination pile, the stack container event is activated and the container gets stacked. The ASC will then check whether there are more jobs in its job queue. If there are still jobs, it will start a new event chain and the process continues. If there are no more jobs, the ASC will move to its pre-determined dwell point and go idle. This will break the chain.

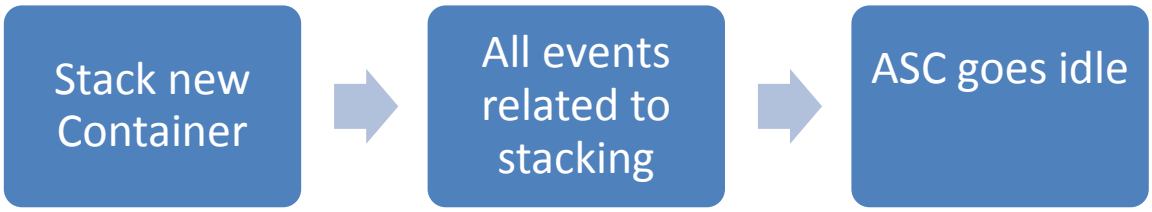


Fig. 5 event chain stack new container

6.4.3 Composition of an event object

The main component in a discrete event simulation is the event. The event must consist of the following things:

- Time scheduled to occur
- Duration of the event
- The actions that occur during the event
- Update the relevant variables
- if applicable schedule a follow event on the timeline

6.4.4 ASC Container event chains

An ASC has basically two event chains. The stack new container and the retrieve container event chain. The stack new container event is scheduled when one of the transport modes arrives and delivers the containers.

The retrieve container event is scheduled when the container has been stacked in the stack. The retrieve container is scheduled on the departure time of the container. The retrieve container event chain contains a sub event chain. The sub event chain is for when the target container is not accessible because other containers are on top of it. The event chain then goes in to the reshuffle sub event chain until the target container can be retrieved.

6.4.5 Twin ASC event chain modification

The previously described two events chain are the basic event chain for an ASC in a lane. There are a couple of additions to the event chains in this simulation, because there are there is another ASC in the same lane. The implementation of the locking strategy result the addition of two new events. The wait event and the retreat event. As described in chapter 3.3.2 the ASC will activate the wait event until the other ASC has release the lock, making it possible for the ASC to complete its job. The retreat event, as described in 3.3.3, will result in an ASC retreating to a position favorable for other ASC to complete its job. In figure 6 the complete and both event chains are displayed.

6.5 Verification

The verification of a model is according to Schlesinger(1979)*:" ensuring that the computer program of the computerized model and its implementation are correct*". What this means is that we need to ensure that the application build does what it is supposed to do. Does it do what it is specified to do? It is also important, especially with models with relations to the real world like simulations, that the application is complex enough to generate information useful for the real world.

A simulation is generally a simplification of the real world. It is simplified in such way that only the relevant operations are simulated. Depending on the scope and the question to be answered, different levels of simplifications are acceptable. in my case my concern is the performance of the stacking algorithms and the simulated process of the ASC scheduling locking mechanism. A good simulated representation of the ASC and the stack is important. For this reason the speed , acceleration of an ASC and the dimensions of a stack are accounted for in the simulation. Although not relevant processes like maintenance and possibility of malfunctioning ASC are left out of the simulation, since it is not directly relevant for the simulation and will make the simulation more complex than necessary. Everything outside the stack has been simulated with a simple model, because they are not the focus of the simulation. Like fixed schedules for transport modes seeded with random times to somewhat approach the uncertainty of arrival time.

