University of Twente

Applied Mathematics
Stochastic Operations Research

master thesis

Continuous-time Markov decision process of a single-aisle warehouse

Author
D. Doppenberg

Supervisors
Prof. dr. N. M. van Dijk
Dr. J. C. W. van Ommeren

Graduation committee
Prof. dr. N. M. van Dijk
Dr. J. C. W. van Ommeren
Dr. D. Demirtas

September 26, 2018

Abstract

A common challenge in warehousing is to find a proper policy for storage assignment. Extensive research has been done on this subject; it provided the random storage policy, first open location storage policy and class-based storage policy using the cube-per-order index measure. A Markov decision process (MDP) has not yet been used to optimise the policy. Therefore, this study is designed to apply this technique.
In this project, the storage challenge is modelled as a continuous-time Markov decision process (CTMDP) that minimises the expected waiting time with the average cost criterion. Due to an excessive state space, several assumptions will be made to obtain a tractable model.
Next, the ϵ-optimal policy provided by the CTMDP will be analysed to extract easy-to-implement storage rules and to construct a heuristic storage policy that is MDP-based.
The performance of the ϵ-optimal policy and heuristic policy as well as the performance of the policies from literature are compared by value iteration. Moreover, the performances of the MDP-based policies and the policies from literature are quantified for a case study using discrete-event simulation.
The results suggest that the easy-to-implement heuristic policy extracted from the MDP improves the performance significantly compared to existing policies.

Keywords: Material handling, order picking, warehousing, storage policy, Markov decision process

Contents

1 Introduction

In warehousing, a lot of decisions have to be made in order to store and route all products as efficiently as possible. In order to choose the optimal action, a decision maker needs a framework of decision rules that determine which action has to be chosen. Markov decision processes (MDPs) provide such a framework. "Markov decision processes, also referred to as stochastic dynamic programming or stochastic control problems, are models for sequential decision making when outcomes are uncertain" (Puterman (2014), p. xv). Uncertainties in warehousing occur for example in the arrival process of containers at the warehouse and placement of products in the warehouse.

Although the choice of an MDP as framework for this sequential decision making problem seems suitable, it has not yet been considered in the literature in the light of storage allocation. Therefore, this report discusses how to use MDPs for optimising warehousing policies. To show the applicability of the framework provided by MDPs, this research will focus on a specific warehouse; it will use the warehouse of KLM Cargo as a case study.

KLM Cargo is a daughter company of KLM Royal Dutch Airlines. The warehouse of KLM Cargo is part of a sorting machine that was built in the summer of 2017. It is a single-aisle warehouse served by an elevated transport vehicle (ETV). Additionally, build-up stations are located on the bottom floor of the warehouse. Sorted packages arrive at these build-up stations to be loaded onto a container. These stations are also served by the ETV.

During an operational period of a few months, it was concluded that the sorter did not meet its expected capacity. An internal research was deployed in May 2017. KLM Cargo concluded from this research that the ETV did not meet its target; the ETV should have been able to make 75 movements an hour, while it was only possible to make 35 movements an hour. This gap between the realisable and the required number of movements per hour causes deadlocks in the system. An example of such a deadlock is the blockage of conveyors by overloading the conveyor that transports containers to the warehouse.

Therefore, Lödige - the supplier and manufacturer of the sorter - proposed to add an extra ETV to the warehouse to increase the capacity. However, from the results of an internship assignment at KLM it was concluded that the efficiency of the ETV could be increased by developing better storage policies. The internship assignment considered a discrete-event simulation of the sorter to test different storage policies of the ETV. However, due to time constraints, no attention was paid to optimising the policy using MDPs. Therefore, this report can be seen as an extension of the research done in the internship at KLM; its focus is optimising the decision making policy of the ETV using an MDP.

The main idea of this paper is to model the warehouse of KLM as an MDP. This requires to make the problem smaller in order to make it solvable. Solving the MDP yields an ϵ-optimal policy. By simulating the continuous-time Markov chain (CTMC) corresponding to this ϵ-optimal policy, easy-to-implement rules can be extracted by analysing the results. These rules together provide a heuristic policy that is MDP-based. With the value iteration algorithm, it can be determined how close to optimal the performance of the heuristic policy is. Next, the heuristic policy has to be enlarged to obtain a policy for the case study. The performance of the heuristic policy in the real-life case can be quantified using a discrete-event simulation of the sorting system. This simulation model was created during the internship period at KLM.

In this case, applying an MDP comes with the curse of dimensionality; the state space of the MDP is very large. Therefore, the MDP has to be made tractable. The main idea to make the problem tractable is to let the policy decide to allocate compartments of storage locations - a collection of multiple storage locations - for a certain type of containers instead of assigning

one storage location at the time. Additionally, the state space can be shrunk even more by putting all types of containers into two classes of containers instead of considering each type of containers individually.

This report will be presented in the following order: Section 2 will give a short overview of the work already done in literature. The overview will mainly focus on literature that considers using MDPs in the order picking branch. An introduction to the sorting machine and specifically to the warehouse and ETV will be given in Section 3. It discusses the main function of the ETV. Additionally, it discusses which parts of the system are taken into account in the MDP model. Section 4 introduces each part of the MDP model by starting to formulate the needed assumptions to make the model tractable. Next, the state space, action space, transition rates and cost function are introduced. Furthermore, Section 4 introduces the mathematical tools that are needed to transform the CTMDP. This transformation allows to solve the CTMDP with techniques for discrete-time Markov decision processes (DTMDP). Thereafter, Section 5 discusses the methods for analysing the MDP and finding the heuristic policy. Next, the results of the MDP provided by value iteration and the results of the case study provided by discrete-event simulation are presented in Section 6. Finally, the conclusion and discussion are given in Section 7.

2 Literature review

"Order picking, the process of retrieving products from storage to satisfy costumers, is a very important component of the supply chain for many companies. It constitutes 50-75% of the total operating costs for a typical warehouse" (Petersen and Aase (2004), p. 11). According to van Gils et al. (2018), the following three order picking topics are frequently discussed in literature:

- Picking: determining which product(s) to pick first.
- Storing: assigning storage locations of the warehouse to products.
- Routing: sequencing the items on the pick list to ensure a good route through the warehouse.

As discussed in de Koster et al. (2007), three frequently used storage assignment policies are:

- Random storage policy
- Closest open location storage policy
- Class-based storage policy

First of all, the random storage policy prescribes to randomly choose an empty location in the warehouse for the incoming products or pallets. The random assignment method results in a high space utilization at the expense of increased travel distance (Sharp et al., 1991). Secondly, if one applies the closest open location storage policy one always uses the first open empty location to store the product or pallet. Finally, in the class-based storage policy all products are divided into classes based on a measure of product demand, such as cube per order index (COI) as proposed by Heskett (1963) and Heskett (1964). Specific areas of the warehouse are dedicated to each of these classes. The storage within this area is done randomly. Most of the time the number of classes is restricted to three, although more classes might reduce travel times in some cases (de Koster et al., 2007). Special cases of class-based storage are dedicated storage and random storage. In the case of dedicated storage one class consists of one kind of product, while in the case of random storage there is only one class that contains all products.

Several studies have tried to optimise routing and picking using Markov decision processes. Bukchin et al. (2012) used an MDP to optimise the trade-off of going on a picking tour or waiting until more orders have arrived. Furthermore, Hodgson et al. (1985) applied a semi-Markov decision process to develop general control rules for routing an automated guided vehicle. However, even though a lot of studies have focused on optimising storage processes, MDPs have not yet been considered in this regard. Hausman et al. (1976) determined optimal boundaries for a class-based storage assignment with two and with three classes considering the racks of the warehouse to be *square-in-time*, meaning that the horizontal travel time equals the vertical travel time. This method has been extended by Eynan and Rosenblatt (1994) to any rectangular rack and by Rosenblatt and Roll (1988) to any number of classes.
Additionally, van den Berg (1996) developed a dynamic programming algorithm that finds a class allocation that minimizes the mean single command cycle time.

All studies about class-based storage assignment consider static storage allocation policies, where the allocation is determined beforehand. Therefore, this allocation cannot be changed over time. The internship assignment at KLM indicated that a storage assignment that varies over time might result in additional cost-savings. Therefore, in this report an MDP will be developed to find storage rules for an automated machine - the elevated transport vehicle - that considers a time dependent storage location assignment. To be able to solve the problem with an MDP, a class-based storage strategy will be developed.

3 Elevated transport vehicle

The elevated transport vehicle (ETV) is a machine that handles various kinds of tasks within a sorting process. In this sorter, mail and cargo packages are sorted on flight destination. These packages arrive in containers at the sorting machine. These containers have to be broken down - removing all packages from the container - at the so-called break-down stations. Thereafter, the packages are sorted and transported to a specific build-up station where the sorted packages are loaded onto a container for a specific flight. A special treatment has to be deployed when certain packages in the container are too heavy. Containers with too heavy packages are labelled as Out of Gauge (OoG) and have to be unloaded mechanically instead of manually at specially designed OoG build-up stations.
In between each of the sorting phases, the containers can be temporarily stored in a warehouse, see Figure 1. This warehouse is served by the ETV.

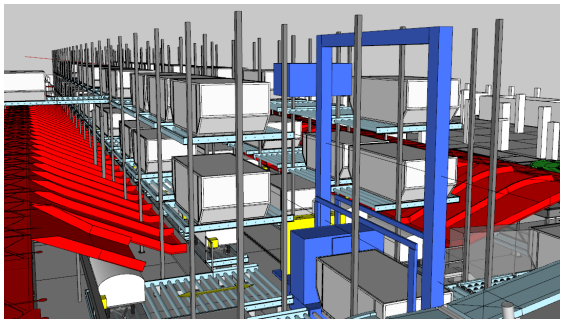

Figure 1: A figure of the the ETV and the warehouse. It shows the three layers of the warehouse. At the lowest layer, the build-up stations are positioned. The middle and upper layer contain all storage positions. The blue machine is the ETV.

Besides managing the warehouse, the ETV sends containers to the outfeed and serves the build-up stations that are located beneath the warehouse. The flow of the containers between components of the sorter is visualised in a simplified manner in Figure 2; a detailed overview can be found in Appendix A.
Because of the great variety of tasks, it is important to have clear rules for decision making. These rules can be obtained by applying an MDP on the warehouse.

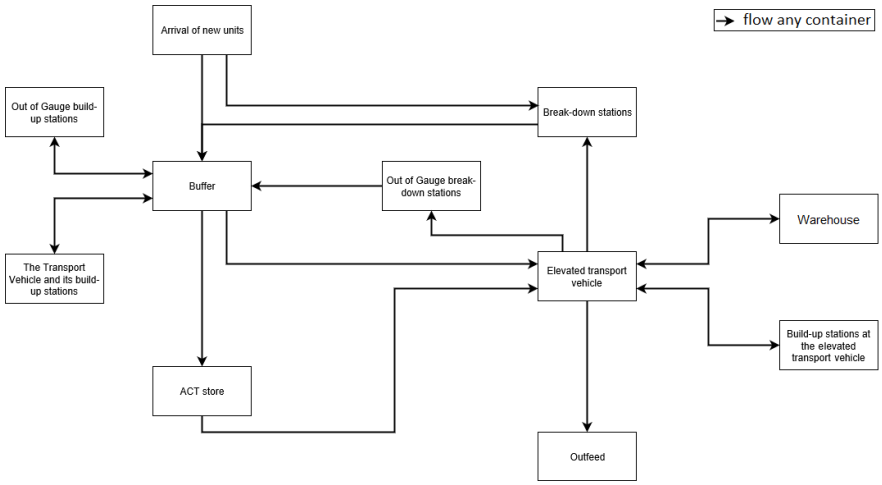

Figure 2: A schematic overview of the flow of containers in the sorting system.

3.1 Overview

In this section, the function of the ETV will be worked out in detail by analysing all tasks that come with serving different kinds of containers. This overview is helpful in constructing the model of the machine.

The ETV is an automated machine that manages a single-aisle warehouse. It deals with various kinds of containers; all containers which are handled by the ETV are:

- Out of Gauge (OoG) containers
- Empty containers
- Built-up containers
- ACT (Active Cooled Temperature) containers; containers that have to cool their payload in order to maintain the quality of the load
- Not yet broken-down containers

On top of the different types of containers, a container can have two sizes: belly cart (BC) and unit loading device (ULD).
An arrival of specific container at the ETV requires the ETV to do a specific task. The time it takes to complete a task depends completely on the current and future position of the container, the current position of the ETV and the loading times. In other words; the service time depends on the travel time of the ETV and the loading times. All specific actions of the ETV that come with an arriving tasks are mentioned below:

- Storing a container that has not been broken-down in the warehouse.
- Removing a not broken-down container from the warehouse and send it to the break-down stations.
- Storing an OoG container in the warehouse.
- Sending an OoG container to the OoG break-down stations.
- Storing an empty container in the warehouse.
- Removing an empty container from the warehouse and send it to the infeed of the ETV.
- Removing an empty container from the warehouse and place it at a build-up station.
- Removing an empty container from the queue of the ETV and place it at a build-up station.
- Storing a container that has been built-up in the warehouse.
- Removing a built-up container from the warehouse and send it to the outfeed.
- Removing a container from a build-up station and place it in the warehouse.
- Removing a container from a build-up station and send it to the outfeed.
- Sending an ACT to the outfeed.

However, taking all these actions and containers into account would make the model too complex. Therefore, it has to be decided which actions and which types of containers are vital.

3.2 Scope

All tasks can be summarised as placing containers in and retrieving containers from the warehouse and sending containers to the break-down stations, build-up stations and outfeed. To be able to operate efficiently, it is important to place containers at strategic locations in the warehouse. For example, placing the containers with the highest throughput rate as close as possible to the infeed of the system could reduce the average service and waiting time. Cost reductions can also be made by optimizing the service discipline and routing policy. However, routing is not an issue since the warehouse has only one aisle and the ETV can only transport one container at the time. Therefore, in the optimisation of the policy both the service discipline and storage assignment will be taken into account.

By assuming that the infeed point and outfeed point have the same location, the following tasks will coincide with each other: sending containers from the warehouse to break-down stations, sending them to build-up stations not located at the ETV and sending them to the outfeed. Therefore, OoG containers, built-up containers and not yet broken-down containers do not have to be distinguished in the model since they follow the same procedure from the point of view of the ETV; these containers will be hereafter referred to as non-empty containers.
Additionally, ACT containers can be excluded from the model, since they never have to be stored and have the highest priority due to their high value payload.
On top of that, the difference between ULDs and BCs are discarded, otherwise the model would get too complicated.

Although the number of different types of containers can be decreased significantly, the difference between empty and non-empty containers is very important: they are stored with different purposes. Non-empty containers have to be stored until they are sent to the outfeed of the sorter or to the break-down stations, while empty containers are stored until they are sent to build-up stations.
All in all, the list of actions to include in the model can be reduced to:

- Storing a non-empty container in the warehouse.
- Removing a non-empty container from the warehouse.
- Storing an empty container in the warehouse.
- Removing a built up container from a build-up station, placing it in the warehouse and sending a new empty container to this station.
- Removing a built up container from a build-up station, sending it to the outfeed and sending a new empty container to this station.

3.3 Service time

The service time of the ETV can be easily calculated. Since the ETV cannot stand still between build-up station, the travel distance can be calculated by counting the number of build-up stations that have to be passed in order to do an action. Then, the travel distance is obtained by multiplying the number of build-up stations to pass by the distance between build-up stations. Since the acceleration and maximum speed of the ETV are known, the travel time can be calculated assuming that the acceleration of the ETV is homogeneous.
Additionally, for each action it is known beforehand how many times containers have to be unloaded from and loaded onto the ETV. Since every container that is loaded onto the ETV has to be unloaded as well, only the number of pick-ups has to be counted. Then, the total time of loading containers onto and removing containers from the ETV is equal to the number of

pick-ups times 20 seconds. This pick-up time is determined in the internship assignment of KLM.
To conclude, the service time is equal to the travel time and loading time together. An elaborate description of the algorithm to calculate the service time has been given in the internship report. For completeness, the algorithm is added to the appendix, see Appendix B.

4 Model

An MDP provides a mathematical framework in sequential decision making; it optimises the decision of what action to choose based on the current state of the system. The set of all possible states is called the state space. In many practical implementation, the state space comes with the curse of dimensionality. Unfortunately, this problem is faced in the context of this warehouse as well.
The state space of the MDP consists of the status of the warehouse. The status contains all information of the current state of the warehouse and can be divided into four parts.
First of all, the status indicates which storage locations are empty and which are filled. Moreover, it also provides information about whether a filled position contains an empty container, a non-empty container or a container that has to be removed from the warehouse. The size of the state space is extremely affected by this information. Since there are 160 storage positions, the number of possible combinations is $4^{160} \approx 2.14 \times 10^{96}$.
Secondly, the choice of action depends on the size of the queue, which containers are queuing and in which order they are queuing. In order to determine the number of combinations, suppose that the queue has capacity M. From Section 3.2 can be concluded that there are 5 different tasks. Therefore, the number of possible combinations for the queue is $\sum_{i=0}^{M} 5^i$.
On top of that, the status contains information about the statuses of the build-up stations beneath the warehouse. These stations can have the status processing, done or empty. Since there are 40 build-up stations, this comes with another $3^{40} \approx 1.22 \times 10^{19}$ combinations.
Finally, the current position of the ETV in the warehouse influences the decision making. Since the ETV has 3 possible vertical positions and 21 horizontal positions, there are 63 possible positions for the ETV.
All in all, formulating the problem in its most detailed form results in an intractable model due to the excessive amount of states. Thus, several adjustments and simplifications have to be made to shrink the state space.

4.1 Assumptions

This section will introduce the assumptions that are needed to make the model tractable.

4.1.1 Compartments

A reduction in the number of states is possible by assigning compartments, i.e. a collection of multiple storage locations, to one type of container instead of assigning the storage locations individually. For example, the storage of the incoming containers within such compartments can be done by assigning the first open storage location or a random open storage location in a compartment. Suppose the warehouse is divided into ten compartments, the number of combinations for assigning of compartments is $4^{10} = 1{,}048{,}576$ instead of 4^{160}.

By assigning compartments, the problem can be transformed into a new problem with a smaller warehouse, see Figure 3. The number of storage locations in the smaller warehouse is equal to the number of compartments in the original warehouse; one storage location in the small warehouse represents one storage compartment in the original warehouse. Therefore, an arriving container in the transformed problem can be interpreted as an arriving batch of containers of a size equal to the capacity of one compartment in the original problem. Similarly, a build-up station in the transformed problem processes a batch of containers and can therefore be interpreted in the original problem by as many build-up stations as the batch size.

Instead of solving the original problem, the transformed problem can be solved using MDP techniques.