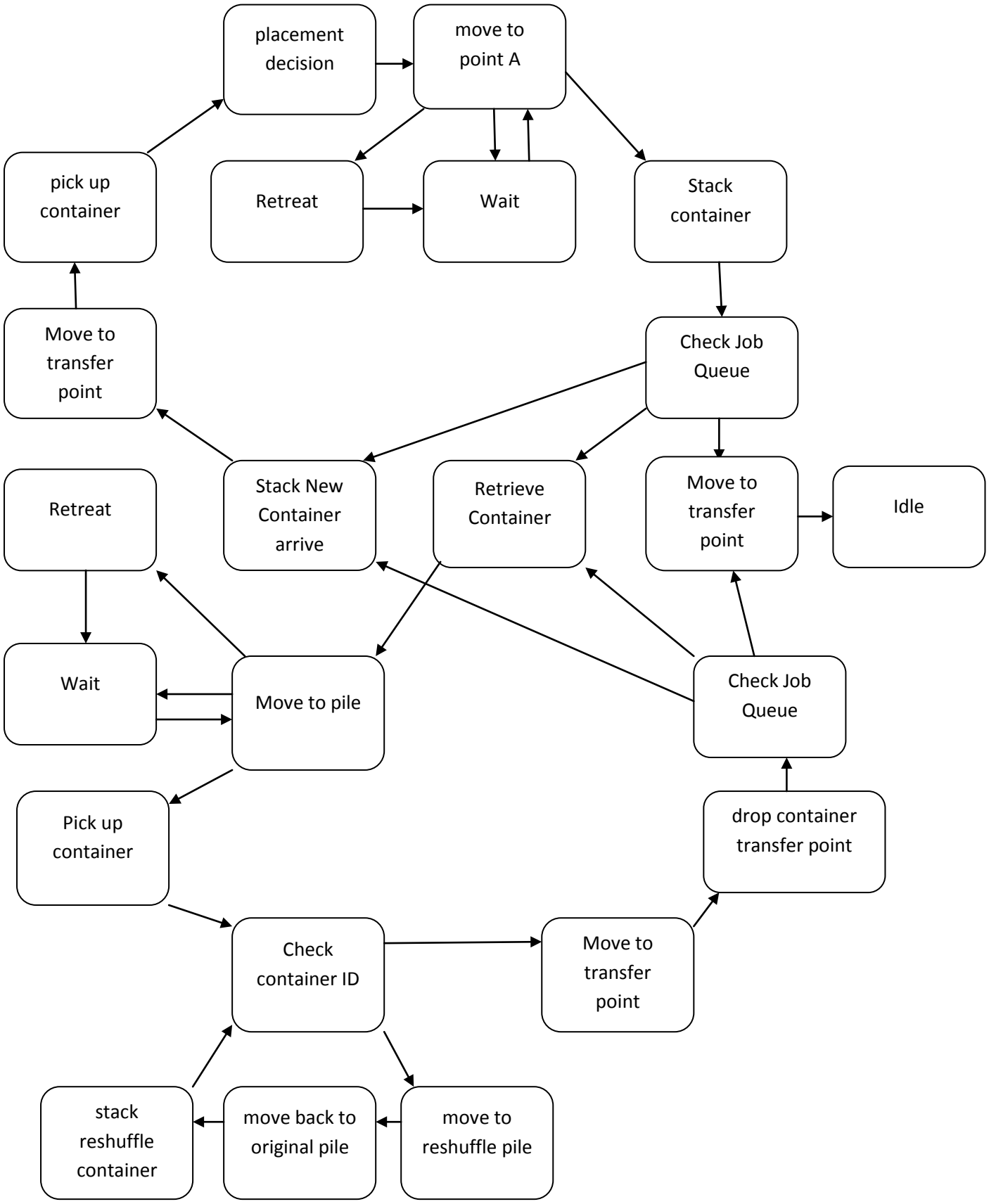


Fig. 6 The simulation event chains

6.5.1 Verification of the developed application

There are 6 techniques to verify a model according to Whitner and Balci (1989):

- Informal analysis
- Static analysis
- Dynamic analysis
- Symbolic analysis
- Constraint analysis
- Formal analysis

6.5.1.1 Informal analysis

Informal analysis refer to checking the code during development. This is generally done to ensure that each part of the code functions as specified before committing it to the main application. Small test cases with input are build and checked the output are checked to verify the code does what it is supposed to do.

During the development of this simulation, this has been done a lot and is a good way to develop a simulation or an application in general. If you postpone the testing of each unit, it will become very complex and time consuming to find the culprit.

6.5.1.2 Static analysis

This refers to syntax analysis of the code. Java is a very strict language, the code has to comply to certain formats for it to compile. In combination with the IDE (integrated development Environment) Eclipse, a lot of programming errors have been caught before compilation of the code. The IDE does syntax analysis on the fly and highlight the errors in the code. It will highlight variable not used or used in a way that will cause errors. I can be assured that if I made a programming error, it is most likely a design error and not the fault of the syntax.

6.5.1.3 Dynamic analysis

This kind of analysis requires the developer to execute the application and verify the output. As mentioned in the informal analysis, this has been done a lot. Creating test cases with known outputs to verify part and the whole program.

6.5.1.4 Symbolic analysis

In dynamic analysis the input and output are verified. In symbolic analysis the inner workings of the applications are verified. Given a certain input we need to examine how the simulation will process the input. This has also been done during the development of the simulator. I build the activity log class for this specific task. It records every event and the accompanying relevant data. With it and the known debug method of System.out.println(""); many of the bugs were tracked and fixed. With the activity log it was made easier to understand what was happening in the simulator. I didn't create an activity log for each simulation though. Each activity log is about 16,5 Mb. Creating all the logs would require 660 MB.

6.5.1.5 Constraint analysis

During this analysis the simulator is test whether the assumptions of the theoretical model are correctly reflected in the program. Although I have build in a lot of checks in the application, like checking whether a pile is maximum height or whether a ASC can move to a place running into the other ASC, not every single input variable will be checked by the simulator. The simulator makes the assumption that the user will input the data in the correct format. I always use the data generated with the generator which I have verified are correct and the simulator assume it will received input data from the generator.

6.5.1.6 Formal analysis

This refers to an analysis in which the developer tries to obtain mathematical correctness of the model. I have only done it with parts of the simulation, like the time an ASC needs to travel from one point to another. I also approximated how long it takes to pick up a container and deliver it to the transfer point. These are the only checks I made mathematical wise. This kind of analysis might be better suited for simulations that can be described with mathematical easier.