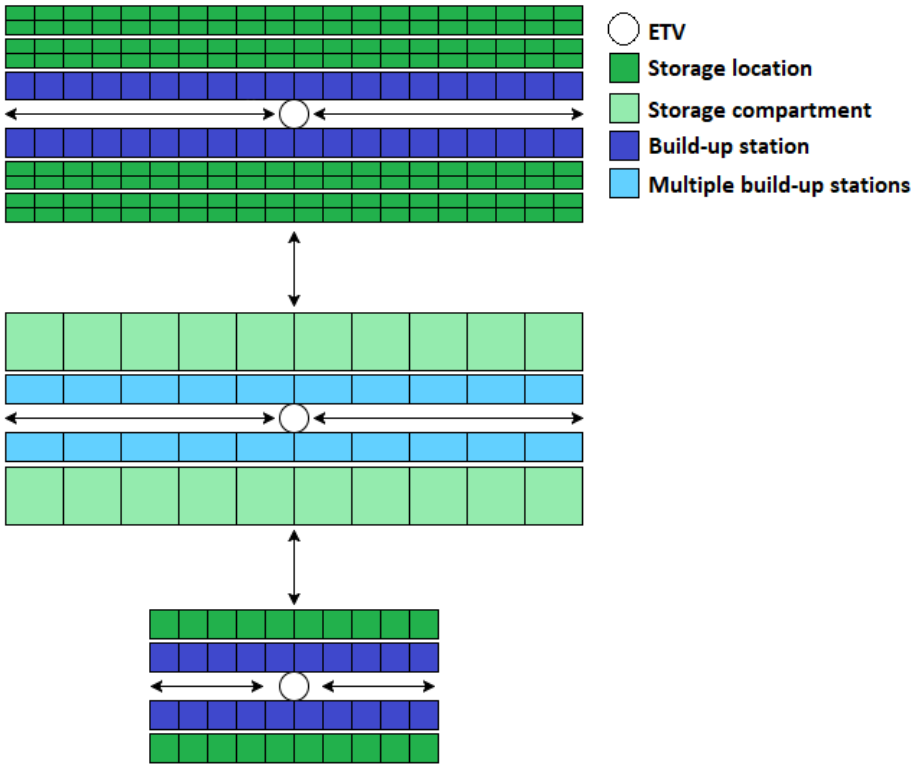

Figure 3: This figure shows the transformation of the original warehouse to the small warehouse. In this example, the warehouse of 160 storage locations and 40 build-up stations is divided into 20 compartments. This results in a small warehouse of 20 storage locations and 20 build-up stations.

4.1.2 Exponentially distributed staying times and build up times

It is possible to reduce the number of types of containers even more by assuming that the staying times of non-empty containers in the warehouse are exponentially distributed. Normally, a non-empty container requests to be removed from the warehouse when, for example, its flight will depart soon. Assuming that all non-empty containers stay for an exponential distributed time in the warehouse, all non-empty containers currently in the warehouse are evenly likely to have made the request when a request arrives. This is a result of the memoryless property of the exponential distribution. Therefore, only how many requests are currently queued have to be kept in mind and not the position of the requesting container. This results in a reduction of the state space since only the following statuses of storage positions have to be considered: empty, filled with an empty container and filled with a non-empty container.
Similarly, by assuming that the build-up times of empty containers at build-up stations are exponentially distributed, the information about which build-up station has finished the build-up process can be neglected. Therefore, only the number of build-up stations that have currently finished the process has to be determined; the statuses of the build-up stations can be neglected.

4.1.3 Travel time

The warehouse consists of two opposite racks with each three shelves located on top of each other. The lower shelf consists of 20 build-up stations and the other shelves consist of 20 storage positions. This comes down to a length and height of the warehouse of approximately 90 and 8 meters respectively. Then, Table 1 shows that the horizontal travel time exceeds the vertical travel time in most cases. Since the ETV can move simultaneously horizontally and vertically, it is reasonable to assume that the travel time depends only on the horizontal travel time. Therefore, only the horizontal position of the ETV is of interest instead of both the horizontal and the vertical position.

Table 1: The travel times for horizontal and vertical movement. The values Δx and Δy indicate the number of build-up stations and store layers that have to be passed respectively, i.e. the horizontal and vertical travel distance respectively.

	Travel time	
d	$(\Delta x = d, \Delta y = 0)$	$(\Delta x = 0, \Delta y = d)$
0	0.0 s	0.0 s
1	7.456 s	5.143 s
2	10.954 s	10.286 s
3	13.417 s	
4	15.667 s	
5	17.917 s	
6	20.167 s	
7	22.417 s	
8	24.667 s	
9	26.917 s	
10	29.167 s	
11	31.417 s	
12	33.667 s	
13	35.917 s	
14	38.167 s	
15	40.417 s	
16	42.667 s	
17	44.917 s	
18	47.167 s	
19	49.417 s	
20	51.667 s	

4.1.4 Built-up containers

After a container has been built up at a build-up station, the container goes immediately to the outfeed if its flight is close to close, i.e. the flight will depart soon. Otherwise, it will be stored in the warehouse until its flight is close to close. The precise definition of when a flight is close to close is a system setting which can be adjusted. The internship assignment uses a close to close time of 16 hours, meaning that a flight is assumed to depart soon if its scheduled departure time is within 16 hours from now.
To keep the model tractable, it is necessary to assume that built-up containers at build-up stations are immediately sent to the outfeed. This is a reasonable assumption when the close to close time is 16 hours, since in general containers are built up for flights that depart within 16 hours or less from then. However, the sorting system is designed to handle lower close to

close times. When the close to close time is set lower, built-up containers will be stored in the warehouse first. This assumption comes with a loss of non-empty containers that have to be stored in the warehouse. To compensate for this reduction, the arrival rate of non-empty containers can be increased accordingly. In this case, the number of containers that go to the warehouse after they are built up can be added to the arrivals of non-empty containers.

4.1.5 Queue

An other way to reduce the state space is truncating the queue of the system. However, instead of using one queue, the queue is divided in four new queues which only allow one kind of task. Thereafter, the capacity of each of these queues will be truncated. Obviously, this decreases the number of states since the queue is truncated. Moreover, the state space is also reduced by the fact that the order in which the tasks have arrived is neglected.

4.1.6 Exponentially distributed processes

Since some stochastic processes are already assumed to be exponentially distributed, a well suited MDP model is a continuous-time Markov decision process (CTMDP). Here, all probability distributions have to be exponential distributions. Thus, as already assumed for the staying times of non-empty containers and build-up times at the build-up stations, all other stochastic processes are also assumed to be exponentially distributed. All these processes are listed below:

- Arrival process of empty containers at the ETV
- Arrival process of non-empty containers at the ETV
- Arrival process of a request to pick up an empty container
- Staying time of non-empty containers in the warehouse
- Filling of empty containers at the build-up stations
- Service times of the ETV

In specific, the assumption of exponentially distributed service times is quite rigorous, since the service time depends only on the travel distance and the number of pick-ups. Therefore, the service time is not random. To preserve the fact that a longer travel distance implies a longer service time, the exponential parameter for each service time will be chosen as $1/\sigma(x_1, x_2)$ where $\sigma(x_1, x_2)$ is the actual service time corresponding to the travel distances x_1 and the number of pick-ups x_2. Thus, the service time is in expectation equal to the actual service time. In this case x_1 is rather a list of travel distances; the ETV has to make multiple movements to finish most actions. For example, it has to go from the current position to the infeed to pick up a container. Then, it has to go to an empty location in the warehouse to store the container. In this case, x_1 would contain two elements: one for going to the infeed and the other of going to the empty location.

4.2 Markov decision process formulation

An MDP is defined by its state space, action space, transition rates and costs. The state space denoted by S describes all the possible states of the system, while the action space A_s is the set of actions that can be chosen in a certain state $s \in S$. The transition rates determine the rate at which the system changes from a certain state $s \in S$ to a new state $s' \in S$ when choosing an action $a \in A_s$. The cost function determines the value of choosing an action $a \in A_s$ in a certain state $s \in S$.
All these parts will be worked out in the following sections.

4.2.1 State space

The state space will be written down formally using the assumptions mentioned earlier. Define n_c as the number of columns in the warehouse. A column in the warehouse consists of the compartments and build-up stations that are directly on the opposite sides of the warehouse, see Figure 4. Additionally, define n_z as the number of compartments used. Then, introduce $w_i \in \{0, 1, 2\}$ as the warehouse information of compartment i for $1 \leq i \leq n_z$. Compartment w_i takes value 0 if it is empty, value 1 if it is filled with an empty container and value 2 if it is filled with a non-empty container. In addition, $\bar{w} = (w_1, w_2, ..., w_{n_z})$ is defined as the vector containing the information of all compartments.

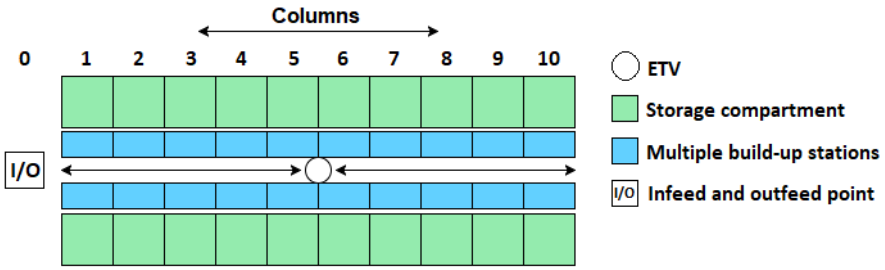

Figure 4: An example of how the columns of the warehouse are labelled for $n_c = 10$ and $n_z = 20$.

Moreover, define $x \in \{0, 1, 2, ..., n_c\}$ as the horizontal position of the ETV in the warehouse. The horizontal component takes the value of the label of the column at which it stand still (it takes 0 if it stands still at the infeed). The labelling procedure is visualised in Figure 4. Furthermore, define $M \in \mathbb{N}$ as the maximum capacity of a queue. Then, define the following queues:

- $q_{0,0} \in \{0, 1, ..., M\}$ is the number of empty containers in the queue that have to be stored.
- $q_{0,1} \in \{0, 1, ..., M\}$ is the number of requests to pick up an empty container from the warehouse.
- $q_{1,0} \in \{0, 1, ..., M\}$ is the number of non-empty containers in the queue that have to be stored.
- $q_{1,1} \in \{0, 1, ..., M\}$ is the number of non-empty containers that have to be removed from the warehouse.
- $q_2 \in \{0, 1, ..., M\}$ is the number of build-up stations that have completed their service, request the ETV to remove the container and pick up an empty container.

In addition, $\bar{q} = (q_{0,0}, q_{0,1}, q_{1,0}, q_{1,1}, q_2)$ contains all information about the number of tasks in each queue.

Then, the state space S is given by

$$S = \Big\{ s = (\bar{w}, \bar{q}, x) \Big| \, \bar{w} \in \{0,1,2\}^{n_z}, \bar{q} \in \{0,1,...,M\}^5, x \in \{0,1,...,n_c\}, ... \\ ..., q_{0,1} \leq \sum_{i=1}^{n_z} \mathbb{1}\{w_i = 1\}, q_{1,1} \leq \sum_{i=1}^{n_z} \mathbb{1}\{w_i = 2\} \Big\}. \tag{1}$$

The constraints in (1) make sure that the number of requests to remove containers from the warehouse cannot exceed the number of containers present in the warehouse. Using this formulation, the values for n_c, n_z and M can be chosen such that the problem is tractable. An upper bound for the number of states $|S|$ can be computed in the following way

$$|S| < 3^{n_z} \cdot (M+1)^5 \cdot (n_c + 1). \tag{2}$$

The upper bound is found by neglecting the constraints in (1). A lower bound can be found by removing not admissible states from the grand total

$$|S| > 3^{n_z}(M+1)^5(n_c+1) - 2\sum_{j=0}^{M-1} \binom{n_z}{j} (M-j)(M+1)^4(n_c+1), \tag{3}$$

provided that $n_z > M$.

In the model, the compartments are chosen in such a way that $n_c = n_z$. Then, intervals for the number of states for different combinations for n_z and M are given in Table 2 (provided $n_c = n_z$).

Table 2: An interval for the number of states per combination of n_z and M provided $n_c = n_z$. The lower bound of the interval is given by (3) and the upper bound by (2).

n_z / M	1	2	3	4
5	(40,512; 46,656)	(214,326; 354,294)	(460,800; 1,492,992)	(296,250; 4,556,250)
10	(20,424,800; 20,785,248)	(145,064,601; 157,837,977)	(525,274,112; 665,127,936)	(1.2342×10^9; 2.0298×10^9)
16	(2.3382×10^{10}; 2.3417×10^{10})	(1.7602×10^{11}; 1.7783×10^{11})	(7.2141×10^{11}; 7.4936×10^{11})	(2.0668×10^{12}; 2.2869×10^{12})
20	(2.3424×10^{12}; 2.3431×10^{12})	(1.7750×10^{13}; 1.7793×10^{13})	(7.4185×10^{13}; 7.4980×10^{13})	(2.2135×10^{14}; 2.2882×10^{14})

The table shows that only the combination $(n_z, M) = (5, 1)$ is feasible; in all other cases the number of states is excessive or the combination of n_z and M is not reasonable. For example, the capacities of all queues together exceed the capacity of the warehouse when $n_z = 5$ and $M \geq 2$.

4.2.2 Action space

In each state, one action from a set actions has to be chosen by the decision maker. Each action can only be chosen under certain conditions. The complete lists of possible actions with the condition under which it can be chosen will be given; note that the label of an action is matches the label of the corresponding queue.

- $a_{0,0,c}$:
  - place an empty container from the queue at compartment c in the warehouse for each $1 \leq c \leq n_c$,
  - only available if $q_{0,0} > 0$ and $w_c = 0$.

- $a_{0,1,c}$:
  - remove an empty container from compartment c in the warehouse for each $1 \leq c \leq n_c$,
  - only available if $q_{0,1} > 0$ and $w_c = 1$.
- $a_{1,0,c}$:
  - place a non-empty container from the queue at compartment c in the warehouse for each $1 \leq c \leq n_c$,
  - only available if $q_{1,0} > 0$ and $w_c = 0$.
- $a_{1,1}$:
  - remove a non-empty container from the warehouse,
  - only available if $q_{1,1} > 0$.
- $a_{2,c}$:
  - remove a non-empty container from a build-up station, send it to the outfeed and place an empty container from compartment c in the warehouse at the build-up station for each $1 \leq c \leq n_c$,
  - only available if $q_2 > 0$ and $w_c = 1$.
- a_2:
  - remove an non-empty container from a build-up station, send it to the outfeed and place an empty container from the queue,
  - only available if $q_2 > 0$ and $q_{0,0} > 0$.
- $a_{3,c}$:
  - move the ETV to compartment c in the warehouse for each $0 \leq c \leq n_c$,
  - only available if all the other actions are not possible.

To formalize, define $I = \{1, 2, ..., n_c\}$ as the set of all indices of the compartments and $I_0 = I \cup \{0\}$. Then, the action space A_s for state $s \in S$ is defined as

$$A_s = \left\{ \begin{array}{ll} a_{0,0,c} & \text{if } q_{0,0} > 0 \text{ and } w_c = 0\text{; for each } c \in I, \\ a_{0,1,c} & \text{if } q_{0,1} > 0 \text{ and } w_c = 1\text{; for each } c \in I, \\ a_{1,0,c} & \text{if } q_{1,0} > 0 \text{ and } w_c = 0\text{; for each } c \in I, \\ a_{1,1} & \text{if } q_{1,1} > 0\text{;} \\ a_{2,c} & \text{if } q_2 > 0 \text{ and } w_c = 1\text{; for each } c \in I, \\ a_2 & \text{if } q_2 > 0 \text{ and } q_{0,0} > 0\text{,} \\ a_{3,c} & \text{if all other actions are not possible; for each } c \in I_0. \end{array} \right\} \forall s \in S.$$

4.2.3 Transition rates

Upon choosing an available action from the action set being in state $s \in S$, the system jumps from s to a new state. However, not all states are reachable from s and the transition rates may be different from going from s to a state $s_1 \in S$ or to a state $s_2 \in S$, provided that $s_1 \neq s_2$. All these features have to be kept in mind by the decision maker; the transition rate function captures all these features. First of all, define the following exponential parameters

- $\lambda_{0,0}$: the arrival rate of a task in queue $q_{0,0}$
- $\lambda_{0,1}$: the arrival rate of a task in queue $q_{0,1}$
- $\lambda_{1,0}$: the arrival rate of a task in queue $q_{1,0}$
- $\lambda_{1,1}$: the exponential parameter of the average staying time of a non-empty container in the warehouse
- λ_2: the arrival rate of a task in queue q_2
- $\mu(x_1, x_2) = \begin{cases} 0 & \text{if } x_1 = [0] \text{ and } x_2 = 0 \\ 1/\sigma(x_1, x_2) & \text{otherwise.} \end{cases}$: the exponential parameter of the service time distribution that depends on the travel time $\sigma(x_1, x_2)$, where x_1 is the vector containing the travel distances and where x_2 is the number of containers to pick up.

Furthermore, define the function $t(s'|s, a)$ as the transition rate function that maps the current state s, the possible new state s' and the chosen action a on the transition rate of going from state s to s' when choosing a, formally $t : S \times S \times A \to [0, \infty)$. Additionally, define the following sets:

- $I_1(s) = \{i \mid w_i = 1 \ \wedge \ w_i \subset s \text{ for } 1 \leq i \leq n_z\}$; the set of indices of all compartments that contain an empty container.
- $I_2(s) = \{i \mid w_i = 2 \ \wedge \ w_i \subset s \text{ for } 1 \leq i \leq n_z\}$; the set of indices of all compartments that contain a non-empty container.