7. Results

7.1 Expectations

While leveling (LEV) is a pretty basic algorithm, I expect it will perform better than the random stacking (RS) strategy. With leveling there is a logical reason to stacking and it gives the seaside priority, which should give it better performance compared to the random stacking algorithm. The amount of reshuffles should be lower with leveling, because it tries to reduce reshuffles with keeping the average pile tier low. Since random stacking don't have a specific strategy in placing the containers, it should be expected that the workload of both of the ASC should be higher compared to leveling, because of the widespread of the containers. And because of the higher workload, the expected exit times and the total time job completion should be higher with random stacking as well. Ground usage should be higher with leveling, because it actively seeks to fill the empty grounds when possible. The performance of the variants random stacking with departure times (RSDT) and leveling with departure times (LDT) should be better than the parent algorithms. Especially in the case of reshuffles. By making use of the departure time of the containers the chances of reshuffles are lowered. Because of a lower amount of reshuffles, this should lead to better exit times(ETS and ETL) and total time job completion(TTJSS and TTJLS). I believe that the LDT will perform better than the RSDT, mainly because the parent LEV outperformed RS also.

	ROC%	RDC%	GPU%	ASCSS%	ASCLS%	ETS (h)	StDev ETS(h)	ETL(h)	StDev ETL(h)	90% ETS(h)	90% ETL(h)
RS	68,630	105,204	88,905	56,159	15,989	0,075	0,030	0,088	0,030	0,115	0,131
RSDT	37,058	51,999	81,370	48,099	14,235	0,057	0,026	0,068	0,026	0,099	0,111
RSDT/RS %	53,997	49,427	91,524	85,648	89,031	75,621	85,487	77,215	85,487	86,395	85,022
LEV	72,126	92,043	99,757	53,471	15,941	0,072	0,024	0,084	0,024	0,102	0,122
LDT	57,2	67,9	84,710	40,410	13,141	0,040	0,006	0,049	0,006	0,047	0,057
LDT/LEV %	0,792	0,738	84,916	75,573	82,436	55,344	23,315	58,985	23,315	45,629	46,312

	TTJSS(h)	StDev TTJSS (h)	TTJLS(h)	StDev TTJLS(h)	90% TTJSS (h)	90% TTJLS (h)
RS	0,632	1,000	0,111	1,000	2,201	0,167
RSDT	0,525	0,928	0,088	0,928	1,902	0,136
RSDT/RS %	83,086	92,799	78,801	92,799	86,431	81,296
LEV	0,557	0,912	0,107	0,912	1,962	0,159
LDT	0,452	0,898	0,064	0,898	1,713	0,084
LDT/LEV %	81,182	98,395	60,306	98,395	87,316	53,142

Table 8. Averages of all 20 simulations for all algorithms (see 5.6 for explanation abbreviations)

7.2 Comparison RS and LEV

As expected the ground pile usage (GPU%)is higher with Leveling(LEV) compared to Random Stacking(RS).The average workload (56.158+15.989)/(53.471 +15.941) =1.0394 is with the RS about 4% higher. Which is not as high as I expected.

What is interesting to see is that the amount of reshuffle occasions (ROC%) is higher with the LEV, the amount of reshuffles(RDC%) is lower compared to RS. The ratio RDC%/ROC%, which is the average amount of reshuffled containers per reshuffle occasion, is 1.53 with the RS and 1.27 with the

LEV. This means that with the LEV more reshuffle occasions happen, but the average amount of reshuffled containers is lower compared to RS.

The exit times are as expected. With RS the ASCs need about 5% more time to retrieve a container from the lane. If we look at the standard deviation, the spread around the mean is also smaller with the LEV. It performs consistently better when it comes to exit times on both sides. There is even a 12% difference on the 90-percentile value on the exit time Seaside.

When we look at the total times for job (TTJ) completion the differences are even higher. The average TTJ is 8.7% higher with RS. When we look at individual sides, the RS needs 13.4% more time on average on the seaside(TTJSS) compared to LEV. The same can be seen at the 90-percentile value on the seaside.

Even though the leveling algorithm is a very simple stacking algorithm, that doesn't need any heavy computation, it performs better than random stacking in almost every performance measurement.

7.3 Comparison RS with RSDT, LEV with LDT and the RSDT with LDT.

Performance increase can be seen all round when comparing RS with RSDT. The use of departure times has increased the performance of the RS significantly. The percentage of ROC and RDC has dropped with at least with 46 % and an average drop of 13.6 % in the workload of the ASCs. The exit times dropped with 22 % , while the standard deviation also decreases with 15 %. This means the lower exit times are more consistent. Even the 90-percentile value has decreased with at least 13%. The total time for job completion all went down at least 14 %. This means shorter delays for the transport modes. Overall can be said that the impact of incorporating the departure times has increased the performance of the random stacking algorithm significantly.

When we look at the difference between LEV and LDT. You can also see signifcant increase of performance of all the measured performance values. Even the ground usage decreased. with 15%. Very significant decrease of processing times like the exit times and total time for job completion, means higher throughput of a lane. The differences are remarkable high. Reaching as high as 45 % decrease of exit times on the seaside. The seaside also saw a average decrease of the total time of job completion of almost 40 %. These numbers are very significant.

But out of all the performance increases the decrease of the reshuffles and reshuffle occasion is the most unexpected. It had a decrease of almost 99 %, which would leave me to believe something was wrong with my simulation. I run the other numbers again for the other algorithms and they were the same and for this algorithm each simulation gave the same results. I have checked the individual result files and saw as low as 64 reshuffles and 54 reshuffle occasions for 9995 outgoing containers. I even checked the code again to make sure things were not programmed wrong, even though it assed all the test cases. Borgman et al(2010) also used this algorithm and apparently they also had this result. It seems that the incorporation of departure times made it possible for algorithm improve the overall performance significantly.

The LDT algorithm is the best performing algorithm of these simulated algorithms, which was expected.

8. Conclusion

With the growing demands of the world trade, it is essential for the maritime container terminals to increase their productivity. Since expansions of ports is either very expensive or not possible, they have to find other ways to increase their productivity.

A growing trend is the use of multiple ASCs in the same lane. This increases the productivity significantly without the need of big expansions. Multiple setups has been implemented with varying cost. In some cases the terminal can use existing infrastructure, e.g. the twin ASC. With the twin ASC, the second ASC can use the same rails the first one is using. When using twin ASC, a proper ASC scheduling system is needed to maximize the performance increase.

In this thesis I have propose one such ASC scheduling system and use it with two basic algorithm stacking algorithms to show it is a viable strategy. The Leveling strategy has outperformed the random stacking strategy in almost every important area. This shows that a more "complex" strategy could significantly improve the performance of a twin ASC setup. When I simulated the variants of the basic algorithms that uses the real departure times, I saw a sharp increase of performance on both algorithms. This means that using more information has led to better stacking strategies. In Dekker et al(2006) and Borgman et al(2010) there is also a proposed algorithm called TVD-DTC-MC that performed the best out of all the online stacking strategies. Implementing this strategy would be interesting to see how it performs in a two ASCs setup. The TVD-DTC-MD also included the traveling time in its strategy, which led to better performance in a single ASC setup.

Other area's that can be improved upon are ASC scheduling strategies. My current implementation might be viable, but might not be the best one for the used stacking strategy. The previously mentioned segmentation strategy could be useful with some of the other stacking strategies. These are interesting areas to be explored. My current ASC strategy do the jobs in the order they appear. A possible improvement is evaluating multiple jobs and optimize the workload of both ASCs.

9. References

Borgman B (2009) Improving container stacking efficiency using simulation. Master's thesis, Erasmus School of Economics, Erasmus University Rotterdam.http://hdl.handle.net/2105/5040. Economics and Informatics programme

Borgman B, van Asperen E, Dekker R (2010) Online rules for container stacking. OR Spectrum, 32(3):687–716. doi:10.1007/s00291-010-0205-4. (Open Access)

Dekker R, Voogd P, van Asperen E (2006) Advanced methods for container stacking. OR Spectrum 28:563–586

Dorndorf U. and Schneider F.(2012) Scheduling automated triple cross-over stacking cranes in a container yard, OR Spectrum,32(3): 617-632

Iris F. A. Vis, Hector J. Carlo, (2010) Sequencing Two Cooperating Automated Stacking Cranes in a container Terminal. Transportation Science 44(2):169-182.

Kim, K.H. and Hong, G.-P. (2006) A heuristic rule for relocating blocks. Computer & Operations Research 33:040-954

Park, T., Choe, R., Ok, S. M., & Ryu, K. R. (2010). Real-time scheduling for twin RMGs in an automated container yard. OR Spectrum, 32(3), 593–616.

Saanan, Y.A.,(2004) An approach for designing robotized marine container terminals, PhD Thesis, Delf University of technology.

Saanen, Y.A., M. V. Valkengoed. (2005) Comparison of three automated stacking alternatives by means of simulation. M.E. Kuhl, N. Steiger, F.B. Armstrong, J. A Joines, eds. Proc. 2005 winter simulation conf., baltimore, 1567-1576

Saanen, Y. A. (2011) Optimizing automated container terminals to boost productivity.Port Technology International, 51, 55–65

Schlesinger S.,(1979) Terminology for model credibility. Simulation, 32(2):103-104 Cited by Borgman(2009)

Van Asperen, E., Borgman, B., & Dekker, R. (2013) Evaluating impact of truck announcements on container stacking efficiency. Flexible Services and Manufacturing Journal, 25, 543–556

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

Whitner R.B. and Balci.O. (1989) 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 Cited by Borgman(2009)

Zhou w. and Wu X., An efficient optimal solution of a two-crane scheduling problem ,Asia-Pacific Journal of Operational Research, 26 (1) (2009), pp. 31–58

Zyngiridis I (2005) Optimizing container movements using one and two automated stacking cranes. Master's thesis, Naval Postgraduate School, Monterey

Appendix A: The statistics of 10 generations of disturbance list

Jumbo Disturbance Distribution				
num. obs.	min (h)	max (h)	average (h)	standard dev. (h)
75	-11,952	11,466	0,246	7,006
75	-11,827	11,629	0,059	7,818
75	-10,9	11,908	1,187	7,311
75	-11,907	11,934	0,518	6,506
75	-11,862	11,527	1,12	6,328
75	-11,879	11,786	-0,791	6,946
75	-11,968	11,818	-0,969	8,058
75	-11,988	10,95	0,335	6,532
75	-11,763	11,926	-0,554	7,417
75	-11,916	11,937	0,495	7,228