Then, the transition rate function for going from a state $s \in S$ choosing an action $a \in A_s$ to a state $s' = (\bar{w}', \bar{q}', x') = (w'_1, w'_2, ..., w'_{n_z}, q'_{0,0}, q'_{0,1}, q'_{1,0}, q'_{1,1}, q'_2, x') \in S$ can now be defined provided that $n_c = n_z$. When choosing a certain action, the transition rate of going to s' will be positive when certain conditions are met. For example, suppose $a_{0,0,1}$ is chosen. Then, the transition rate of going to s' will be positive if all of the following conditions are met:

1. $w'_1 = 1$
2. $\sum_{i=2}^{n_z} \mathbb{1}\{w'_c \neq w_c\} = 0$
3. $q'_{0,0} - q_{0,0} = -1$
4. $\sum_{i=2}^{5} \mathbb{1}\{\bar{q}'_i \neq \bar{q}_i\} = 0$
5. $x' = 1$

Alternatively, these conditions can also be captured by subtracting s from s'

$$s' - s = (1, \underbrace{0, 0, ..., 0}_{n_z - 1}, -1, 0, 0, 0, 0, 1 - x) \tag{4}$$

Although using (4) might be less formal, it will result in a more intuitive formulation of the transition rate function. Therefore, using the more intuitive approach, the transition rate function is defined in (5). The more formal formulation can be found in Appendix C.

$$
t(s'|s,a) = \begin{cases}
\mu([x,c],1) & \text{if } a = a_{0,0,c} \text{ and} \\
& s' - s = (\underbrace{0,0,...,0}_{c-1},1,\underbrace{0,0,...,0}_{n_z-c},-1,0,0,0,0,c-x); \\
& \text{for each } c \in I, \\
\mu([|x-c|,c],1) & \text{if } a = a_{0,1,c} \text{ and} \\
& s' - s = (\underbrace{0,0,...,0}_{c-1},-1,\underbrace{0,0,...,0}_{n_z-c},0,-1,0,0,0,-x); \\
& \text{for each } c \in I, \\
\mu([x,c],1) & \text{if } a = a_{1,0,c} \text{ and} \\
& s' - s = (\underbrace{0,0,...,0}_{c-1},2,\underbrace{0,0,...,0}_{n_z-c},0,0,-1,0,0,c-x); \\
& \text{for each } c \in I, \\
\frac{1}{|I_2(s)|}\ \mu([|x-i|,i],1) & \text{if } a = a_{1,1} \text{ and} \\
& s' - s = (\underbrace{0,0,...,0}_{i-1},-2,\underbrace{0,0,...,0}_{i-c},0,0,0,-1,0,-x); \\
& \text{for each } i \in I_2(s), \\
\frac{1}{n_c}\mu([|x'-x|,c,|c-x'|]),2) & \text{if } a = a_{2,c} \text{ and } x' > 0 \text{ and} \\
& s' - s = (\underbrace{0,0,...,0}_{c-1},-1,\underbrace{0,0,...,0}_{n_z-c},0,0,0,0,-1,x'-x); \\
& \text{for each } c \in I_1(s), \\
\frac{1}{n_c}\mu([|x'-x|,x',x'],2) & \text{if } a = a_2 \text{ and } x' > 0 \text{ and} \\
& s' - s = (\underbrace{0,0,...,0}_{n_z},-1,0,0,0,-1,x'-x); \\
\mu([|x-c|],0) & \text{if } a = a_{3,c} \text{ and} \\
& s' - s = (\underbrace{0,0,...,0}_{n_z},0,0,0,0,0,c-x); \\
& \text{for each } c \in I_0, \\
\lambda_{0,0} & \text{if } q_{0,0} < M \text{ and} \\
& s' - s = (\underbrace{0,0,...,0}_{n_z},1,0,0,0,0,0), \\
\lambda_{0,1} & \text{if } q_{0,1} < M \text{ and } |I_1(s)| - q_{0,1} > 0 \text{ and} \\
& s' - s = (\underbrace{0,0,...,0}_{n_z},0,1,0,0,0,0), \\
\lambda_{1,0} & \text{if } q_{1,0} < M \text{ and} \\
& s' - s = (\underbrace{0,0,...,0}_{n_z},0,0,1,0,0,0), \\
\lambda_{1,1}||I_2(s)| - q_{1,1}| & \text{if } q_{1,1} < M \text{ and } |I_2(s)| - q_{1,1} > 0 \text{ and} \\
& s' - s = (\underbrace{0,0,...,0}_{n_z},0,0,0,1,0,0), \\
\lambda_2 & \text{if } q_2 < M \text{ and} \\
& s' - s = (\underbrace{0,0,...,0}_{n_z},0,0,0,0,1,0), \\
0 & \text{otherwise.}
\end{cases} \tag{5}
$$

4.2.4 Costs

Choosing a certain action $a \in A_s$ in state $s \in S$ comes not only with a change in the state of the system but it comes also with costs. In the case of the warehouse, the costs are defined as the waiting time of containers in the queue of the ETV. These costs have to be minimised. Therefore, the MDP will be evaluated using the average cost criterion; when minimising the expected waiting time it is more logical to look at the average cost than at (total) discounted costs. Thus, the cost function $k(s, a)$ determines the costs of the waiting time of containers in the queue choosing action a in state s, i.e. $k : S \times A \to [0, \infty)$. Define the following costs:

- $c_{0,0}$ as the costs of waiting in queue $q_{0,0}$ for a unit of time
- $c_{0,1}$ as the costs of waiting in queue $q_{0,1}$ for a unit of time
- $c_{1,0}$ as the costs of waiting in queue $q_{1,0}$ for a unit of time
- $c_{1,1}$ as the costs of waiting in queue $q_{1,1}$ for a unit of time
- c_2 as the costs of waiting in queue q_2 for a unit of time.

Moreover, define $\bar{c} = (c_{0,0}, c_{0,1}, c_{1,0}, c_{0,1}, c_2)$.

The MDP minimises the average waiting time when choosing $\bar{c} = (1, 1, 1, 1, 1)$, while the MDP minimises the average weighted waiting time when choosing $\bar{c}$ differently. The costs of having a container in the queue has to be multiplied with the expected sojourn time τ in a state. Moreover, because the problem is transformed to a smaller problem, the waiting time of one container in the transformed problem represents the waiting time of a batch of containers in the original problem. Therefore, the waiting time has to be multiplied with the batch size. Then, the cost function is defined as

$$\begin{aligned} k(s,a) &= \frac{160}{n_z} \cdot \langle \bar{c}, \bar{q} \rangle \cdot \mathbb{E}[\tau | s, a] \\ &= \frac{160}{n_z} \cdot \langle \bar{c}, \bar{q} \rangle \cdot \frac{1}{\sum_{s' \in S} t(s'|s,a)}. \end{aligned}$$

4.3 Uniformisation

The CTMDP can be solved by applying a uniformisation transformation. This means that all the sojourn times are made independent from action and state. This is done by allowing the system to have additional self-transitions. First of all, the theory on the uniformisation transformation will be introduced in the light of continuous-time Markov chains (CTMCs). Afterwards, the theory for CTMCs will be transformed and extended such that it holds for CTMDPs.

4.3.1 Continuous-time Markov chain

Suppose a CTMC - where actions do not play a role - has transition rates $q(j|i)$ for going from i to j for $i \neq j$ and total exit rates defined as $q(i) = \sum_{k \neq i} q(j|i)$ (Puterman, 2014). As Puterman (2014) describes, the probabilistic behaviour of a CTMC can be summarized by its infinitesimal generator A, which is a matrix satisfying

$$A(i,j) = \begin{cases} -[1 - p(i|i)]q(i), & j = i \\ p(j|i)q(i), & j \neq i \end{cases} \tag{6}$$

where $p(j|i)$ is the transition probability of going fromn state i to state j.

The infinitesimal generator determines the probability distribution of the system state over time t through the differential equations

$$\frac{d}{dt}P^t(i,j) = \sum_{k\in S} A(k,j)P^t(i,k)$$

or

$$\frac{d}{dt}P^t(i,j) = \sum_{k\in S} P^t(k,j)A(i,k),$$

where P^t is t-step transition probability matrix of the Markov chain. These equations are known as the Kolmogorov forward and backward equations respectively. By construction, processes with the same infinitesimal generator and initial distribution have identical finite-dimensional distributions. This gives rise to the following observation: the CTMC can be analysed more easily if it is transformed into a more simple process with the same infinitesimal generator. This is one way to justify the uniformisation.

Transformation

First of all, the CTMC and its uniformisation will be introduced following the notation in van Dijk et al. (2018). The CTMC has $q(j|i)$ as transition rate of going from state i to state j, $j \neq i$. Then, the total exit rate of state i is given by

$$q(i) = \sum_{k\neq i} q(k|i).$$

These characteristics of the chain are contained in the transition rate matrix Q, which is defined as

$$Q(i,j) = \begin{cases} -q(i) & j = i, \\ q(j|i) & j \neq i. \end{cases} \tag{7}$$

Now, the uniformisation of the CTMC can be defined. To be able to apply the uniformisation transformation, it is necessary that all exit rates are uniformly bounded, i.e. there exists a constant $B < \infty$ such that

$$q(i) = \sum_{j\neq i} q(j|i) \leq B \ \forall i \in S. \tag{8}$$

Then, the transition probability matrix of the uniformisation - all parts of the uniformisation will be recognisable by a tilde '~' - is given by

$$\tilde{P}(i,j) = \tilde{p}(j|i) = \begin{cases} 1 - q(i)/B, & j = i, \\ q(j|i)/B, & j \neq i, \end{cases} \tag{9}$$

or

$$\tilde{P} = I + \frac{1}{B}Q.$$

Thus, the uniformisation can be interpreted as an equivalent process where the original CTMC is observed at exponentially distributed random times with rate B. Alternatively, the transformation may be viewed as adding fictitious self-transitions to the original chain (Puterman, 2014). Furthermore, as described by van Dijk et al. (2018), the transformation can be interpreted as a time-discretisation. Suppose the step-size $h \leq \frac{1}{B}$ is sufficiently small such that the continuous-time transition probabilities become proportional to the length of the time interval. In this case, the following transition probability matrix can be used in the discrete-time analogue

$$P = I + hQ.$$

Theorem 4.1. *Assume that* (8) *holds. A CTMC and its uniformisation defined through* (9) *and* (8) *have the same infinitesimal generator.*

PROOF. The infinitesimal generator of the uniformisation is given by

$$\tilde{A}(i,j) = \begin{cases} -(1-\tilde{p}(i|i))\tilde{q}(i) & j = i \quad (a) \\ \tilde{p}(j|i)\tilde{q}(i) & j \neq i \quad (b) \end{cases}$$

Part (a) can be rewritten using (9) as:

$$\begin{aligned} -(1-\tilde{p}(i|i))\tilde{q}(i) &= -(1-(1-\sum_{k\neq i} q(k|i)/B))B \\ &= -\sum_{k\neq i} q(k|i) \\ &= -\frac{\sum_{k\neq i} q(k|i)}{q(i)} q(i) \\ &= -\sum_{k\neq i} p(k|i)q(i) \\ &= -(1-p(i|i))q(i) \end{aligned}$$

Part (b) can be rewritten using (9) as:

$$\begin{aligned} \tilde{p}(j|i)\tilde{q}(i) &= \frac{q(j|i)}{B} B \\ &= \frac{q(j|i}{q(i)} q(i) \\ &= p(j|i)q(i). \end{aligned}$$

Rewriting part (a) and (b) establishes

$$\tilde{A} = A,$$

since the infinitesimal generator A of the CTMC is defined as (6). □

An example of how a CTMC can be transformed will be given below.

Example 1. *Suppose a two state CTMC has transition rates* $q(s_2|s_1) = 2$ *and* $q(s_1|s_2) = 10$*, see Figure 5.*

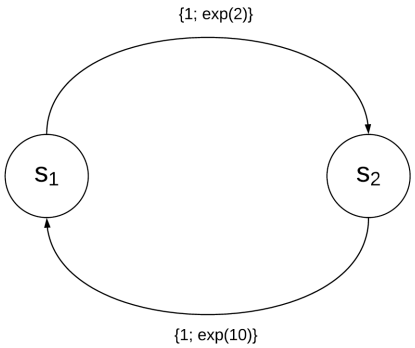

Figure 5: Original CTMC. The first position in the braces represents the probability of the transition and the second position represents the probability distribution of the duration of the transition.

Then, it has transition matrix

$$Q = \begin{bmatrix} -2 & 2 \\ 10 & -10 \end{bmatrix}.$$

Thus, the exit rates are $q(s_1) = 2$ *and* $q(s_2) = 10$*. Therefore, B has to suffice*

$$B \geq \max\{q(s_1), q(s_2)\} = \max\{2, 10\} = 10.$$

Suppose $B = 10$*. Then, the uniformisation has transition matrix*

$$\tilde{P} = I + \frac{1}{B}Q = \begin{bmatrix} \frac{4}{5} & \frac{1}{5} \\ 1 & 0 \end{bmatrix}$$

This corresponds with the CTMC in Figure 6.

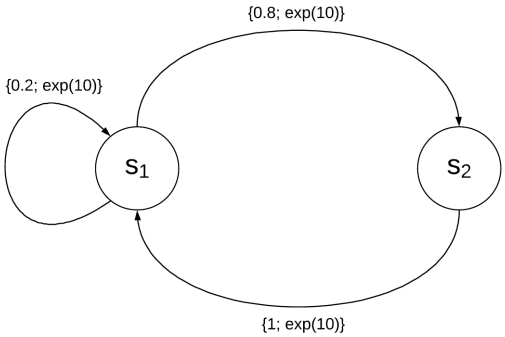

Figure 6: Uniformisation with $B = 10$. The first position in the braces represents the probability of the transition and the second position represents the probability distribution of the duration of the transition.

Suppose $B = 20$*. Then, the uniformisation has transition matrix*

$$\tilde{P} = I + \frac{1}{B}Q = \begin{bmatrix} \frac{9}{10} & \frac{1}{10} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

This corresponds to the CTMC in Figure 7.

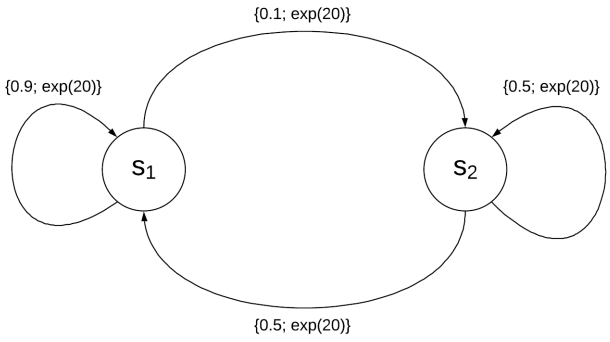

Figure 7: Uniformisation with $B = 20$. The first position in the braces represents the probability of the transition and the second position represents the probability distribution of the duration of the transition.

4.3.2 Continuous-time Markov decision process

In this section the uniformisation theory will be derived for a CTMDP. Moreover, the theory for CTMDPs will be immediately linked to the MDP model from the previous sections.
First of all, define D^{MR} and D^{MD} as the set of decision rules that are Markov random and Markov deterministic respectively. On top of that, define Π^{MR} and Π^{MD} as the sets of policies that only use Markov random decision rules and Markov deterministic decision rules respectively.
Let $d \in D^{MR}$ be a decision rule prescribing to take a specific action in a state, i.e. $d : S \to A$. The transition rates of the CTMDP of going from a state i to a state j under policy d are defined by

$$q_d(j|i) = q(j|i, d(i)) = t(j|i, d(i)), j \neq i,$$

where t is the transition rate function described in Section 4.2.3. The exit rate of state i under policy d is given by

$$q_d(i) = q(i|d(i)) = \sum_{k \neq i} q_d(k|i) = \sum_{k \neq i} t(k|i, d(i)).$$

Furthermore, define Q_d as the corresponding transition rate matrix under policy d as follows

$$Q_d(i,j) = \begin{cases} -q_d(i), & j = i, \\ q_d(j|i), & j \neq i. \end{cases}$$

Now, the uniformisation of the CTMDP can be defined. Again, for the uniformisation to be possible, all exit rates have to be uniformly bounded, i.e. there exists a constant $B < \infty$ such that

$$q_d(i) = \sum_{j \neq i} q_d(j|i) \leq B \ \forall i \in S \text{ and } \forall d \in D^{MR}. \tag{10}$$

Then, the transition probability matrix for the uniformisation under policy d is given by

$$\tilde{P}_d(i,j) = \tilde{p}_d(j|i) = \tilde{p}(j|i, d(i)) = \begin{cases} 1 - q_d(j)/B, & j = i, \\ q_d(j|i)/B, & j \neq i, \end{cases} \tag{11}$$

or

$$\tilde{P}_d = I + \frac{1}{B} Q_d.$$

In addition to the theory on uniformisation of a CTMC, the cost function has to be transformed; the uniformisation has cost function $\tilde{k}(i,a)$ defined below

$$\begin{aligned} \tilde{k}_d(i) = \tilde{k}(i, d(i)) &= k_d(i) \frac{q_d(i)}{B} \\ &= \frac{160}{n_c} \frac{q_d(i)}{B} \frac{\langle \bar{c}, \bar{q} \rangle}{q_d(i)} \\ &= \frac{160}{n_c} \langle \bar{c}, \bar{q} \rangle \frac{1}{B}. \end{aligned} \tag{12}$$

To be complete, the state space of the uniformisation is given by $\tilde{S} = S$ and the action space is given by $\tilde{A}_s = A_s$ for all $s \in \tilde{S}$. The newly derived MDP can be analysed using the same techniques as for discrete-time Markov decision processes, see Puterman (2014). To actually proof the relation between the CTMDP and its uniformisation, a relation between both gains of the MDPs will be derived.

Theorem 4.2. *Suppose* (10) *holds, and that* $\tilde{k}$ *and* $\tilde{p}$ *satisfy* (12) *and* (11). *Then,*

$$\tilde{g} = g/B. \tag{13}$$

Proof. Fix a $d \in D^{MR}$. From Theorem 4.1, it can be concluded that the infinitesimal generators of the CTMC corresponding to decision rule d and of its uniformisation are equal. Therefore, both processes have the same finite-dimensional distributions for each initial state distribution.

First, the cost structure of the uniformisation will be analysed. Introduce $\tilde{v}_s$ as the number of transitions from state s to itself in the uniformisation. In the uniformisation, the expected average cost of a sojourn time in a state s is given by

$$\begin{aligned} \mathbb{E}_s^d\big[\sum_{i=1}^{\tilde{v}_s} \tilde{k}_d(s)\big]/\mathbb{E}_s^d\big[\tilde{v}_s\big] &= \tilde{k}_d(s)\mathbb{E}_s^d\big[\sum_{i=1}^{\tilde{v}_s} 1\big]/\mathbb{E}_s^d\Big[\sum_{i=1}^{\tilde{v}_s} 1\Big] \\ &= \tilde{k}_d(s), \end{aligned}$$

where $\mathbb{E}_s^d$ is the expectation given current state s under decision rule d.

Applying

$$\tilde{k}_d(s) = k_d(s)\frac{\beta_d(s)}{B},$$

yields

$$\mathbb{E}_s^d\big[\sum_{i=1}^{\tilde{v}_s} \tilde{k}_d(s)\big]/\mathbb{E}_s^d\big[\tilde{v}_s\big] = \frac{q_d(s)}{B}k_d(s).$$

Secondly, define $\tau_d(s)$ as the sojourn time in state s using decision rule d in the CTMC corresponding to decision rule d. For this process, the expected average cost of a sojourn time state s yields

$$\mathbb{E}_s^d\big[k_d(s)\big]/\mathbb{E}_s^d\big[\tau_d(s)\big] = q_d(s)k_d(s).$$

Note that the expected average cost of a sojourn time in state s in the uniformisation is a factor $1/B$ times the expected average cost of the original problem.

Since the expected average cost during sojourn times in state s differs a factor $1/B$ and both processes have the same state occupancy distributions, the following holds

$$\tilde{g}^{d^\infty} = g^{d^\infty}/B,$$

where $\tilde{g}^{d^\infty}$ and g^{d^∞} represent the gain using the stationary policy d^∞, which is a policy that always uses decision rule d.
Since it holds for any $d \in D^{MR}$, it holds for the optimal policy

$$\tilde{g} = g/B.$$

□

The relation between g and $\tilde{g}$ can therefore be interpreted in the following way: the gain in the uniformisation captures the optimal expected average cost per transition, while the gain in the original CTMDP captures the optimal expected average cost per time unit.

4.4 Optimality equation

Puterman (2014) suggests a method to obtain the optimality equations for the CTMDP by formulating the CTMDP as a semi-Markov decision process and then applying the uniformisation transformation. The optimality equation becomes

$$0 = \min_{d \in D^{MR}} \{k_d(s) - g/q_d(s) + \sum_{j \in S} p_d(j|s)h(j) - h(s)\} \ \forall s \in S.$$

Using this optimality equation, the relation between the bias of the CTMDP and its uniformisation will be derived.