Uniform prob. distribution with -12 and + 12 as interval and a mean of 0

Deep Sea Disturbance Distribution				
num. obs.	min (h)	max (h)	average (h)	standard dev. (h)
105	-11,877	11,993	0,112	6,737
105	-11,521	11,748	-0,649	6,695
105	-11,847	11,915	1,308	6,723
105	-11,966	11,777	0,421	6,419
105	-11,6	11,939	0,536	7,226
105	-11,313	11,849	-0,189	6,796
105	-11,912	11,989	0,476	6,512
105	-11,508	11,937	-1,274	6,727
105	-11,957	11,818	0,053	7,542
105	-11,926	11,968	0,509	7,307

Uniform prob. distribution with -12 and + 12 as interval and a mean of 0

Truck Disturbance Distribution				
num. obs.	min (h)	max (h)	average (h)	standard dev. (h)
3300	-0,1	0,1	2,30E-04	0,058
3300	-0,1	0,1	-7,70E-04	0,058
3300	-0,1	0,1	3,60E-04	0,059
3300	-0,1	0,1	3,70E-04	0,057
3300	-0,1	0,1	-4,70E-04	0,058
3300	-0,1	0,1	7,40E-04	0,057
3300	-0,1	0,1	1,20E-03	0,058
3300	-0,1	0,1	4,30E-04	0,058
3300	-0,1	0,1	1,70E-03	0,058
3300	-0,1	0,1	-1,40E-03	0,058

Uniform prob. distribution with -0.1 and + 0.1 as interval and a mean of 0

Residence Time Disturbance Distribution				
num. obs.	min (h)	max (h)	average (h)	standard dev. (h)
12480	0,74	217,981	110,531	45,277
12480	1,236	218,377	110,489	44,86
12480	0,178	220,328	110,441	45,199
12480	1,474	219,488	110,445	44,733
12480	1,425	219,737	110,16	44,884
12480	1,524	218,497	110,373	45,264
12480	1,804	218,539	110,622	45,376
12480	1,952	220,471	109,856	44,944
12480	1,394	219,911	110,013	45,066
12480	0,614	219,039	110,739	45,18

Triangular prob. distribution with interval of 0 and 220.8 and a mean of 110.4

Appendix B: Averages 20 simulations using Random Stacking

Gen	ROC%	RDC%	GPU%	ASCSS%	ASCLS%	ETS (h)	StDev ETS(h)	ETL(h)	StDev ETL(h)	90% ETS(h)	90% ETL(h)	TTJSS(h)	StDev TTJSS(h)	TTJLS(h)	StDev TTJLS(h)	90% TTJSS (h)	90% TTJLS (h)
1	67,737	104,473	88,664	56,126	16,056	0,075	0,030	0,087	0,030	0,115	0,129	0,630	1,019	0,111	1,019	2,223	0,168
2	68,635	105,656	89,136	56,134	16,145	0,076	0,030	0,088	0,030	0,115	0,131	0,639	1,003	0,110	1,003	2,140	0,165
3	68,524	104,940	88,995	55,858	16,103	0,075	0,030	0,089	0,030	0,115	0,133	0,615	0,994	0,112	0,994	2,130	0,171
4	68,381	104,381	88,917	56,081	15,962	0,075	0,030	0,089	0,030	0,114	0,131	0,645	1,037	0,114	1,037	2,368	0,175
5	68,505	105,316	89,133	56,245	15,771	0,075	0,030	0,088	0,030	0,115	0,130	0,589	0,935	0,111	0,935	2,033	0,165
6	68,313	104,950	88,833	56,271	15,929	0,075	0,031	0,088	0,031	0,116	0,133	0,616	0,963	0,111	0,963	2,111	0,165
7	69,083	106,167	89,031	56,238	16,041	0,076	0,030	0,088	0,030	0,116	0,131	0,633	0,994	0,110	0,994	2,212	0,165
8	67,952	104,405	88,603	55,902	16,083	0,075	0,030	0,088	0,030	0,114	0,132	0,611	0,968	0,111	0,968	2,168	0,169
9	68,676	105,186	88,837	55,913	16,095	0,075	0,030	0,088	0,030	0,115	0,129	0,651	1,029	0,112	1,029	2,299	0,164
10	68,201	104,274	89,285	56,240	15,849	0,075	0,030	0,086	0,030	0,114	0,127	0,639	1,009	0,109	1,009	2,198	0,164
11	69,213	105,635	88,997	56,472	16,180	0,076	0,030	0,088	0,030	0,115	0,132	0,622	0,984	0,113	0,984	2,132	0,172
12	69,450	106,986	88,962	56,467	16,056	0,076	0,030	0,089	0,030	0,115	0,133	0,634	1,005	0,116	1,005	2,302	0,173
13	68,903	105,596	88,771	55,957	15,918	0,076	0,030	0,087	0,030	0,115	0,129	0,667	1,054	0,109	1,054	2,318	0,165
14	68,500	104,557	88,650	55,876	16,149	0,075	0,030	0,088	0,030	0,115	0,133	0,620	0,990	0,113	0,990	2,162	0,171
15	69,065	105,795	89,067	56,359	15,974	0,076	0,030	0,088	0,030	0,114	0,131	0,648	1,030	0,111	1,030	2,344	0,169
16	68,229	104,597	88,498	56,157	15,933	0,075	0,030	0,088	0,030	0,115	0,130	0,629	0,988	0,110	0,988	2,135	0,162
17	69,477	105,984	88,572	56,401	15,890	0,076	0,030	0,088	0,030	0,115	0,134	0,623	0,985	0,112	0,985	2,084	0,170
18	68,676	105,029	89,282	56,126	16,020	0,075	0,030	0,087	0,030	0,115	0,131	0,628	1,010	0,111	1,010	2,162	0,162
19	68,229	104,976	88,834	56,197	15,857	0,075	0,030	0,087	0,030	0,114	0,130	0,661	1,014	0,110	1,014	2,299	0,162
20	68,844	105,169	89,032	56,164	15,757	0,075	0,030	0,087	0,030	0,115	0,131	0,638	0,998	0,110	0,998	2,191	0,165
average	68,630	105,204	88,905	56,159	15,989	0,075	0,030	0,088	0,030	0,115	0,131	0,632	1,000	0,111	1,000	2,201	0,167

see 5.6 for explanation abbreviations

Appendix C: Averages 20 simulations using Leveling

Gen	ROC%	RDC%	GPU%	ASCSS%	ASCLS%	ETS (h)	StDev ETS(h)	ETL(h)	StDev ETL(h)	90% ETS(h)	90% ETL(h)	TTJSS(h)	StDev TTJSS(h)	TTJLS(h)	StDev TTJLS(h)	90% TTJSS (h)	90% TTJLS (h)
1	72,173	91,685	99,75863821	53,450	15,945	0,072	0,024	0,083	0,024	0,102	0,121	0,548	0,915	0,105	0,915	1,920	0,156
2	72,867	93,334	99,76690541	53,616	16,160	0,072	0,024	0,084	0,024	0,102	0,123	0,568	0,914	0,106	0,914	1,941	0,158
3	71,582	91,383	99,74936282	53,072	15,948	0,072	0,024	0,083	0,024	0,102	0,123	0,546	0,912	0,106	0,912	1,912	0,157
4	72,371	92,331	99,76105788	53,597	15,885	0,072	0,024	0,085	0,024	0,102	0,123	0,576	0,955	0,109	0,955	2,102	0,164
5	72,239	91,621	99,76670377	53,744	15,807	0,072	0,024	0,082	0,024	0,102	0,119	0,531	0,868	0,105	0,868	1,879	0,154
6	72,550	92,020	99,75480707	53,607	15,914	0,072	0,023	0,084	0,023	0,102	0,122	0,542	0,881	0,107	0,881	1,924	0,159
7	72,302	92,118	99,76690541	53,528	15,946	0,072	0,024	0,084	0,024	0,102	0,123	0,547	0,888	0,106	0,888	1,905	0,158
8	71,896	91,350	99,75178249	53,158	15,904	0,071	0,023	0,083	0,023	0,101	0,122	0,544	0,889	0,104	0,889	1,972	0,155
9	72,660	92,372	99,7606546	53,315	16,204	0,072	0,024	0,085	0,024	0,102	0,122	0,575	0,942	0,109	0,942	2,041	0,157
10	72,155	92,703	99,75984804	53,651	15,860	0,072	0,024	0,084	0,024	0,102	0,124	0,560	0,917	0,106	0,917	1,926	0,159
11	71,909	91,813	99,74270874	53,729	16,034	0,072	0,024	0,084	0,024	0,102	0,121	0,547	0,898	0,108	0,898	1,924	0,165
12	71,775	92,312	99,74371693	53,664	15,739	0,072	0,024	0,082	0,024	0,103	0,121	0,558	0,924	0,107	0,924	2,025	0,159
13	72,172	92,038	99,75500871	53,416	15,974	0,072	0,024	0,083	0,024	0,103	0,123	0,603	0,975	0,106	0,975	2,097	0,161
14	70,879	90,644	99,74996774	52,917	16,023	0,071	0,024	0,083	0,024	0,102	0,120	0,532	0,882	0,106	0,882	1,898	0,157
15	72,366	92,979	99,75682346	53,704	15,950	0,072	0,024	0,084	0,024	0,103	0,122	0,574	0,936	0,106	0,936	2,075	0,159
16	71,964	91,464	99,75621854	53,314	15,924	0,071	0,024	0,085	0,024	0,101	0,125	0,553	0,902	0,108	0,902	1,891	0,160
17	72,133	91,520	99,75440379	53,510	15,840	0,072	0,024	0,084	0,024	0,102	0,120	0,554	0,907	0,107	0,907	1,893	0,158
18	72,256	93,009	99,76307427	53,471	16,021	0,072	0,024	0,083	0,024	0,103	0,123	0,554	0,923	0,107	0,923	1,950	0,160
19	72,314	92,470	99,76186443	53,524	16,014	0,072	0,024	0,084	0,024	0,102	0,123	0,576	0,919	0,108	0,919	2,023	0,165
20	71,960	91,695	99,75460543	53,433	15,721	0,072	0,024	0,082	0,024	0,102	0,121	0,555	0,900	0,104	0,900	1,944	0,155
average	72,126	92,043	99,757	53,471	15,941	0,072	0,024	0,084	0,024	0,102	0,122	0,557	0,912	0,107	0,912	1,962	0,159