Theorem 4.3. *Suppose* (10) *holds, and that* $\tilde{k}$ *and* $\tilde{p}$ *satisfy* (12) *and* (11). *Then,*

$$\tilde{h} = h. \tag{14}$$

Proof. First of all, the relation between the probability distributions of both chains are defined through

$$\tilde{p}_d(j|s) = \begin{cases} 1 - \frac{[1-p_d(s|s)]q_d(s)}{B}, & j = s, \\ \frac{p_d(j|s)q_d(s)}{B}, & j \neq s, \end{cases} \tag{15}$$

see Puterman (2014).

The optimality equation will be rewritten using the relations between the uniformisation and original CTMDP. Moreover, the relation of Theorem 4.2 will also be used.

$$\begin{aligned} 0 &= \min_{d \in D^{MR}} \left\{ k_d(s) - \frac{g}{q_d(s)} + \sum_{j \in S} p_d(j|s)h(j) - h(s) \right\} \\ &= \min_{d \in D^{MR}} \left\{ \tilde{k}_d(s)\frac{B}{q_d(s)} - \tilde{g}\frac{B}{q_d(s)} + \sum_{\substack{j \in S \\ j \neq s}} p_d(j|s)h(j) + [p_d(s|s) - 1]h(s) \right\} \\ &= \min_{d \in D^{MR}} \left\{ \tilde{k}_d(s)\frac{B}{q_d(s)} - \tilde{g}\frac{B}{q_d(s)} + \sum_{\substack{j \in S \\ j \neq s}} p_d(j|s)h(j) + \frac{B}{q_d(s)}h(s) - \frac{B}{q_d(s)}h(s) + [p_d(s|s) - 1]h(s) \right\} \end{aligned}$$

Multiplying both sides of the equation by $q_d(s)/B$ results in

$$\begin{aligned} 0 &= \min_{d \in D^{MR}} \left\{ \tilde{k}_d(s) - \tilde{g} + \sum_{\substack{j \in S \\ j \neq s}} \frac{p_d(j|s)q_d(s)}{B}h(j) + h(s) - h(s) - \frac{[1 - p_d(s|s)]q_d(s)}{B}h(s) \right\} \\ &= \min_{d \in D^{MR}} \left\{ \tilde{k}_d(s) - \tilde{g} + \sum_{\substack{j \in S \\ j \neq s}} \frac{p_d(j|s)q_d(s)}{B}h(j) - h(s) + \left(1 - \frac{[1 - p_d(s|s)]q_d(s)}{B}\right)h(s) \right\} \end{aligned}$$

Using relation (15) gives

$$\begin{aligned} 0 &= \min_{d \in D^{MR}} \left\{ \tilde{k}_d(s) - \tilde{g} + \sum_{\substack{j \in S \\ j \neq s}} \tilde{p}_d(j|s)h(j) - h(s) - \tilde{p}_d(s|s)h(s) \right\} \\ &= \min_{d \in D^{MR}} \left\{ \tilde{k}_d(s) - \tilde{g} + \sum_{j \in S} \tilde{p}_d(j|s)h(j) - h(s) \right\} \end{aligned}$$

For $h(s) = \tilde{h}(s)$ for all $s \in S$, precisely the optimality equation of the corresponding DTMDP of the uniformisation is obtained

$$0 = \min_{d \in D^{MR}} \left\{ \tilde{k}_d(s) - \tilde{g} + \sum_{j \in S} \tilde{p}_d(j|s)\tilde{h}(j) - \tilde{h}(s) \right\} \tag{16}$$

Since it holds for all $s \in S$, the following holds

$$\tilde{h} = h.$$

□

The optimality equation in (16) can be solved with the tools developed for analyzing DTMDPs with the average cost criterion; the value iteration algorithm will be used.

5 Analysis

The completed MDP model can now be solved. This will be done with the well-known value iteration algorithm, which will be discussed in Section 5.1. To be able to execute the value iteration algorithm, the values for the cost function and transition rate function have to be determined. Section 5.2 shows how the discrete-event simulation can be used to determine estimates for the exponential parameters for different system configurations. Section 5.3 discusses how tasks can be ranked in order to determine the costs for letting tasks wait in the queue.
After applying the value iteration algorithm, the provided ϵ-optimal policy will be analysed in Section 5.4 in order to extract easy-to-implement rules. The rules are extracted from the data generated by simulating the CTMC corresponding to the ϵ-optimal provided by the MDP. These rules together form the heuristic policy, which is therefore MDP-based.
Finally, Section 5.5 discusses how the value iteration algorithm can be used to validate and to quantify the performance of the heuristic policy. Furthermore, this section shows how the heuristic policy and ϵ-optimal policy can be enlarged such that it can be tested for the warehouse of KLM Cargo; the discrete-event simulation quantifies the performance of those policies for the case study. Moreover, the MDP-based policies and the policies from literature can be compared by quantifying the performance of each policy with the discrete-event simulation model.

5.1 Value iteration

For solving the uniformisation of the CTMDP, the value iteration algorithm - also known as successive approximation - will be implemented in Python. This algorithm finds a stationary ϵ-optimal policy $(d_\epsilon)^\infty$ and an approximation to its value.

Value Iteration Algorithm
Adaptation of the algorithm described in Puterman (2014)

Step 1. Set $v^0 = \mathbf{0}$, specify $\epsilon > 0$, and set $n = 0$.
Step 2. For each $s \in \tilde{S}$, compute v^{n+1} by

$$v^{n+1}(s) = \min_{a \in \tilde{A}_s} \Big\{ \tilde{k}(s,a) + \sum_{j \in \tilde{S}} \tilde{p}(j|s,a) v^n(j) \Big\}.$$

Step 3. If

$$\max_{s \in \tilde{S}} \{v^{n+1}(s) - v^n(s)\} - \min_{s \in \tilde{S}} \{v^{n+1}(s) - v^n(s)\} < \epsilon,$$

go to step 4. Otherwise increment n by 1 and return to step 2.
Step 4. For each $s \in S$, choose

$$d_\epsilon(s) \in \arg\min_{a \in \tilde{A}_s} \Big\{ \tilde{k}(s,a) + \sum_{j \in \tilde{S}} \tilde{p}(j|s,a) v^n(j) \Big\},$$

and stop.

In Step 3 of the algorithm the so-called Odoni Bounds are used as stopping criterion (Odoni, 1969).

Alternatively, the accuracy of any guess for the gain $\hat{g}$ and any guess for the bias $\hat{h}$ can be verified using Theorem 5.1. Using this theorem, it can be verified for the guess of the gain whether it is within a distance ϵ of the optimal gain. The following notation is used in the theorem: $\bar{e} = (1, 1, ..., 1)^T$, $k_d = (k_d(s_1), k_d(s_2), ...)^T$ for each $s_1, s_2, ... \in S$, P_d is the transition probability matrix under decision rule d, P_π^t is the t-step transition probability matrix under policy π and v_n^π is the expected total reward under policy π after n decision epochs.

Theorem 5.1. *Suppose the following holds for given $\hat{g}$ and $\hat{h}$*

$$||B(\hat{g}, \hat{h})||_\infty \equiv || \min_{d \in D^{MR}} \{k_d - \hat{g}\bar{e} + (P_d - I)\hat{h}\}||_\infty \leq \epsilon \tag{17}$$

for a certain $\epsilon \geq 0$ and the optimal gain g exists. Then,

$$|g - \hat{g}| \leq \epsilon$$

Proof. Note that (17) implies

$$-\epsilon\bar{e} \leq B(\hat{g}, \hat{h}) \leq \epsilon\bar{e}.$$

The argumentation used below is an extension of the argumentation in Theorem 8.4.1 in Puterman (2014).

First of all, suppose $-\epsilon \leq B(\hat{g}, \hat{h}) \leq 0$, then the following holds

$$(\hat{g} - \epsilon)\bar{e} = \hat{g}\bar{e} - \epsilon\bar{e} \leq k_d + (P_d - I)\hat{h} \ \ \forall d \in D^{MR}. \tag{18}$$

Let $\pi = (d_1, d_2, ...) \in \Pi^{MR}$. Applying $d = d_1$ in (18) gives

$$(\hat{g} - \epsilon)\bar{e} \leq k_{d_1} + (P_{d_1} - I)\hat{h}.$$

Similarly, using (18) with $d = d_2$ and multiplying by P_{d_1} yields

$$(\hat{g} - \epsilon)\bar{e} = (\hat{g} - \epsilon)P_{d_1}\bar{e} \leq P_{d_1}k_{d_2} + (P_{d_1}P_{d_2} - P_{d_1})\hat{h}.$$

Repeating these arguments gives the following relation for $n \geq 2$

$$(\hat{g} - \epsilon)\bar{e} \leq P_{d_1}P_{d_2} \cdots P_{d_{n-1}}k_{d_n} + P_{d_1}P_{d_2} \cdots P_{d_{n-1}}(P_{d_n} - I)\hat{h}.$$

By summing these expressions over $n = N$ gives, for any $N \geq 1$ and any $\pi \in \Pi^{MR}$,

$$N(\hat{g} - \epsilon)\bar{e} \leq \sum_{t=1}^{N} P_\pi^{t-1}k_{d_n} + (P_\pi^N - I)\hat{h}.$$

Noting that $\sum_{t=1}^{N} P_\pi^{t-1}k_{d_n} = v_{N+1}^\pi$ holds by definition, yields

$$(\hat{g} - \epsilon)\bar{e} \leq \frac{1}{N}v_{N+1}^\pi + \frac{1}{N}(P_\pi^N - I)\hat{h}.$$

Taking the limit of $N \to \infty$ gives

$$\lim_{N\to\infty}(\hat{g} - \epsilon)\bar{e} \leq \lim_{N\to\infty}[\frac{1}{N}v_{N+1}^\pi + \frac{1}{N}(P_\pi^N - I)\hat{h}] = \lim_{N\to\infty}\frac{1}{N}v_{N+1}^\pi + \lim_{N\to\infty}\frac{1}{N}(P_\pi^N - I)\hat{h}.$$

Since $\hat{h} \in V$ and $P_\pi^N\hat{h} \in V$, where V is the space of bounded functions on S, it follows that $\lim_{N\to\infty}\frac{1}{N}(P_\pi^N - I)\hat{h} \to 0$ (Puterman, 2014). Applying this shows that

$$(\hat{g} - \epsilon)\bar{e} \leq \lim_{N\to\infty}\frac{1}{N}v_{N+1}^\pi + 0 = g\bar{e}.$$

Thus,

$$\hat{g} - g \leq \epsilon. \tag{19}$$

Secondly, assuming $0 \leq B(\hat{g}, \hat{h}) \leq \epsilon$ implies that there exist an optimal $d^* \in D^{MD}$ such that

$$(\hat{g} + \epsilon)\bar{e} = \hat{g}\bar{e} + \epsilon\bar{e} \geq k_{d^*} + (P_{d^*} - I)\hat{h},$$

see Puterman (2014). Now, applying the same argumentation as above by replacing all P_{d_n} by P_{d^*} yields

$$(\hat{g} + \epsilon)\bar{e} \geq \lim_{N\to\infty} \frac{1}{N} v_{N+1}^{(d^*)^\infty} = g\bar{e},$$

which establishes

$$\hat{g} - g \geq -\epsilon. \tag{20}$$

Combining the results of (19) and (20) gives

$$|\hat{g} - g| \leq \epsilon,$$

which completes the proof. □

An alternative stopping criterion for the value iteration algorithm can be constructed with Theorem 5.1. This criterion will be derived below.
Suppose the value iteration algorithm has been running for some time and gives us a new guess $\hat{g}^n$ for the gain at iteration n, for example

$$\hat{g}^n = \frac{\max_{s\in\tilde{S}}\{v^{n+1}(s) - v^n(s)\} + \min_{s\in\tilde{S}}\{v^{n+1}(s) - v^n(s)\}}{2}. \tag{21}$$

Then, a new a guess $\hat{h}^n$ can be made for the bias. Using the relation

$$v^n = (n-1)\cdot g + h + o(1),$$

see Puterman (2014), a guess for the bias can be constructed

$$\hat{h}^n = v^n - (n-1)\cdot \hat{g}^n. \tag{22}$$

Every new iteration, new guesses can be constructed using (21) and (22). When the guesses suffice the condition in (17), the value iteration can be stopped.
Nevertheless, the Odoni bounds are still used as stopping criterion of the value iteration algorithm in the implementation in Python.

5.2 Rates

The value iteration requires the parameters of the exponential distribution of all stochastic processes as input. The parameters for the service time of the ETV can easily be determined by the algorithm discussed in Section 3.3. However, determining values analytically for $\lambda_{0,0}, \lambda_{0,1}, \lambda_{1,0}, \lambda_{1,1}$ and λ_2 is almost impossible. Therefore, instead of determining the values analytically, they can be determined using the discrete-event simulation model created during the internship at KLM.

5.2.1 Statistical analysis

The simulation model uses the Replication/Deletion Approach; it makes n replications of length m using a warm-up period of length l. The warm-up period is determined using Welch's Approach proposed by Welch (1981). During these replications, the model gathers the desired observations. After finishing a replication, the mean of these observations is determined. Define the observation i in replication j as Y_{ji}. Then, the mean of these observations in replication j is defined as

$$X_j = \frac{\sum_{i=l+1}^{m} Y_{ji}}{m-l} \text{ for } j = 1, 2, ..., n.$$

After making n replications, the model determines the sample mean of the observed means of all replications and it constructs the corresponding confidence interval. The sample mean $\bar{X}(n)$ of n replications is calculated by

$$\bar{X}(n) = \sum_{j=1}^{n} X_j.$$

To construct a confidence interval for the sample mean, the sample variance $S^2(n)$ of n replications has to be determined using the following relation

$$S^2(n) = \frac{1}{n-1} \sum_{i=1}^{n} [X_i - \bar{X}(n)]^2.$$

The Replication/Deletion Approach guarantees that the X_i's are independent identical distributed (IID) random variables. Therefore, $\bar{X}(n)$ is an approximately unbiased point estimator for the mean and an approximate $100(1-\alpha)$ per cent confidence interval for the steady-state mean and is given by

$$\bar{X}(n) \pm t_{n-1,1-\alpha/2} \sqrt{\frac{S^2(n)}{n}},$$

where $t_{n-1,1-\alpha/2}$ is the upper $1-\alpha/2$ critical point for the t distribution with $n-1$ degrees of freedom (Law, 2015).

In all cases, a simulation of 50 replications of 2 years and 20 weeks is made using a warm-up period of 20 weeks. During these simulations at most ten different performance measures are observed, because of Bonferroni's Inequality (Bonferroni, 1936). The inequality gives an lower bound of the probability that all performance measures are captured in their confidence intervals simultaneously. Suppose that performance measures μ_s for $s = 1, ..., k$ have to be determined. For each of these performance measures a $100(1-\alpha_s)$ per cent confidence interval J_s is constructed. Then, the lower bound is given by

$$\mathbb{P}\big(\mu \in J_s \text{ for all } s = 1, 2, ..., k\big) \geq 1 - \sum_{s=1}^{k} \alpha_s.$$

Therefore, the simulation model measures at most ten different performance measures while always determining 99% per cent confidence intervals for each performance measure. This guarantees that the overall probability of containing all means in their calculated confidence intervals is at least 90%.

5.2.2 Determining parameters

With information about the amount and type of containers that enter the sorting system, it can be simulated which type, what amount and when containers are transported from one component to the other. To determine the arrival rate of each type of container at the ETV, only the flows that go directly to the ETV are of interest. This point of view results in a new schematic overview of the sorter, which is visualised in Figure 8. The overview contains only detailed information about the processes that concern the ETV directly; all other processes are captured in so-called black boxes, i.e. an object of which the input and output can be observed without knowing how it works.

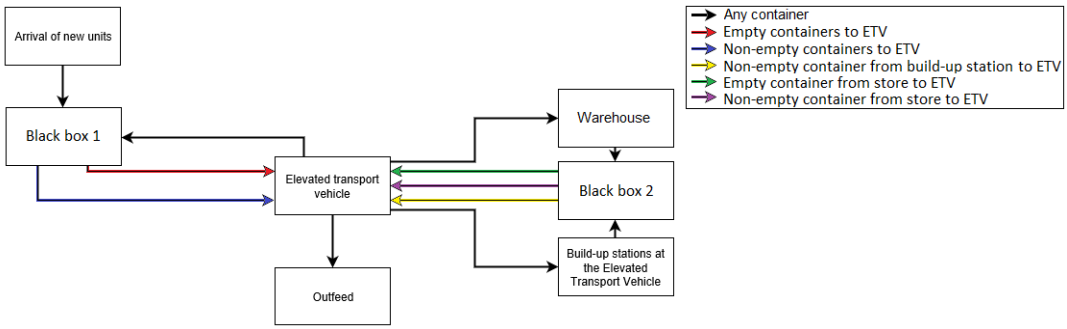

Figure 8: A schematic overview of the flow of containers focused on the flows that are of direct interest for the ETV; the other flows are captured in the black boxes.

Black box 1 calculates how many arriving containers at the black box go to the ETV as empty or non-empty container; black box 2 calculates the same for empty containers that go from the warehouse to the ETV and for non-empty containers from that go from a build-up station to the ETV. Additionally, black box 2 determines how long non-empty containers stay in the warehouse on average. Each of the black boxes calculates an estimate for the exponential parameters in the following way:

1. The simulation model simulates 50 replications of 2 years and 20 weeks using a warm-up period of 20 weeks.
2. During each of the fifty replications, it determines the average flow of containers per day via for the red, blue, yellow and green arrows in Figure 8 separately. For the purple coloured arrow, it determines the average staying time of a non-empty container in the warehouse.
3. The sample means and their confidence intervals are determined.
4. The sample means are used as estimates for the exponential parameters $\lambda_{0,0}, \lambda_{0,1}, \lambda_{1,0}, \lambda_{1,1}$ and λ_2. The estimates are indicated by $\bar{\lambda}_{0,0}, \bar{\lambda}_{0,1}, \bar{\lambda}_{1,0}, \bar{\lambda}_{1,1}$ and $\bar{\lambda}_2$.

The estimates are determined for different workload scenarios; the scenarios are called 'No Peak', 'Peak' and 'High Peak', and are ranked from a low to high workload. A higher workload means that there are more containers flowing into and out of the system than when the workload is lower. The results are given in Table 3.

Table 3: The estimates and confidence intervals of the exponential parameters for the three workload scenarios No Peak, Peak and High Peak.