see 5.6 for explanation abbreviations

Appendix D: Averages 20 simulations using Random Stacking Departure Times

Gen	ROC%	RDC%	GPU%	ASCSS%	ASCLS%	ETS (h)	StDev ETS(h)	ETL(h)	StDev ETL(h)	90% ETS(h)	90% ETL(h)	TTJSS(h)	StDev TTJSS(h)	TTJLS(h)	StDev TTJLS(h)	90% TTJSS (h)	90% TTJLS (h)
1	37,1386	52,0460	81,1316	48,2512	14,2915	0,0572	0,0259	0,0674	0,0259	0,0995	0,1122	0,5236	0,9388	0,0865	0,9388	1,8981	0,1344
2	37,6447	52,7046	81,5335	48,1922	14,4103	0,0571	0,0259	0,0684	0,0259	0,0994	0,1123	0,5412	0,9433	0,0874	0,9433	1,8997	0,1336
3	37,7633	53,4203	81,3530	48,0627	14,2988	0,0575	0,0259	0,0680	0,0259	0,0998	0,1117	0,5069	0,9148	0,0878	0,9148	1,8113	0,1351
4	36,8083	51,6840	81,6674	48,1053	14,0557	0,0570	0,0258	0,0675	0,0258	0,0999	0,1126	0,5337	0,9526	0,0876	0,9526	1,9949	0,1380
5	37,4112	52,6679	81,4595	48,3604	14,2193	0,0572	0,0256	0,0682	0,0256	0,0995	0,1121	0,5037	0,8942	0,0880	0,8942	1,8444	0,1371
6	37,1626	52,0296	81,3916	48,1764	14,2151	0,0569	0,0254	0,0678	0,0254	0,0984	0,1098	0,5014	0,8796	0,0876	0,8796	1,8214	0,1372
7	37,8359	53,1689	81,5787	48,1676	14,3773	0,0572	0,0258	0,0700	0,0258	0,0993	0,1150	0,5188	0,9115	0,0894	0,9115	1,8699	0,1413
8	36,9143	51,6119	80,7921	47,9625	14,1648	0,0571	0,0257	0,0664	0,0257	0,0993	0,1079	0,5130	0,9018	0,0852	0,9018	1,8901	0,1328
9	36,0697	51,0562	81,0724	47,6566	14,3822	0,0566	0,0256	0,0680	0,0256	0,0990	0,1114	0,5447	0,9671	0,0886	0,9671	1,9939	0,1338
10	37,8441	53,2980	81,6585	48,5246	14,2228	0,0574	0,0257	0,0686	0,0257	0,0993	0,1119	0,5321	0,9374	0,0887	0,9374	1,9036	0,1365
11	36,6075	51,2764	81,2440	48,2358	14,3675	0,0569	0,0258	0,0674	0,0258	0,0991	0,1111	0,5134	0,9053	0,0880	0,9053	1,8684	0,1393
12	38,1477	53,7837	81,3867	48,4622	14,2419	0,0577	0,0263	0,0690	0,0263	0,1006	0,1131	0,5354	0,9461	0,0915	0,9461	1,9985	0,1388
13	36,8368	51,5595	81,1046	47,8898	14,2097	0,0569	0,0256	0,0668	0,0256	0,0992	0,1082	0,5600	0,9886	0,0864	0,9886	1,9842	0,1325
14	37,0809	51,6362	81,1429	47,8972	14,3822	0,0569	0,0256	0,0686	0,0256	0,0993	0,1132	0,5090	0,9024	0,0885	0,9024	1,8372	0,1377
15	36,7106	51,5588	81,6726	47,9336	14,2020	0,0568	0,0257	0,0676	0,0257	0,0991	0,1125	0,5369	0,9462	0,0868	0,9462	1,9833	0,1344
16	36,1883	50,3998	80,9066	47,9442	14,1970	0,0566	0,0255	0,0673	0,0255	0,0984	0,1104	0,5296	0,9373	0,0863	0,9373	1,9053	0,1317
17	36,5978	51,3132	81,0695	48,0309	14,1182	0,0567	0,0255	0,0679	0,0255	0,0986	0,1117	0,5160	0,9144	0,0877	0,9144	1,8381	0,1347
18	37,3615	52,6195	81,9245	48,0929	14,1524	0,0573	0,0258	0,0669	0,0258	0,0995	0,1097	0,5222	0,9380	0,0863	0,9380	1,8932	0,1364
19	36,5075	51,2366	81,5964	48,0882	14,1693	0,0568	0,0257	0,0675	0,0257	0,0996	0,1104	0,5400	0,9291	0,0883	0,9291	1,9389	0,1377
20	36,5294	50,9167	81,7087	47,9530	14,0152	0,0568	0,0256	0,0665	0,0256	0,0988	0,1096	0,5200	0,9197	0,0864	0,9197	1,8638	0,1349
average	37,0580	51,9994	81,3697	48,0994	14,2347	0,0570	0,0257	0,0678	0,0257	0,0993	0,1113	0,5251	0,9284	0,0877	0,9284	1,9019	0,1359

see 5.6 for explanation of abbreviations

Appendix E: Averages 20 simulations using Leveling with Departure Times

Gen	ROC%	RDC%	GPU%	ASCSS%	ASCLS%	ETS (h)	StDev ETS(h)	ETL(h)	StDev ETL(h)	90% ETS(h)	90% ETL(h)	TTJSS(h)	StDev TTJSS(h)	TTJLS(h)	StDev TTJLS(h)	90% TTJSS (h)	90% TTJLS (h)
1	0,5203	0,6403	84,2897	40,3953	13,0955	0,0397	0,0055	0,0492	0,0055	0,0466	0,0563	0,4446	0,8914	0,0633	0,8914	1,6875	0,0803
2	0,5289	0,6287	85,1733	40,3813	13,1661	0,0398	0,0055	0,0494	0,0055	0,0466	0,0565	0,4622	0,9046	0,0643	0,9046	1,7481	0,0868
3	0,4918	0,5520	84,9890	40,2830	13,1631	0,0397	0,0052	0,0493	0,0052	0,0466	0,0565	0,4427	0,8947	0,0640	0,8947	1,6694	0,0838
4	0,4511	0,5513	84,9454	40,6180	13,0685	0,0398	0,0054	0,0492	0,0054	0,0466	0,0565	0,4693	0,9320	0,0648	0,9320	1,7884	0,0858
5	0,5006	0,6207	84,9360	40,4850	13,0744	0,0398	0,0056	0,0492	0,0056	0,0466	0,0563	0,4245	0,8488	0,0645	0,8488	1,5925	0,0856
6	0,6499	0,7998	84,6083	40,5493	13,1840	0,0397	0,0056	0,0493	0,0056	0,0466	0,0565	0,4384	0,8612	0,0641	0,8612	1,6671	0,0862
7	0,5917	0,6919	85,1370	40,3817	13,1777	0,0397	0,0055	0,0493	0,0055	0,0466	0,0565	0,4445	0,8809	0,0638	0,8809	1,6360	0,0847
8	0,6107	0,7409	83,9923	40,3039	13,2984	0,0397	0,0057	0,0495	0,0057	0,0465	0,0568	0,4508	0,8839	0,0641	0,8839	1,7346	0,0834
9	0,4605	0,5306	84,3663	40,1559	13,1607	0,0396	0,0053	0,0490	0,0053	0,0465	0,0565	0,4745	0,9369	0,0639	0,9369	1,7998	0,0807
10	1,1711	1,4313	85,1697	40,7270	13,0956	0,0399	0,0064	0,0496	0,0064	0,0466	0,0570	0,4583	0,9075	0,0645	0,9075	1,7253	0,0851
11	0,6681	0,8077	84,4107	40,6176	13,1551	0,0398	0,0056	0,0492	0,0056	0,0466	0,0563	0,4410	0,8748	0,0641	0,8748	1,6676	0,0852
12	0,5212	0,6114	84,7888	40,4517	13,0578	0,0397	0,0055	0,0494	0,0055	0,0466	0,0565	0,4562	0,9043	0,0673	0,9043	1,8024	0,0870
13	0,4513	0,5215	84,2447	40,2093	13,1396	0,0397	0,0052	0,0494	0,0052	0,0466	0,0568	0,4772	0,9513	0,0643	0,9513	1,7988	0,0839
14	0,4919	0,5421	84,4889	40,1532	13,2197	0,0397	0,0052	0,0494	0,0052	0,0465	0,0565	0,4389	0,8740	0,0642	0,8740	1,6540	0,0850
15	0,5700	0,6700	85,0499	40,3306	13,0456	0,0397	0,0056	0,0490	0,0056	0,0466	0,0563	0,4577	0,8997	0,0633	0,8997	1,7603	0,0827
16	0,4997	0,5797	84,2490	40,3932	13,1675	0,0397	0,0053	0,0493	0,0053	0,0465	0,0563	0,4559	0,9087	0,0643	0,9087	1,7077	0,0834
17	0,5213	0,6315	83,9423	40,3688	13,0887	0,0396	0,0054	0,0493	0,0054	0,0465	0,0563	0,4423	0,8818	0,0642	0,8818	1,6655	0,0850
18	0,7492	0,8690	85,5913	40,3640	13,1949	0,0398	0,0056	0,0494	0,0056	0,0466	0,0568	0,4512	0,9173	0,0647	0,9173	1,7144	0,0867
19	0,4005	0,4506	84,9523	40,6283	13,1708	0,0397	0,0053	0,0491	0,0053	0,0466	0,0563	0,4637	0,8958	0,0639	0,8958	1,7568	0,0832
20	0,5810	0,7113	84,8688	40,3967	13,0941	0,0397	0,0057	0,0492	0,0057	0,0466	0,0565	0,4536	0,9024	0,0636	0,9024	1,6876	0,0841
average	0,5715	0,6791	84,7097	40,4097	13,1409	0,0397	0,0055	0,0493	0,0055	0,0466	0,0565	0,4524	0,8976	0,0643	0,8976	1,7132	0,0844

see 5.6 for explanation of abbreviations.