	No Peak	Confidence interval	Peak	Confidence interval	High Peak	Confidence interval
$\bar{\lambda}_{0,0}$ (/day)	144.38	[144.23; 144.53]	178.47	[178.27; 178.67]	220.79	[220.64; 220.94]
$\bar{\lambda}_{0,1}$ (/day)	33.31	[33.17; 33.45]	35.95	[35.76; 36.13]	31.91	[31.77; 32.05]
$\bar{\lambda}_{1,0}$ (/day)	42.75	[42.61; 42.88]	78.09	[77.85; 78.34]	131.10	[130.80; 131.39]
$\bar{\lambda}_{1,1}$ (/s)	1/14,500.03	[1/14,541.28; 1/14,458.78]	1/11,851.39	[1/11,876.65; 1/11,826.13]	1/10,689.92	[1/10,710.30; 1/10,669.53]
$\bar{\lambda}_2$ (/day)	111.08	[110.97; 111.19]	142.56	[142.46; 142.66]	188.88	[188.78; 188.98]

To use these estimates in the MDP model, they have to be scaled in order to match the transformed problem discussed in Section 4.1.1. Namely, one container in the transformed problem represents a batch of containers in the original problem. Therefore, the batch size is used as scale factor for the arrival rates. Additionally, for simplicity it is assumed that the staying time of a batch of non-empty containers is equal to the staying time of one non-empty container.
Define, $\hat{\lambda}_{0,0}$, $\hat{\lambda}_{0,1}$, $\hat{\lambda}_{1,0}$, $\hat{\lambda}_{1,1}$ and $\hat{\lambda}_2$ as the estimates of the parameters that can be used in the MDP model. The relations between the estimates of the simulation model and the estimates for the MDP model are given by

$$\begin{aligned}
\hat{\lambda}_{0,0} &= \frac{n_z}{160}\bar{\lambda}_{0,0}, \\
\hat{\lambda}_{0,1} &= \frac{n_z}{160}\bar{\lambda}_{0,1}, \\
\hat{\lambda}_{1,0} &= \frac{n_z}{160}\bar{\lambda}_{1,0}, \\
\hat{\lambda}_{1,1} &= \bar{\lambda}_{1,1}, \\
\hat{\lambda}_2 &= \frac{n_z}{160}\bar{\lambda}_2.
\end{aligned}$$

Similarly, the service time of the original problem has to be scaled for the transformed problem. Using the batch size as scaling factor of the service time yields

$$\hat{\mu}(x_1, x_2) = \frac{n_z}{160}\mu(x_1, x_2),$$

where μ is the service time function discussed in Section 4.2.3 and Appendix B.

The scaled parameters $\hat{\lambda}_{0,0}, \hat{\lambda}_{0,1}, \hat{\lambda}_{1,0}, \hat{\lambda}_{1,1}, \hat{\lambda}_2$ and $\hat{\mu}(x_1, x_2)$ are plugged into the transition rate function of Section 4.2.3.

5.3 Costs

The importance of a task can be captured by assigning certain costs for letting it wait in the queue; more important tasks will be given a higher waiting cost than less important tasks. Therefore, a hierarchy of the five different tasks has to be made. This hierarchy is based on the loss a delay in the sorting process of a tasks will cause.

The task with the highest priority is removing non-empty containers from the warehouse. Such a container is ready to be shipped to aircraft it is assigned to. If the container is sent to the aircraft too late, it may delay or miss the flight; this comes with a penalty and damages the confidence of clients.
The second most important tasks are removing non-empty containers from a build-up station and placing empty containers at a build-up station. When a delay occurs in removing a non-empty container from a build-up station, the station is not able to process any of the sorted packages that arrive at the station. This will result in a backlog of packages that have to be

loaded onto a container. Similarly, when the process of placing an empty container at a build-up station is delayed, the build-up station will not have a container to load the sorted packages onto.
Finally, the least important tasks are placing containers in the warehouse. A delay in this process will have minor impact on other processes. The only problem will arise when the queue of the ETV has reached its capacity; this results in blockage of the conveyors of the sorting system.
To conclude, the hierarchy discussed above is represented by the values for the cost coefficients; the coefficients can be found in Table 4.

Table 4: The cost coefficients for each task.

	$c_{0,0}$	$c_{0,1}$	$c_{1,0}$	$c_{1,1}$	c_2
value	1.0	1.1	1.0	1.2	1.1

5.4 Heuristic

Solving the CTMDP using the value iteration algorithm guarantees to find an ϵ-optimal policy $(d_\epsilon)^\infty$. This policy prescribes a specific decision rule for each state. This results in a lot of rules due to the very large state space. Therefore, a heuristic policy will be extracted from the ϵ-optimal policy that tries to mimic the logic of the ϵ-optimal policy as close as possible with less rules. The idea is to find general rules that capture the logic of the ϵ-optimal policy. Therefore, the heuristic policy is MDP-based.
To find this heuristic policy, the CTMDP is solved with the value iteration algorithm using the High Peak scenario and setting $\epsilon = 0.05$; it converges at iteration 2453 with gain $g = 11706.48$. The High Peak scenario is used, because it has the highest workload; in a high workload scenario a bad decision may have a bigger impact than in a low workload scenario.
To analyse the ϵ-optimal policy, the CTMC that corresponds with this policy will be simulated for 250,000 days. The simulation keeps track of how many times a certain state is visited and which action is chosen in that state. The general rules that from the heuristic policy can be extracted from these results. The following sections discuss the analysis of the CTMC simulation results.
The results will be analysed in the light of the transformed problem. This means that the term 'storage location' is used rather than 'compartment'. However, the reader should keep in mind that the term 'storage location' in the transformed problem is equivalent to 'compartment' in the original problem. Similarly, filling storage location c with a container of type X (transformed problem) is equivalent to allocating compartment c to container type X (original problem).

5.4.1 Action $a_{1,0,c}$

To get a clear overview of in which states which variant of action $a_{1,0,c}$ is chosen, states are divided into categories. These categories will be analysed instead of the individual states. For action $a_{1,0,c}$ the states are divided into categories $[u, v, w]$ of the form

$$[u, v, w] = [\sum_{i=1}^{2} \mathbb{1}\{w_i = 0\}, \mathbb{1}\{w_3 = 0\}, \sum_{i=4}^{5} \mathbb{1}\{w_i = 0\}]. \tag{23}$$

So, u can be interpreted as the number of empty storage locations of the first two locations, v indicates whether the third storage location is empty or not and w can be interpreted as the number of empty storage locations of the last two storage locations. Aggregating all simulation results of states that belong to the same category, gives Table 5.

Table 5: Results of simulating the CTMC that corresponds to the ϵ-optimal policy for 250,000 days. The table shows how many times the action $a_{1,0,c}$ for any $c \in I$ in a certain category of states has been chosen during the simulation. The categories are defined in (23).

	$a_{1,0,1}$	$a_{1,0,2}$	$a_{1,0,3}$	$a_{1,0,4}$	$a_{1,0,5}$
$[0,0,0]$	0	0	0	0	0
$[0,0,1]$	0	0	0	5,226	6,560
$[0,0,2]$	0	0	0	8,430	0
$[0,1,0]$	0	0	13,444	0	0
$[0,1,1]$	0	0	11,378	0	0
$[0,1,2]$	0	0	14,621	0	0
$[1,0,0]$	82,703	24,824	0	0	0
$[1,0,1]$	24,216	6,217	0	0	0
$[1,0,2]$	17,320	3,685	0	0	0
$[1,1,0]$	30,838	9,480	0	0	0
$[1,1,1]$	22,510	9,922	0	0	0
$[1,1,2]$	28,451	17,286	0	0	0
$[2,0,0]$	39,307	0	0	0	0
$[2,0,1]$	18,697	0	0	0	0
$[2,0,2]$	14,538	0	0	0	0
$[2,1,0]$	38,784	0	0	0	0
$[2,1,1]$	45,855	0	0	0	0
$[2,1,2]$	67,253	0	0	0	0

The table shows that a non-empty container is always placed at the first open storage location. For example, whenever both the first and second storage location are empty, the container is always placed at the first open storage location. Similarly, whenever only the first storage location or only the second storage location is empty, the non-empty container is placed at that location that is empty. The same structure can be observed for all other categories.

5.4.2 Action $a_{0,0,c}$

To analyse action $a_{0,0,c}$, the following categories are used

$$[u, v, w, x, y] = [w_1, w_2, w_3, w_4, w_5]. \tag{24}$$

In this case, u indicates whether storage location 1 is empty or contains a non-empty or an empty container. Similarly, the letters v, w, x and y indicate the same for storage locations 2, 3, 4 and 5 respectively.

Table 6: Part of the results of simulating the CTMC that corresponds to the ϵ-optimal policy for 250,000 days. The table shows how many times the action $a_{0,0,c}$ for any $c \in I$ in a certain category of states has been chosen during the simulation. The categories are defined in (24).

	$a_{0,0,1}$	$a_{0,0,2}$	$a_{0,0,3}$	$a_{0,0,4}$	$a_{0,0,5}$
$[2,0,1,0,1]$	0	2,418	0	0	0
$[2,0,1,0,2]$	0	184	0	0	0
$[2,0,1,1,0]$	0	107	0	0	1,990
$[2,1,0,1,0]$	0	0	0	0	4,414
$[2,1,0,2,0]$	0	0	518	0	0
$[2,1,1,0,0]$	0	0	0	0	2,627
$[2,2,0,1,0]$	0	0	1,588	0	0
$[2,2,0,2,0]$	0	0	1,055	0	0
$[2,2,1,0,0]$	0	0	0	2,440	0
$[1,0,0,0,0]$	0	74	41	0	0
$[1,0,0,0,1]$	0	0	167	0	0
$[1,0,0,0,2]$	0	10	1	4	0
$[1,0,0,1,0]$	0	0	108	0	0
$[1,0,0,1,1]$	0	0	1,056	0	0
$[1,0,0,1,2]$	0	0	19	0	0
$[0,1,0,0,0]$	3,487	0	12,153	0	0
$[0,1,0,1,1]$	816	0	7,570	0	0
$[0,1,0,1,2]$	76	0	438	0	0
$[0,1,0,2,0]$	35	0	602	0	0
$[0,1,1,0,0]$	258	0	0	5,050	0
$[0,1,1,1,0]$	129	0	0	0	2,616
$[0,1,2,1,0]$	94	0	0	0	1,232
$[0,1,2,2,0]$	42	0	0	0	649

From the full results - shown in tables 20, 21 and 22 in Appendix D - the following observations can be made:

1. Suppose $w_1 = 2$ and $w_2 = 0$. If $\mathbb{1}\{w_3 = 1\} + \mathbb{1}\{w_4 = 1\} \leq 1$, the first open storage location is used. Otherwise, the second open storage location is used.
2. Suppose $w_1 = 2$ and $w_2 = 1$. If $\mathbb{1}\{w_3 = 2\} + \mathbb{1}\{w_4 = 2\} \geq 1$, the first open storage location is used. Otherwise, the second open storage location is used.
3. Suppose $w_1 = 2$ and $w_2 = 2$, then the first open storage location is used.
4. Suppose $w_1 = 1$. If $\mathbb{1}\{w_2 = 2\} + \mathbb{1}\{w_3 = 2\} + \mathbb{1}\{w_4 = 2\} \geq 2$, the first open storage location is used. Otherwise, the second open storage location is used.
5. Suppose $w_1 = 0$ and $w_2 = 1$, then the second open storage location is used.
6. For all other cases where $w_1 = 0$, no pattern can be observed.

To illustrate these observations, multiple examples from the results are given in Table 6. Observations 1 up to and including 5 can be summarised by the following two goals:

1. At least one of the first two storage locations has to contain a non-empty container.
2. At least two of the first four storage locations have to contain a non-empty container.

These goals for positioning non-empty containers gives rise to the following rules for storing empty containers. In general, it is preferred to use the first open storage location to store an empty container. However, when storing an empty container in the first open storage location causes a violation of one of the goals for positioning non-empty containers, the second open storage location - when available - is used to store the empty containers.

Since the results for $w_1 = 0$ show no clear pattern, different categories will be studied for this case. In the following analyses the categories defined in (23) are extended with some information about the queue:

$$[[u, v, w, x, y], z] = [[\mathbb{1}\{w_1 = 0\}, \mathbb{1}\{w_2 = 0\}, \mathbb{1}\{w_3 = 0\}, \mathbb{1}\{w_4 = 0\}, \mathbb{1}\{w_5 = 0\}], \mathbb{1}\{q_{0,0} + q_{0,1} + q_{1,0} + q_{0,1} + q_2 \geq 2\}]. \tag{25}$$

In this category, z indicates whether there are two or more tasks currently waiting in the queue. The results are shown in Table 7.

Table 7: Results of simulating the CTMC that corresponds to the ϵ-optimal policy for 250,000 days. The table shows how many times the action $a_{0,0,c}$ for any $c \in I$ in a certain category of states has been chosen during the simulation. The categories are defined in (25).

	$a_{0,0,1}$	$a_{0,0,2}$	$a_{0,0,3}$	$a_{0,0,4}$	$a_{0,0,5}$
$[[1,0,0,0,1],0]$	1,614	0	0	0	6,237
$[[1,0,0,0,1],1]$	1,601	0	0	0	0
$[[1,0,0,1,0],0]$	1,460	0	0	4,160	0
$[[1,0,0,1,0],1]$	1,964	0	0	0	0
$[[1,0,0,1,1],0]$	1,793	0	0	8,965	0
$[[1,0,0,1,1],1]$	2,086	0	0	0	0
$[[1,0,1,0,0],0]$	2,896	0	9,994	0	0
$[[1,0,1,0,0],1]$	3,322	0	6	0	0
$[[1,0,1,0,1],0]$	439	0	5,012	0	0
$[[1,0,1,0,1],1]$	1,136	0	2	0	0
$[[1,0,1,1,0],0]$	1,304	0	5,322	320	0
$[[1,0,1,1,0],1]$	2,179	0	0	5	0
$[[1,0,1,1,1],0]$	0	0	16,482	0	0
$[[1,0,1,1,1],1]$	5,741	0	0	0	0
$[[1,1,0,0,0],0]$	9,685	4,938	0	0	0

	$a_{0,0,1}$	$a_{0,0,2}$	$a_{0,0,3}$	$a_{0,0,4}$	$a_{0,0,5}$
$[[1,1,0,0,0],1]$	3,830	35	0	0	0
$[[1,1,0,0,1],0]$	1,735	2,390	0	0	0
$[[1,1,0,0,1],1]$	1,596	14	0	0	0
$[[1,1,0,1,0],0]$	1,668	1,788	0	0	0
$[[1,1,0,1,0],1]$	1,782	28	0	0	0
$[[1,1,0,1,1],0]$	0	6,179	0	0	0
$[[1,1,0,1,1],1]$	2,427	42	0	0	0
$[[1,1,1,0,0],0]$	2,391	9,999	0	0	0
$[[1,1,1,0,0],1]$	6,377	13	0	0	0
$[[1,1,1,0,1],0]$	0	5,491	0	0	0
$[[1,1,1,0,1],1]$	4,734	6	10	0	0
$[[1,1,1,1,0],0]$	0	9,560	0	0	0
$[[1,1,1,1,0],1]$	8,790	6	10	0	0
$[[1,1,1,1,1],0]$	0	23,321	0	0	0
$[[1,1,1,1,1],1]$	15,728	0	0	0	0

From these results, a good rule of thumb can be derived for using the second open storage location instead of the first open storage location when $w_1 = 0$: whenever there are two or more tasks currently waiting in the queue, the first open storage location is used. Otherwise, the second open storage location is used.

5.4.3 Action $a_{0,1,c}$

For the analysis of removing an empty container from storage location c, the following categories are used

$$[u, v, w] = [\sum_{i=1}^{2} \mathbb{1}\{w_i = 1\}, \mathbb{1}\{w_3 = 1\}, \sum_{i=4}^{5} \mathbb{1}\{w_i = 1\}]. \tag{26}$$

In this case, u indicates how many of the first two storage locations contain an empty container, v indicates whether storage location 3 contains an empty container and w indicates how many of the last two storage locations contain an empty container.

Table 8: Results of simulating the CTMC that corresponds to the ϵ-optimal policy for 250,000 days. The table shows how many times the action $a_{0,1,c}$ for any $c \in I$ in a certain category of states has been chosen during the simulation. The categories are defined in (26).

	$a_{0,1,1}$	$a_{0,1,2}$	$a_{0,1,3}$	$a_{0,1,4}$	$a_{0,1,5}$
$[0,0,0]$	0	0	0	0	0
$[0,0,1]$	0	0	0	4,011	6,333
$[0,0,2]$	0	0	0	11,262	77
$[0,1,0]$	0	0	5,111	0	0
$[0,1,1]$	0	0	6,749	158	52
$[0,1,2]$	0	0	16,520	47	161
$[1,0,0]$	984	10,870	0	0	0
$[1,0,1]$	18,127	10,080	0	611	84
$[1,0,2]$	9,335	8,680	0	214	324
$[1,1,0]$	1,356	2,371	1,000	0	0
$[1,1,1]$	1,561	2,095	1,343	184	86
$[1,1,2]$	4,446	8,851	7,177	179	141
$[2,0,0]$	1,511	5	0	0	0
$[2,0,1]$	1,598	12	0	0	0
$[2,0,2]$	2,157	13	0	1	1
$[2,1,0]$	477	1	0	0	0
$[2,1,1]$	854	6	0	0	0
$[2,1,2]$	4,870	127	27	1	1

It can easily be observed from Table 8 that in almost all cases the first empty container near the infeed of the ETV will be removed. This holds for all cases except the categories $[1,1,\cdot]$. In these categories the choice of removing an empty container from storage location c is evenly spread between $a_{0,1,1}$, $a_{0,1,2}$ and $a_{0,1,3}$. Therefore, these categories will be investigated a little further by extending the category with the information of the current position of the ETV:

$$[[u, v, w], y] = [[\sum_{i=1}^{2} \mathbb{1}\{w_i = 1\}, \mathbb{1}\{w_3 = 1\}, \sum_{i=4}^{5} \mathbb{1}\{w_i = 1\}], x]. \tag{27}$$

Table 9: Results of simulating the CTMC that corresponds to the ϵ-optimal policy for 250,000 days. The table shows how many times the action $a_{0,1,c}$ for any $c \in I$ in a certain category of states has been chosen during the simulation. The categories are defined in (27).

	$a_{0,1,1}$	$a_{0,1,2}$	$a_{0,1,3}$	$a_{0,1,4}$	$a_{0,1,5}$
$[[1,1,0],0]$	65	838	0	0	0
$[[1,1,0],1]$	1,190	809	0	0	0
$[[1,1,0],2]$	71	562	0	0	0
$[[1,1,0],3]$	19	41	993	0	0
$[[1,1,0],4]$	9	91	6	0	0
$[[1,1,0],5]$	2	30	1	0	0
$[[1,1,1],0]$	205	716	0	0	0
$[[1,1,1],1]$	1,009	972	0	0	0
$[[1,1,1],2]$	251	218	0	0	0
$[[1,1,1],3]$	72	43	1,234	0	0
$[[1,1,1],4]$	15	40	77	184	0
$[[1,1,1],5]$	9	106	32	0	86
$[[1,1,2],0]$	679	476	0	0	0
$[[1,1,2],1]$	2,462	5,674	0	0	0
$[[1,1,2],2]$	1,083	2,317	0	0	0
$[[1,1,2],3]$	205	339	7,162	0	0
$[[1,1,2],4]$	8	14	0	179	0
$[[1,1,2],5]$	9	31	15	0	141

In these cases, when the current position of the ETV is between 0 and 2, the empty container at the the front of the warehouse will be removed. If the ETV is currently at storage location 3, it will remove the empty container at storage location 3. Finally, if it stands at storage location 4 or 5 and that storage location contains an empty container it will remove the container from storage location 4 or 5 respectively. In the case the storage location at which the ETV currently stands contains no empty container, it will remove the empty containers from the first storage location from the infeed that contains an empty containers.

5.4.4 Actions a_2 and $a_{2,c}$

In this section the actions a_2 and $a_{2,c}$ are discussed.
First of all, a rule will be extracted for when to choose a_2 and when to choose $a_{2,c}$ for any $c \in I$, whenever there is an empty container waiting in the queue. The following categories are used

$$[[u,v],w] = [[\sum_{i=1}^{3} \mathbb{1}\{w_i = 1\}, \sum_{i=4}^{5} \mathbb{1}\{w_i = 1\}], \mathbb{1}\{q_{0,0} = 1\}]. \tag{28}$$

The interpretation of the category is as follows: u represents the number of storage locations of the first three storage locations that contain an empty container, v represents the number of storage locations of the last two storage locations that contain an empty container and w indicates whether there is an empty container in the queue or not.

Table 10: Results of simulating the CTMC that corresponds to the ϵ-optimal policy for 250,000 days. In the table, it can be found how many times the actions $a_{2,c}$ for any $c \in I$ and a_2 in a certain category of states has been chosen during the simulation. The categories are defined in (28).

	$a_{2,c}; c \in I$	a_2
$[[0,0],1]$	0	55,538
$[[0,1],1]$	0	16,097
$[[0,2],1]$	0	17,865
$[[1,0],1]$	144	11,990
$[[1,1],1]$	232	17,237
$[[1,2],1]$	297	29,855
$[[2,0],1]$	121	3,510
$[[2,1],1]$	867	4,995
$[[2,2],1]$	7,022	23,649
$[[3,0],1]$	136	76
$[[3,1],1]$	438	272
$[[3,2],1]$	10,313	0

For almost all categories holds that a_2 is preferred over $a_{2,c}$. However, for the categories $[[3,\cdot],1]$ action $a_{2,c}$ is preferred, especially for $[[3,2],1]$. For the latter, it is quite obvious: there is no more space to store non-empty containers. Since these containers are the most valuable, space in the warehouse has to be created in order to store them. For the other categories, a similar argument can be used; since the front positions of the warehouse are filled, space has to be created for the non-empty containers. Therefore, the rule will be that a_2 is chosen whenever possible except when the current state belongs to the category $[[3,\cdot],1]$.

Secondly, action $a_{2,c}$ will be analysed. For this action, the categories defined in (26) are used

$$[u,v,w] = [\sum_{i=1}^{2} \mathbb{1}\{w_i = 1\}, \mathbb{1}\{w_3 = 1\}, \sum_{i=4}^{5} \mathbb{1}\{w_i = 1\}]$$

Table 11: Results of simulating the CTMC that corresponds to the ϵ-optimal policy for 250,000 days. The table shows how many times the action $a_{2,c}$ for any $c \in I$ in a certain category of states has been chosen during the simulation. The categories are defined in (26).

	$a_{2,1}$	$a_{2,2}$	$a_{2,3}$	$a_{2,4}$	$a_{2,5}$
$[0,0,0]$	0	0	0	0	0
$[0,0,1]$	0	0	0	12,571	22,295
$[0,0,2]$	0	0	0	23,089	0
$[0,1,0]$	0	0	11,693	0	0
$[0,1,1]$	0	0	10,698	0	0
$[0,1,2]$	0	0	18,448	0	0
$[1,0,0]$	26,588	41,870	0	0	0
$[1,0,1]$	9,048	33,611	0	0	0
$[1,0,2]$	4,690	26,269	0	0	0
$[1,1,0]$	1,847	11,706	0	0	0
$[1,1,1]$	3,022	12,928	0	0	0
$[1,1,2]$	10,641	34,079	0	0	0
$[2,0,0]$	3,466	0	0	0	0
$[2,0,1]$	1,302	0	0	0	0
$[2,0,2]$	3,170	0	0	0	0
$[2,1,0]$	464	0	0	0	0
$[2,1,1]$	1,487	0	0	0	0
$[2,1,2]$	22,305	0	0	0	0

Table 11 shows the same pattern as Table 5: the empty container the closest to the infeed will be removed.

5.4.5 Action $a_{3,c}$

For standing still at a certain storage location, the following categories are used

$$[u, v, w] = [\sum_{i=1}^{2} \mathbb{1}\{w_i > 0\}, \mathbb{1}\{w_3 > 0\}, \sum_{i=4}^{5} \mathbb{1}\{w_i > 0\}]. \tag{29}$$

In this case, u indicates how many of the first two storage locations are filled, v indicates whether storage location 3 is filled and w indicates how many of the last two storage locations are filled.

Table 12: Results of simulating the CTMC that corresponds to the ϵ-optimal policy for 250,000 days. In the table, it can be found how many times the action $a_{3,c}$ for any $c \in I$ in a certain category of states has been chosen during the simulation. The categories are defined in (29).

	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$
$[0,0,0]$	195,971	9,796	0	0	0	0
$[0,0,1]$	51,225	4,105	0	0	10,938	7,950
$[0,0,2]$	42,941	3,888	0	0	10,748	1,649
$[0,1,0]$	16,268	0	0	20,589	0	0
$[0,1,1]$	16,201	0	0	21,732	1,387	696
$[0,1,2]$	44,911	0	0	14,724	10,655	999
$[1,0,0]$	119,585	170,287	43,797	0	0	0
$[1,0,1]$	79,264	81,193	17,170	0	5,720	4,875
$[1,0,2]$	61,993	71,803	15,526	0	11,795	3,649
$[1,1,0]$	55,120	31,460	50,206	14,953	0	0
$[1,1,1]$	47,380	48,591	40,833	43,788	3,581	1,897
$[1,1,2]$	0	112,280	152,284	67,933	19,720	3,593
$[2,0,0]$	0	210,141	4,489	0	0	0
$[2,0,1]$	0	142,027	15,138	0	3	151
$[2,0,2]$	0	131,087	20,331	2,558	1,202	412
$[2,1,0]$	0	117,951	17,278	10	0	0
$[2,1,1]$	0	122,083	55,984	9,532	3	280
$[2,1,2]$	0	876,698	262,186	83,066	21,516	3,732

First of all, as a rule of thumb, the ETV stands still at the infeed when the first three storage locations are empty. Additionally, when the first two storage locations are empty but the third is not, the ETV stands still at storage location 3.
For all other categories, no clear pattern can be found. However, because non-empty containers have the highest priority and highest cost weight, it is more important to anticipate on requests for removing non-empty containers than for empty containers. Therefore, the influence of the positions of non-empty containers and empty containers on the decision where to stand still will be analysed as two separate cases. In the first case, the warehouse contains only empty containers. In the second case, the warehouse contains only non-empty containers. For both cases, rules will be derived separately. Since in general non-empty and empty containers are stored in the warehouse simultaneously, it is assumed that the rules for non-empty containers are leading when there is at least one non-empty container in the warehouse.
To be able to analyse these cases, the necessary information will be obtained by using part of the information of the following category

$$[u, v, w, x, y] = [w_1, w_2, w_3, w_4, w_5]. \tag{30}$$

In this category, u, v, w, x and y indicate what kind of container is stored in the storage location.

Table 13: Results of simulating the CTMC that corresponds to the ϵ-optimal policy for 250,000 days. The table shows how many times the action $a_{3,c}$ for any $c \in I$ in a certain category of states has been chosen during the simulation. The categories are defined in (30).

(a) The part of the results where the warehouse only contains empty containers.

	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$
[1, 0, 0, 0, 0]	255	0	0	0	0	0
[0, 1, 0, 0, 0]	119,330	0	761	0	0	0
[0, 0, 1, 0, 0]	16,268	0	0	0	0	0
[0, 0, 0, 1, 0]	16,633	1,357	0	0	0	0
[0, 0, 0, 0, 1]	34,592	2,748	0	0	0	0
[1, 1, 0, 0, 0]	0	64	1	0	0	0
[1, 0, 1, 0, 0]	0	692	42	0	0	0
[1, 0, 0, 1, 0]	0	681	28	0	0	0
[1, 0, 0, 0, 1]	0	1,391	35	0	0	0
[0, 1, 1, 0, 0]	55,120	0	9,642	0	0	0
[0, 1, 0, 1, 0]	33,505	0	514	0	0	0
[0, 1, 0, 0, 1]	45,759	0	506	0	0	0
[0, 0, 1, 1, 0]	8,375	0	0	0	0	0
[0, 0, 1, 0, 1]	7,826	0	0	0	0	0
[0, 0, 0, 1, 1]	42,941	3,888	0	0	0	0
[1, 1, 1, 0, 0]	0	76	0	0	0	0
[1, 1, 0, 1, 0]	0	96	3	0	0	0
[1, 1, 0, 0, 1]	0	185	5	0	0	0
[1, 0, 1, 1, 0]	0	7,123	184	0	0	0
[1, 0, 1, 0, 1]	0	7,030	187	0	0	0
[1, 0, 0, 1, 1]	0	11,474	132	0	0	0
[0, 1, 1, 1, 0]	26,768	0	6,220	3,616	0	0
[0, 1, 1, 0, 1]	20,612	0	3,704	0	0	0
[0, 1, 0, 1, 1]	61,993	0	1,046	0	0	0
[0, 0, 1, 1, 1]	44,911	0	0	0	0	0
[1, 1, 1, 1, 0]	0	632	0	0	0	0
[1, 1, 1, 0, 1]	0	1,125	10	13	0	0
[1, 1, 0, 1, 1]	0	3,527	28	9	0	0
[1, 0, 1, 1, 1]	0	50,231	2,010	785	0	0
[0, 1, 1, 1, 1]	0	0	105,152	3,248	1,794	0
[1, 1, 1, 1, 1]	0	207,228	906	257	0	0

(b) The part of the results where the warehouse only contains non-empty containers.

	$a_{3,0}$	$a_{3,1}$	$a_{3,2}$	$a_{3,3}$	$a_{3,4}$	$a_{3,5}$
[2, 0, 0, 0, 0]	0	170,287	0	0	0	0
[0, 2, 0, 0, 0]	0	0	43,036	0	0	0
[0, 0, 2, 0, 0]	0	0	0	20,589	0	0
[0, 0, 0, 2, 0]	0	0	0	0	10,938	0
[0, 0, 0, 0, 2]	0	0	0	0	0	7,950
[2, 2, 0, 0, 0]	0	79,562	3,047	0	0	0
[2, 0, 2, 0, 0]	0	20,008	0	0	0	0
[2, 0, 0, 2, 0]	0	9,203	0	0	0	0
[2, 0, 0, 0, 2]	0	7,476	0	0	0	0
[0, 2, 2, 0, 0]	0	0	21,879	0	0	0
[0, 2, 0, 2, 0]	0	0	4,125	0	0	0
[0, 2, 0, 0, 2]	0	0	1,627	0	0	0
[0, 0, 2, 2, 0]	0	0	0	9,486	0	0
[0, 0, 2, 0, 2]	0	0	0	2,712	0	0
[0, 0, 0, 2, 2]	0	0	0	0	7,366	0
[2, 2, 2, 0, 0]	0	40,106	5,170	0	0	0
[2, 2, 0, 2, 0]	0	6,376	369	0	0	0
[2, 2, 0, 0, 2]	0	3,380	148	0	0	0
[2, 0, 2, 2, 0]	0	8,219	0	0	0	0
[2, 0, 2, 0, 2]	0	2,848	0	0	0	0
[2, 0, 0, 2, 2]	0	6,102	0	0	0	0
[0, 2, 2, 2, 0]	0	0	2,224	10,699	0	0
[0, 2, 2, 0, 2]	0	0	1,833	1,253	0	0
[0, 2, 0, 2, 2]	0	0	353	0	1,797	0
[0, 0, 2, 2, 2]	0	0	0	1,555	7,558	0
[2, 2, 2, 2, 0]	0	221	23,250	1,752	0	0
[2, 2, 2, 0, 2]	0	86	5,541	0	0	0
[2, 2, 0, 2, 2]	0	1,978	1,608	0	0	0
[2, 0, 2, 2, 2]	0	86	0	7,398	0	0
[0, 2, 2, 2, 2]	0	0	2,391	11,610	0	0
[2, 2, 2, 2, 2]	0	0	56,742	1,357	1,120	0

Table 13a shows a warehouse that only contains empty containers. The table shows that for almost all cases the following two rules hold:

1. If storage location 1 contains an empty container, the ETV has to stand still at storage location 1.
2. If storage location 1 contains no empty container, the ETV has to stand still at the infeed.

The only exception is [0, 1, 1, 1, 1]; in this case the ETV has to stand still at storage location 2.

Table 13b shows a warehouse that only contains non-empty containers. The first observation is that whenever there are two or less storage locations that contain a non-empty container, the ETV has to stand still at the storage location closest to the infeed that contains a non-empty container. The same holds for the cases where the first, second and one other storage location contain a non-empty container.
Furthermore, when the cases described above do not hold, the following rule of thumb can be used: whenever there are three or more storage locations that contain a non-empty container, the ETV has to stand still at the storage location second closest to the infeed that contains a non-empty container.

5.4.6 Order picking

Finally, the simulation results for picking are analysed. The following categories are used:

$$[u, v, w, x, y] = [\mathbb{1}\{q_{0,0} > 0\}, \mathbb{1}\{q_{0,1} > 0\}, \mathbb{1}\{q_{1,0} > 0\}, \mathbb{1}\{q_{1,1} > 0\}, \mathbb{1}\{q_2 > 0\}]. \tag{31}$$

This category u indicates whether there are tasks waiting in queue $q_{0,0}$. Similarly, v, w, x and y indicate whether there are tasks waiting in queue $q_{0,1}$, $q_{1,0}$, $q_{1,1}$ and q_2 respectively.

Table 14: The results of simulation of the CTMC that corresponds to the ϵ-optimal policy for 250,000 days. The table shows the simulation results of picking. The bold value in each row is the action with the highest pick frequency in that row. The categories are defined in (31).

	$a_{0,0,c}; c \in I$	$a_{0,1,c}; c \in I$	$a_{1,0,c}; c \in I$	$a_{1,1}$	$a_{2,c}; c \in I$	a_2
$[0,0,0,0,0]$	0	0	0	0	0	0
$[0,0,0,0,1]$	0	0	0	0	**302,068**	0
$[0,0,0,1,0]$	0	0	0	**182,036**	0	0
$[0,0,0,1,1]$	0	0	0	**54,089**	32	0
$[0,0,1,0,0]$	0	0	**250,316**	0	0	0
$[0,0,1,0,1]$	0	0	**67,303**	0	25,582	0
$[0,0,1,1,0]$	0	0	1,939	**84,899**	0	0
$[0,0,1,1,1]$	0	0	1,112	**12,488**	35	0
$[0,1,0,0,0]$	0	**51,356**	0	0	0	0
$[0,1,0,0,1]$	0	**13,356**	0	0	0	0
$[0,1,0,1,0]$	0	3,678	0	**4,677**	0	0
$[0,1,0,1,1]$	0	**1,014**	0	769	0	0
$[0,1,1,0,0]$	0	**29,172**	5,643	0	0	0
$[0,1,1,0,1]$	0	**8,830**	784	0	0	0
$[0,1,1,1,0]$	0	**7,181**	0	6,499	0	0
$[0,1,1,1,1]$	0	**871**	0	217	0	0
$[1,0,0,0,0]$	**426,482**	0	0	0	0	0
$[1,0,0,0,1]$	35,951	0	0	0	4,848	**166,066**
$[1,0,0,1,0]$	74	0	0	**74,440**	0	0
$[1,0,0,1,1]$	107	0	0	**22,782**	0	29
$[1,0,1,0,0]$	1,417	0	**191,342**	0	0	0
$[1,0,1,0,1]$	0	0	**40,447**	0	14,722	14,116
$[1,0,1,1,0]$	0	0	1,641	**90,178**	0	0
$[1,0,1,1,1]$	0	0	710	**7,388**	0	1
$[1,1,0,0,0]$	8,645	**12,395**	0	0	0	0
$[1,1,0,0,1]$	**3,059**	2,341	0	0	0	725
$[1,1,0,1,0]$	0	542	0	**4,286**	0	0
$[1,1,0,1,1]$	0	10	0	**782**	0	0
$[1,1,1,0,0]$	**19,688**	15,429	308	0	0	0
$[1,1,1,0,1]$	**4,110**	877	0	0	0	147
$[1,1,1,1,0]$	0	5,185	0	**15,769**	0	0
$[1,1,1,1,1]$	0	5	0	**245**	0	0

The results show that removing containers from the store has the highest priority in general, where removing non-empty containers has a higher priority than removing empty containers from the warehouse. However, for the other cases no clear rules can be found. Therefore, the heuristic policy will prescribe that if the system is in a state belonging to a certain category, it chooses the action with the highest frequency in that category. The actions with the highest frequency are made bold for each category in Table 14.

5.4.7 Overview

To summarise the rules discussed in the previous sections, an overview of the heuristic policy based on the ϵ-optimal will be given. This version of heuristic policy is labelled as 'small' since it corresponds to a policy for the MDP problem.

Heuristic policy (small)

Picking

The current state of the system is converted to a category defined in (31). Then, the action that has the highest pick frequency in the results of the CTMC simulation will be chosen.

Storing empty containers

First of all, if storage location 1 is filled with an empty or a non-empty container, a new empty container is stored in the first open storage location whenever it does not violate one of the following rules:

1. At least one of the first two storage locations has to contain a non-empty container.
2. At least two of the first four storage locations have to contain a non-empty container.

If it does violate one of the rules, the empty container will be stored in the second open storage location.
Secondly, if storage location 1 is empty and there are two or more tasks waiting in the queue, the empty container will be stored in the first open storage location. Otherwise, the empty container will be stored in the second open storage location.

Removing empty containers from the warehouse

In almost all cases, the empty container that is closest to the infeed will be removed. This does not hold for categories $[1, 1, \cdot]$ where the ETV stands at a storage location containing an empty container. In that case, the ETV will remove the empty container from the storage location it currently stands.

Storing non-empty containers

For storing non-empty containers, the first open storage location is used.

Serving build-up stations

When not all of the first three storage locations are filled and there is an empty container in the queue, this empty container will be used to be placed at the build-up station that has finished its service. However, when the first three storage locations are all filled, the ETV picks an empty container from the warehouse.
In the case the ETV has to pick an empty container from the warehouse, the ETV picks the empty container the closest to the infeed.

Standing still during idle periods

First of all, in case the warehouse contains only empty containers and the first storage location is not empty, the ETV stands still at the first storage location. If the first storage location is empty, it will stand still at the infeed. However, there is one exception: if storage locations 2, 3, 4 and 5 contain an empty container, the ETV stands still at storage location 2.
Secondly, when at least one storage location contains a non-empty container, the ETV stands still at the storage location that contains a non-empty container that is the closest to the infeed. This also holds when storage locations 1 and 2 and one other storage location contain

a non-empty container.
If there are three or more storage locations that contain a non-empty container, it stands still at the second storage location that contains a non-empty container.

5.5 Comparing policies

The performance of the following storage policies will be compared:

- Random storage policy: policy from literature
- Closest open location storage policy: policy from literature
- Class-based storage policy based on COI: policy from literature
- Heuristic policy: the easy-to-implement policy based on the ϵ-optimal policy
- ϵ-optimal policy: policy provided by solving the CTMDP

The comparison will be twofold. First of all, the value iteration algorithm will be used to quantify the performance of each policy by determining the gain of the corresponding Markov chain. Secondly, for the case study, the performance of each policy is determined with discrete-event simulation by calculating the average daily weighted waiting time and an average distribution of the waiting time for each policy. Besides that, a robustness analysis will be conducted for all strategies with value iteration and discrete-event simulation for the No Peak, Peak and High Peak workload scenarios, see Table 3.

To be able to analyse each policy in the MDP model, all storage policies will be translated such that they are implementable in the MDP model. However, the storage policies from the literature only prescribe storage rules. Therefore, the picking rules, rules for standing still in an idle period and rules for retrieving containers from the warehouse are set to be the same as those used for the heuristic policy. This concerns the closest open location storage policy, the random storage policy and the class-based storage policy using the COI. In this way, the differences in performance of the storage assignment rules can be observed more clearly.
Furthermore, two variants of the class-based storage policy using COI will be used. Both variants allocate the first two compartments to non-empty containers and the last three to empty containers. One variant will assign the closest open compartment of the available compartments allocated to a type of container, while the other will assign a random compartment of the available compartments allocated to a type of container. Note that only the latter is discussed in the literature, see de Koster et al. (2007).
Then, value iteration will be used to determine the gain of the corresponding Markov chain for each policy. These values will be used to compare the different strategies.

The analysis of each policy in the case study is more complicated. In the case study, each rack of the warehouse has two shelves where containers can be placed next to each other but also in front of each other. The shelf has depth one, meaning one container can be placed in front of one other. The placing of containers in front of each other can cause a blockage. A blockage means that a container at the back of the shelf has to be removed while a container is currently stored in front of that container. Therefore, the container at the back cannot be removed immediately; the shelf can only be emptied last in first out. Thus, a blockage will result in a longer service time of the ETV.
Therefore, different rules for different types of containers have to be used in order to prevent blockages. These rules have already been discussed in the internship report of KLM. However, for completeness, the rules will be discussed shortly.
Since the simulation model differentiates between the different sizes of containers (ULDs and

BCs) and between different types of non-empty containers (built-up containers and not broken-down containers), extra rules have to be implemented for these cases. These rules only apply to all class-based storage policies: ϵ-optimal storage policy, the heuristic policy and the class-based storage policy based on the COI, hereafter referred to as COI policy. The rules are listed below:

1. Compartments are chosen such that they contain two storage locations at the same shelf; one of the storage locations is directly in front of the other. Storage within these compartments is done by first filling the position at the back of the shelf and afterwards the position at the front.
2. For empty containers and built-up containers, compartments are specially assigned for only BCs or only ULDs.
3. A built-up container is only placed in front of another built-up container if the one at the back of the shelf has a scheduled departure time (SDT) that is later than or similar to the SDT of the container that has to be stored. A similar departure time means that both SDTs do not differ more than 45 minutes.
   Additionally, in the case that there is no suitable storage location for a built-up container, it is sometimes better to allocate a new compartment for this container than placing it in front of a container that has no similar SDT. Therefore, if less than 20 compartments have been allocated for built-up containers, a new compartment will be allocated. If more or exactly twenty compartments have been allocated, first all open positions within the allocated compartments will be filled.
4. A similar principle as for built-up containers is applied for not broken-down containers. Not broken-down containers are only placed in front of each other when more than eight compartments have been allocated for not broken-down containers. When eight or less compartments have been allocated, a new compartment will be allocated for not broken-down containers.

Since the ϵ-optimal policy is based on compartments of 32 storage locations instead of two storage locations, it has to be implemented differently. In the simulation model, this policy will divide the warehouse in zones of 32 storage locations (or 16 compartments) for non-empty containers and for empty containers. The choice of which zone is used for which type of container is done with the rules of the ϵ-optimal policy (provided by solving the MDP model). Thereafter, the first open compartment within a zone will be used to store a container of the corresponding type. The allocation of a zone to a specific type of container is reset when all compartments in that zone are empty.

The COI policy also uses zones for a specific type of container. It assigns a zone of 24 compartments to non-empty containers. This zone contains the compartments closest to the infeed of the warehouse. The rest of the compartments are contained in a zone for empty containers. In this case, the allocation of zones to a specific type of container cannot be reset.

Similarly as the ϵ-optimal policy, the heuristic policy is based on compartments of 32 storage locations instead of two storage locations. Therefore, the extracted rules have to be adapted to be able to simulate the policy. This adaptation is an interpretation of the decision rules of the policy described in Section 5.4.7. This means that the new heuristic policy is not set in stone, but it will be interpreted to derive decision rules that can be implemented in the warehouse of the case study. On top that, two versions of the picking rules are used in the adaptation. These rules will be described below. Heuristic policy 1 uses picking rule 1; heuristic policy 2 uses picking rule 2. Picking rule 2 is the current picking rule of KLM Cargo. This results in the heuristic policy for simulation below. This version of heuristic policy is labelled as 'big' since it corresponds to a policy for the case study. Note that this version is still MDP-based.

Heuristic policy (big)

Picking 1

The current state of the system is translated to a state in the MDP. Then, it is determined to which category the current state belongs; the categories are defined in (31). Then, the action that has the highest pick frequency in the results of the CTMC simulation will be chosen.

Picking 2

The following picking hierarchy is used: 1. Removing built-up containers from the warehouse; 2. Serving build-up station and retrieving empty container; 3. Retrieving OoG and not broken-down containers from warehouse and storing any container in the warehouse.
Additionally, after a task has been waiting longer than 500 seconds in the queue, it gets maximum priority.

Storing empty containers

Empty containers will never be stored in the first eight compartments; these compartments are left as a gap for storing non-empty containers. The only exception is when the rest of the store is completely filled. For the next eight compartments, the second open compartment is used. For the rest of the compartments, the first open compartment will be used for storing empty containers.

Removing empty containers from the warehouse

In all cases, the empty container that is closest to the infeed will be removed.

Storing non-empty containers

For storing non-empty containers, the first open compartment is used.

Serving build-up stations

In almost all cases, whenever an empty container is in the queue of the ETV, this empty container will be used to be placed at the build-up station that has finished its service. However, when the warehouse is completely filled, the ETV picks an empty container from the store.
When the ETV has to pick an empty container from the warehouse, the ETV picks the first empty container from the infeed.

Standing still during idle periods

If there are only empty containers or no containers stored in the warehouse, the ETV will stand still at the infeed. If there are non-empty containers in the warehouse, the ETV will stand still at the compartment that equals the average of the column labels of the positions of all non-empty containers.

6 Results

In this section, the execution of the analysis is discussed. First, the differences in performance of the MDP-based policies and of the policies from literature are quantified in the MDP model using value iteration. Afterwards, the robustness of each of these policies is tested using different workload scenarios. Secondly, the performance of the enlarged heuristic policy is tested under picking rule 1 and 2 using the discrete-event simulation model. Next, the performance of the MDP-based policies and the policies from literature are compared in the discrete-event simulation model of the case study. Additionally, a robustness analysis will be performed for all policies with the discrete-event simulation model.

6.1 MDP model results

The stopping criterion of the value iteration algorithm is set to $\epsilon = 0.05$.

6.1.1 Policy comparison

The results in Table 15 show a logical pattern; more complex policies have a lower gain. However, the COI policy using random storage performs worse than the closest open location storage. Since the MDP model only decides about compartment allocation, the closest open location storage policy becomes a closest open compartment storage policy. Thus, the closest open location storage policy becomes a class-based storage policy. Therefore, the average service time of the closest open location policy is lower than the average service time of the COI policy, because random storage assignment increases the space utilization at the expense of increased travel distance (Sharp et al., 1991).
Furthermore, it can be observed that the performance of the heuristic policy is closest to the performance of the ϵ-optimal policy.

Table 15: The gain of the CTMC of each policy for the High Peak scenario.

Policy	Gain
Random storage	13,474.895
Closest open location storage	12,233.572
COI (random)	12,294.828
COI (closest)	12,134.300
Heuristic (small)	12,049.629
Optimum (ϵ-optimal)	11,589.037

6.1.2 Robustness analysis

In the robustness analysis, the performances of all policies are tested under different workloads. To compare the performance of each policy, the optimal gain for each scenario is calculated.
Table 16 shows a similar pattern as Table 15. However, the COI policy using closest open location storage performs worse than the closest open location storage policy in low workload scenarios. This is a result of the lack of variability in the compartment allocation that increases the travel distance.
Additionally, it shows that when the workload decreases, the differences between the gains of all policies also decrease. Although the differences become smaller, the performance of the

heuristic policy is still the closest to the performance of the ϵ-optimal policy. Moreover, the ϵ-optimal policy is optimal or very close to optimal in all scenarios. Therefore, it can be concluded that both the heuristic and the ϵ-optimal policy perform well under different workload scenarios.

Table 16: The gain of the CTMC of each policy for the No Peak, Peak and High Peak scenario.

	Gain		
	No Peak	Peak	High Peak
Random storage policy	8,303.548	10,556.487	13,474.895
Closest open location storage policy	7,626.259	9,595.315	12,233.572
COI policy (random)	7,840.168	9,763.369	12,294.828
COI policy (closest)	7,728.443	9,617.550	12,134.300
Heuristic policy (small)	7,615.234	9,501.495	12,049.629
ϵ-optimal policy (High Peak)	7,379.466	9,174.677	11,589.037
Optimum	7,334.935	9,157.839	11,589.037

6.2 Case study results

During the simulations, information is gathered about the following performance indicators:

- Daily (weighted) waiting time
- Waiting time per container
- Service time per container
- Percentage of containers waiting less than x seconds, for $x \in \{60, 120, 180, 240, 300, 360\}$

A 99% confidence interval for each of the performance indicators is constructed.

6.2.1 Heuristic policy

Using the simulation model, the picking rules of the heuristic policy can be compared. It can be observed from Table 17 that the heuristic policy performs better in the simulation when picking rule 2 is used, since it has a lower daily weighted waiting time. However, at first glance, the results of the waiting time distribution seem to contradict this observation: picking rule 1 has higher percentages than picking rule 2 at all five observed points. A closer look reveals that when x increases the probability that a container waits less than x seconds under picking rule 1 gets closer to that of under picking rule 2: for $x = 60$ the difference between the percentage of picking rule 1 and of rule 2 is 3.20% and for $x = 360$ the difference is 1.43%. Combining this with the observation that picking rule 2 has a lower daily weighted waiting time, it can be concluded that the waiting time distribution of picking rule 1 has a longer tail than that of picking rule 2. This is a result of the fact that picking rule 2 assigns the highest priority to containers that have been waiting longer than 500 seconds, while picking rule 1 does not. Therefore, containers can wait very long in the queue under picking rule 1. Thus, picking rule 2 is still better since it has a lower percentage of high waiting times.
To conclude, since picking rule 2 has a lower daily weighted waiting time and its waiting time distribution has a smaller tail, only the heuristic policy with picking rule 2 will be analysed in the following simulations.

Table 17: The simulation results of the heuristic policy with picking rule 1 and 2 using the High Peak scenario. For each of the performance indicators a 99% confidence interval is constructed.

	Heuristic policy 1 (big)		Heuristic policy 2 (big)	
	Mean	Confidence Interval	Mean	Confidence Interval
Daily weighted waiting time (s)	181,341.12	[180,637.86; 182,044.37]	167,527.42	[166,741.02; 168,313.83]
Daily waiting time (s)	172,538.19	[171,852.90; 173,223.47]	158,432.40	[157,688.66; 159,176.15]
Waiting time per container (s)	245.41	[244.64; 246.18]	225.78	[224.87; 226.69]
Service time per container (s)	54.90	[54.89; 54.91]	54.41	[54.40; 54.42]
Percentage waiting time <60 s (×100%)	0.5365	[0.5361; 0.5369]	0.5045	[0.5040; 0.5049]
Percentage waiting time <120 s (×100%)	0.6399	[0.6395; 0.6404]	0.6002	[0.5996; 0.6009]
Percentage waiting time <180 s (×100%)	0.7101	[0.7097; 0.7105]	0.6711	[0.6705; 0.6718]
Percentage waiting time <240 s (×100%)	0.7619	[0.7614; 0.7623]	0.7275	[0.7268; 0.7282]
Percentage waiting time <300 s (×100%)	0.8020	[0.8016; 0.8024]	0.7715	[0.7709; 0.7722]
Percentage waiting time <360 s (×100%)	0.8223	[0.8219; 0.8227]	0.8080	[0.8074; 0.8087]

6.2.2 Policy comparison

Table 18 shows again a clear pattern for almost all policies; the more complex the policy, the better the performance. However, this does not hold for the implementation of the ϵ-optimal policy and the COI policy using random storage assignment. The ϵ-policy performs less, possibly since it is far away from reality due to all assumptions that are needed to make the MDP tractable. The COI policy using random compartment assignment performs less, since random assignment increases the travel distance.
Furthermore, the table shows that the results of the waiting time distribution of the ϵ-optimal policy are better than the results of the heuristic policy, while the daily weighted waiting time of the ϵ-optimal policy is higher than that of the heuristic policy. This is again the result of the fact that that the ϵ-optimal policy uses a MDP-based picking rule (the MDP-optimal variation of picking rule 1), which results in a longer tailed waiting time distribution.
The full simulation results can be found in Table 18 in Appendix D, see Table 23.

Table 18: The simulation results of the mean of each performance indicator of each policy using the High Peak scenario.

	Random storage	Closest open location storage	COI (random)	COI (closest)	Heuristic (big)	ϵ-optimal
Daily weighted waiting time (s)	368,618.18	266,787.53	332,398.43	184,741.38	167,527.42	188,227.92
Daily waiting time (s)	349,576.07	251,968.49	315,799.92	174,879.21	158,432.40	179,292.42
Waiting time per container (s)	517.47	360.10	453.40	249.18	225.78	256.81
Service time per container (s)	73.43	61.37	68.46	56.32	54.41	56.88
Percentage waiting time <60 s (×100%)	0.3762	0.4470	0.3874	0.4874	0.5045	0.5449
Percentage waiting time <120 s (×100%)	0.4508	0.5230	0.4692	0.5830	0.6002	0.6446
Percentage waiting time <180 s (×100%)	0.5009	0.5810	0.5243	0.6541	0.6711	0.7102
Percentage waiting time <240 s (×100%)	0.5468	0.6298	0.5746	0.7119	0.7275	0.7560
Percentage waiting time <300 s (×100%)	0.5881	0.6707	0.6191	0.7574	0.7715	0.7955
Percentage waiting time <360 s (×100%)	0.6253	0.7071	0.6588	0.7952	0.8080	0.8167

6.2.3 Robustness analysis

Figure 9 shows the change in the daily weighted waiting time for each policy for different workload scenarios. The figure shows that the heuristic policy performs better than all other policies in all work scenarios.
Additionally, it shows almost the same pattern as in the other cases; more complex policies perform better. Still, the ϵ-optimal policy performs less than the COI policy using closest open compartment storage in the High Peak scenario. However, when the workload decreases, the ϵ-optimal policy performs better than the COI policy.

Furthermore, the COI policy performs the best using the closest open compartment storage. This is a result of a lower travel distance to allocate the closest open compartment instead of a random open compartment.
Finally, the figure shows again that the differences in performance of policies decrease when the workload decreases. However, the random storage policy, the closest open location storage policy and the COI policy with random compartment storage perform significantly less than the other policies in all scenarios.
The simulation results on which the figure is based can be found in Tables 23, 24 and 25 in Appendix D.

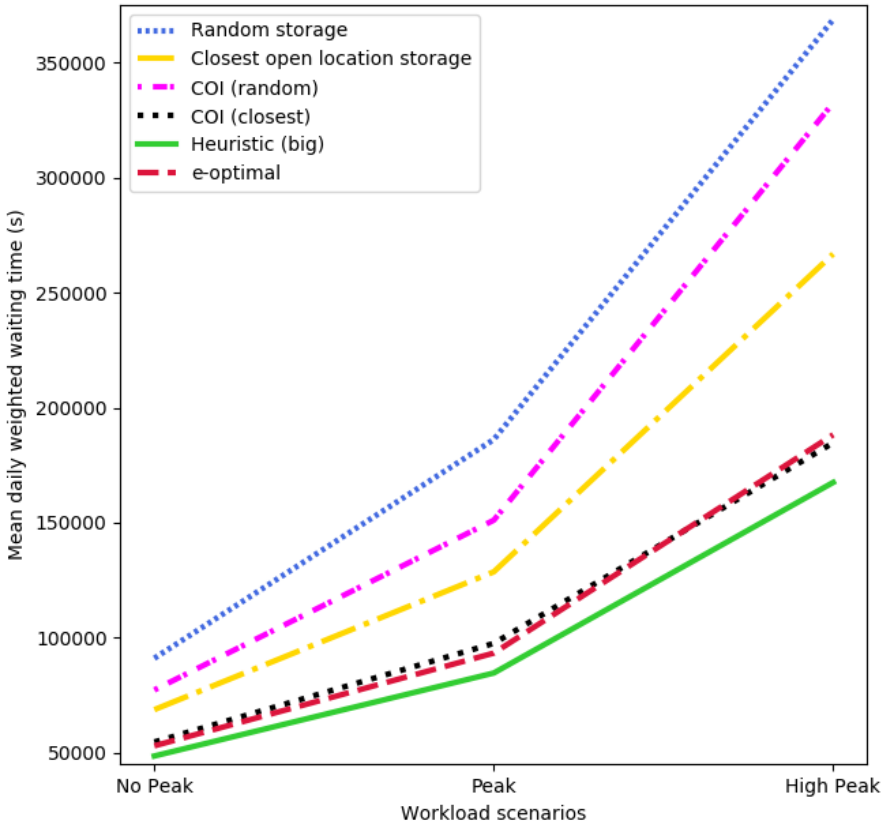

Figure 9: The mean daily weighted waiting time in seconds of all policies for the No Peak, Peak and High Peak scenarios.

7 Conclusion

In this report, a CTMDP with the average cost criterion was developed of an automated machine that serves a single-aisle warehouse and build-up stations to obtain a new picking and storage policy. The CTMDP minimises the average weighted waiting time of containers. An ϵ-optimal policy and an approximation to its gain was found using the value iteration algorithm. From this policy a heuristic policy was extracted. The heuristic policy, ϵ-optimal policy and the policies from the literature were tested in the MDP model and the case study using the value iteration algorithm and a discrete-event simulation of the sorting system respectively. To be able to simulate the policies, the heuristic policy had to be interpreted to be able to implement it. Moreover, some extra rules had to be implemented for parts of the sorting system which were disregarded in the CTMDP. Thereafter, a robustness analysis was done by running three scenarios with a different workload.

The results show that the heuristic policy is close to optimal and better than the existing policies. Moreover, it can be concluded that the heuristic policy performs well under different workloads. On top of that, the decision rules extracted from the ϵ-optimal policy can be interpreted easily to obtain an implementable policy.

One of the limitations of this research is the scalability due to the curse of dimensionality. However, with a slightly better computer or a more memory-efficient implementation of the value iteration algorithm, the model should be tractable for 10 compartments with all queues having capacity 1. Moreover, with better computers some of the assumptions could be relaxed or removed. For example, the assumption of having the infeed and outfeed at the same place or the assumption of disregarding which non-empty container has to be removed from the warehouse. This would result in a more detailed ϵ-optimal policy, which allows for a more thorough analysis.

Another limitation of this research is the applicability of the heuristic for multi-aisle warehouses or warehouses with more types of products. Therefore, it would be interesting for future research to try to interpret the heuristic in such a manner that it can be used for those types of warehouses. A possibility could be dividing all products into three categories based on their COI and then applying the heuristic policy on those categories.
Furthermore, it would be interesting to extend the current formulation to a warehouse served by two ETVs.
Additionally, for future research it may be interesting to use machine learning techniques such as reinforcement learning to obtain a policy instead of using the value iteration algorithm.

Furthermore, as an extension to this research, it would be interesting to analyse the transient behaviour of the MDP. In this research, the ϵ-optimal policy tells the decision maker what to do based on the steady-state behaviour of the corresponding CTMC. Therefore, it would be interesting to look for policies that prescribe how to go as fast as possible from a certain state in a subset of states $S_1 \subset S$ to an arbitrary state in an other subset $S_2 \subset S$, where $S_1 \cap S_2 = \emptyset$. For example, S_1 could be chosen as the set of all states where the system is very busy and S_2 as the set of all states where the system is almost empty.

All in all, the results of this research suggest that the MDP-based heuristic policy is an improvement of the current storage policies. Moreover, the simulation results suggest that the heuristic policy can be easily adjusted to be implemented to increase the efficiency of the warehouse.

Acknowledgment

First of all, I would like to thank prof. dr. N. M. van Dijk for his time, inspiration, enthusiasm and efforts during the supervision of my final project. Secondly, I would like to thank prof. dr. N. M. van Dijk, dr. J. C. W. van Ommeren and dr. D. Demirtas for making the time available to be present at and to evaluate my final presentation. Special thanks to KLM for allowing me to use the data and simulation model developed during my internship period. Finally, I would like to thank Elise Smit for the effort she put into reviewing my thesis along the way.

References

C. E. Bonferroni. Teoria statistica delle classi e calcolo delle probabilita. *Pubbl. d. R. Ist. Super. di Sci. Econom. e Commerciali di Firenze (in Italian)*, 8:1–62, 1936.

Y. Bukchin, E. Khmelnitsky, and P. Yakuel. Optimizing a dynamic order-picking process. *European Journal of Operational Research*, 219(2):335–346, 2012.

R. de Koster, T. Le-Duc, and K. J. Roodbergen. Design and control of warehouse order picking: A literature review. *European journal of operational research*, 182(2):481–501, 2007.

A. Eynan and M. J. Rosenblatt. Establishing zones in single-command class-based rectangular as/rs. *IIE transactions*, 26(1):38–46, 1994.

W. H. Hausman, L. B. Schwarz, and S. C. Graves. Optimal storage assignment in automatic warehousing systems. *Management science*, 22(6):629–638, 1976.

J. L. Heskett. Cube-per-order index - a key to warehouse stock location. *Transport and Distribution Management*, 3:27–31, 1963.

J. L. Heskett. Putting the cube-per-order index to work in warehouse layout. *Transport and Distribution Management*, 4:23–30, 1964.

T. J. Hodgson, R. E. King, S. K. Monteith, and S. R. Schultz. Developing control rules for an agv using markov decision processes. In *Decision and Control, 1985 24th IEEE Conference on*, volume 24, pages 1817–1821. IEEE, 1985.

A. M. Law. *Simulation Modeling and Analysis*. McGraw-Hill Education, 2015.

A. R. Odoni. On finding the maximal gain for markov decision processes. *Operations Research*, 17(5):857–860, 1969.

C. G. Petersen and G. Aase. A comparison of picking, storage, and routing policies in manual order picking. *International Journal of Production Economics*, 92(1):11–19, 2004.

M. L. Puterman. *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons, 2014.

M. J. Rosenblatt and Y. Roll. Warehouse capacity in a stochastic environment. *The International Journal Of Production Research*, 26(12):1847–1851, 1988.

G. P. Sharp, K. Il-Choe, and C. S. Yoon. Small parts order picking: Analysis framework and selected results. *Material Handling '90. Progress in Material Handling and Logistics*, 2:317–341, 1991.

J. P. van den Berg. Class-based storage allocation in a single-command warehouse with space requirement constraints. *International Journal of Industrial Engineering*, 3:21–28, 1996.

N. M. van Dijk, S. P. J. van Brummelen, and R. J. Boucherie. Uniformization: Basics, extensions and applications. *Performance Evaluation*, 118:8–32, 2018.

T. van Gils, K. Ramaekers, A. Caris, and R. B. M. de Koster. Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review. *European Journal of Operational Research*, 267(1):1–15, 2018.

P. D. Welch. On the problem of the initial transient in steady-state simulation. *IBM Watson Research Center*, 1981.

Abbreviations and symbols

Abbervation/Symbol	Meaning
A	Probabilistic generator of a CTMC
ACT	Active Cooled Temperature
a	An action from the action set A_s
a_h	Horizontal acceleration
α	Level of significance
$a_{0,0,c}$	Placing an empty container at compartment c in the warehouse
$a_{0,1,c}$	Removing an empty container from compartment c in the warehouse
$a_{1,0,c}$	Placing a non-empty container at compartment c in the warehouse
$a_{1,1}$	Removing a non-empty container from the warehouse
$a_{2,c}$	Removing a built-up container from a build-up station and placing the empty container from compartment c at that build-up station.
a_2	Removing a built-up container from a build-up station and placing an empty container from the queue at that build-up station
$a_{3,c}$	The ETV moves to compartment c and stands still at this compartment
A_s	Action set in state $s \in S$
B	The uniformisation rate
BC	Belly Cart
COI	Cube per order index
CTMC	Continuous-time Markov chain
CTMDP	Continuous-time Markov decision process
D	Set of all decision rules
d	Decision rule
d_ϵ	ϵ-optimal decision rule
d^∞	Policy that uses always decision rule d
d_a	Distance travelled by accelerating and slowing down
d_h	Horizontal travel distance
d_v	Vertical travel distance
$\mathbb{E}[\cdot]$	Expectation of $\cdot$
$\mathbb{E}_s^d[\cdot]$	Expectation of $\cdot$ given decision rule d and being in state s
ETV	Elevated Transport Vehicle
ϵ	Stopping criterion of the value iteration algorithm
FCFS	First Come First Served
I	Set of all indices of compartments
I_0	Set of all indices of compartments and the index of the infeed
I_1	Set of all indices of compartments that contain empty containers
I_2	Set of all indices of compartments that contain non-empty containers
IID	Independent Identically Distributed
J	Confidence interval

Abbervation/Symbol	Meaning
h	Bias
g	Gain
$k(s,a)$	Cost function depending on s and a
$k_d(s)$	The cost function for state s under decision rule d
k_d	Vector containing $k_d(s)$ for all $s \in S$
KLM	Royal Dutch Airlines
$\lambda_{0,0}$	Exponential parameter for the arrival rate at $q_{0,0}$
$\lambda_{0,1}$	Exponential parameter for the arrival rate at $q_{1,1}$
$\lambda_{1,0}$	Exponential parameter for the arrival rate at $q_{1,0}$
$\lambda_{1,1}$	Exponential parameter for the arrival rate at $q_{1,1}$
λ_2	Exponential parameter for the arrival rate at q_2
l	Warm-up period
M	Queue capacity
$\max_t\{\cdot\}$	The maximum of $\cdot$ over all possible values of t
$\min_t\{\cdot\}$	The minimum of $\cdot$ over all possible values of t
MDP	Markov decision process
MD	Markov deterministic
MR	Markov random
m	Length of a replication
μ	Mean
$\mu(\cdot,\cdot)$	Exponential parameter for the service time of the ETV that is a function of the travel distance and number of pick-ups
n	Number of replications
n_c	Number of columns in the warehouse
n_r	Number of rows in the warehouse
n_z	Number of compartments used
OoG	Out of Gauge
P	Transition probability matrix
P^t	t-step transition probability matrix
P_π	Transition probability matrix under policy π
$p(i\|j)$	Transition probability from j to i in a CTMC
$p(i\|j,a)$	Transition probability from j to i choosing action a in a CTMDP
$p_d(i\|j)$	Transition probability from j to i under decision Rule d in a CTMDP
$\mathbb{P}(\cdot)$	Probability of $\cdot$
Π	Set of all policies
π	Policy from the set of all policies
Q	Transition rate matrix
$\bar{q}$	Vector containing $q_{0,0}$, $q_{0,1}$, $q_{1,0}$, $q_{1,1}$ and q_2
$q_{0,0}$	The number of empty containers to store in the warehouse
$q_{0,1}$	The number of empty containers to remove from the warehouse
$q_{1,0}$	The number of non-empty containers to store in the warehouse
$q_{1,1}$	The number of non-empty containers to remove from the warehouse
q_2	The number of build-up stations that have finished building up a container

Abbervation/Symbol	Meaning
$q(i\|j)$	The transition rate of going from state j to state i in a CTMC
$q(i)$	The exit rate of state i in a CTMC
$q(i\|j,a)$	The transition rate of going from state j to state i choosing a in a CTMDP
$q_d(i\|j)$	The transition rate of going from state j to state i under decision rule d in a CTMDP
$q(i\|a)$	The exit rate of state i choosing action a in a CTMDP
$q_d(i)$	The exit rate of state i under decision rule d in a CTMDP
$\sum$	Sum symbol
$\sigma(\cdot,\cdot)$	Travel time function depending on the horizontal travel distances and number of pick-ups
S	State space
s	An element from S
$S^2(n)$	Sample variance of n replications
$t_{n-1,1-\alpha/2}$	Upper $1-\alpha/2$ critical point for the t-distribution with $n-1$ degrees of freedom
$t_{v_h^*}$	Time to reach maximum horizontal velocity
t_a	Maximum acceleration time
t_h	Horizontal travel time
t_v	Vertical travel time
$\tau(s'\|s,a)$	Transition rate function of going to state s' from state s choosing action a
ULD	Unit Loading Device
$\mathrm{Var}(\cdot)$	Variance of $\cdot$
v^n	The total reward vector after n steps in the value iteration algorithm
v_N^π	Total reward vector after N steps using policy π
v_h^*	Maximum horizontal velocity
v_v^*	Maximum vertical velocity
W	Waiting time
$\bar{w}$	A vector containing w_i for $1 \leq i \leq n_c$
$\bar{w}_i$	Information of compartment i
$\bar{X}(n)$	Sample mean of n replications
x	Current position of the ETV
x_1	The list of horizontal travel distance to complete an action
x_2	The number of pick-ups to complete an action
$\mathbb{1}\{\cdot\}$	Indicator function
$\wedge$	'And' symbol

A Overview sorter

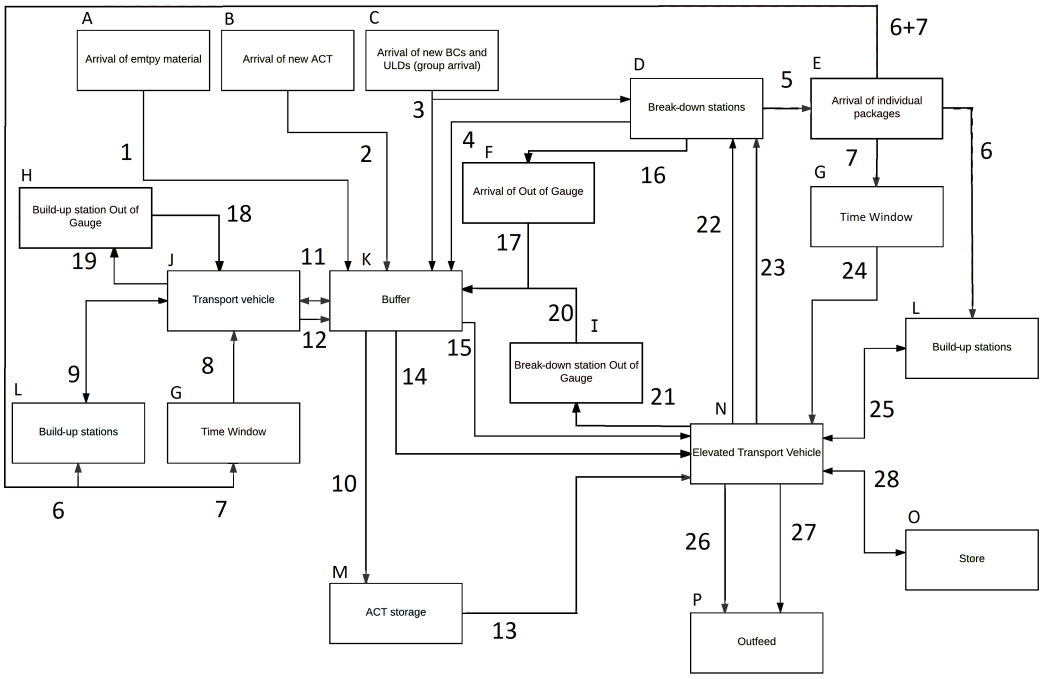

Figure 10: Overview sorter.

The explanation of this figure is given below.

Components

- Components A, B and C are arrival processes of a specific kind of container in. They enter the sorter via the infeed.
  - Component A is the arrival process of empty ULDs and BCs. These containers will be filled at the sorter at the build-up stations.
  - The arrival process of ACT can be found at component B. ACT are cooled containers filled with, for example, medicines.
  - Component C represents the arrival process of BCs and ULDs. These containers contain small packages and mail bags, which have to be sorted in the machine.
- At component D the packages and mail bags are manually removed from the ULDs and BCs. This is done at the so-called break-down stations.
- Component E represents the process of the transportation of packages and mail bags within the sorter. These are obtained by breaking down ULDs and BCs.
- Sometimes a package is too heavy to be lifted manually and then has to be lifted by a machine. Such a package is called Out of Gauge (OoG). This arrival process of OoG packages is contained in component F.

- Component G represents the build-up stations that are used to collect packages and mail that cannot be sorted at the moment. Such a build-up station is called a Time Window (TW).
- At component H ULDs and BCs are built up with OoG containers at the specially assigned OoG build-up stations.
- Sometimes during the break-down process at component D a packages is encountered in the container that is too heavy to lift manually. This container has to be broken down at a break-down station that only handles OoG containers. These OoG break-down stations can be found at component I.
- Component J represents the Transport Vehicle (TV). The transport vehicle places empty BCs or ULDs at a build-up stations and brings built-up containers from the build-up stations to the buffer.
- Component K is the buffer. In the buffer BCs and ULDs are temporarily stored. It can store up to five containers.
- At component L the ULDs and BCs are built up.
- ACT containers are stored at a store that has electricity connection. This store is represented by component M.
- Component N is the Elevated Transport Vehicle (ETV). This automated machine places BCs and ULDs at build-up stations, but also in the warehouse. Additionally, it can remove containers from the warehouse and build-up station. Subsequently, the ETV sends them away.
- Component O is the warehouse of the sorter. Here built-up containers, empty containers or new containers are temporarily stored.
- The outfeed of the system is represented by component P.

Arrows

- Arrows 1 and 2 represent the flow of containers to the buffer.
- Containers that flow through arrow 3 come from the infeed of the sorter and go to the break-down stations - if possible - or go immediately to the buffer.
- The flow of broken-down containers to the buffer is represented by arrow 4.
- Arrow 5 is the flow of packages and mail to the transporter during the break-down process.
- Arrow 6 represents the flow from the transporter to the build-up stations.
- Arrow 7 represents the flow from the transporter to the Time Window.
- The flow of built-up TW containers to the TV is captured in arrow 8.
- Built-up containers to the TV from the build-up stations and empty containers destined for the build-up stations coming from the buffer go via arrow 9.
- ACTs flow via arrow 10 from the buffer to the ACT storage.
- Arrow 11 represents the flow of built-up containers from the TV to the buffer and empty containers from the buffer to the TV.
- Built-up TW containers go via arrow 12 from TV to the buffer.

- Arrow 13 represents the flow of ACT containers from the ACT storage to the ETV.
- OoG containers flow via arrow 14 to the ETV from the buffer.
- New and empty containers flow from the buffer to the ETV via arrow 15.
- Arrow 16 represents the flow of BCs and ULDs that are labelled as OoG during the break-down process at component D.
- Arrow 17 represents the flow of OoG BCs or OoG ULDs to the buffer.
- The flow of built-up OoG containers from the OoG build-up stations to the TV is captured in arrow 18.
- Arrow 19 is the flow of empty containers to the OoG build-up stations from the TV.
- Broken-down OoG containers go from the OoG break-down stations to the buffer via arrow 20.
- An OoG container goes via the ETV to the OoG break-down stations. This is captured in arrow 21.
- BCs or ULDs that were stored before they were broken down go via arrow 22 from the ETV to the break-down stations.
- Built-up TW containers go from the ETV to the break-down stations via arrow 23.
- Arrow 24 represents the flow of built-up TW containers that go from the build-up station to the ETV.
- Built-up containers destined for the ETV coming from the build-up stations and empty containers destined for the build-up stations coming from the ETV go via arrow 25.
- ACTs go from the ETV to the outfeed via arrow 26.
- ULDs and BCs go from the ETV to the outfeed via arrow 27.
- Containers are stored and retrieved from the warehouse via arrow 28 by the ETV.

B Algorithm service time ETV

To accurately measure the performance of the ETV in the simulation, a detailed model for the ETV has to be implemented. Slightly different performance capacities may lead to inconclusive results of the model. The model of the ETV consists of two parts; the first part is the behaviour of the ETV and the other part consists of the technical specifications of the ETV. Behaviour of the ETV refers to the rules that are used to decide on its actions. This part is kept variable in the simulation model, since implementing different behaviours will result in different performances.

The technical specifications of the ETV are fixed in the model. These specifications consist of the maximum speed, acceleration, the lifting speed and the conveying speed of the ETV. The first three are given in the technical report by Lödige - the manufacturer of the sorting system - and the conveying speed is determined by measuring the time it takes to load a container onto or unload a container from the ETV. Evidently, the exact distances between store positions and build-up stations are also needed to compute the service times. These are delivered by Lödige in their ground plan of the sorter. By using the technical properties given by Lödige it is possible to accurately compute the time it takes to complete an action.

This section will focus on how to use the technical specifications to calculate the service time. First, the method of calculating the travel distance of the ETV will explained. Afterwards, an algorithm for calculating the service time of a certain actions will be presented.

B.1 Distance

From the ground plan of the store, the distance between adjacent build-up stations and adjacent store positions can be determined. Note that the distance between two adjacent store positions is equal to the distance between two adjacent build-up stations since they are situated directly above the build-up stations. The ground plan can be found in Figure 11.

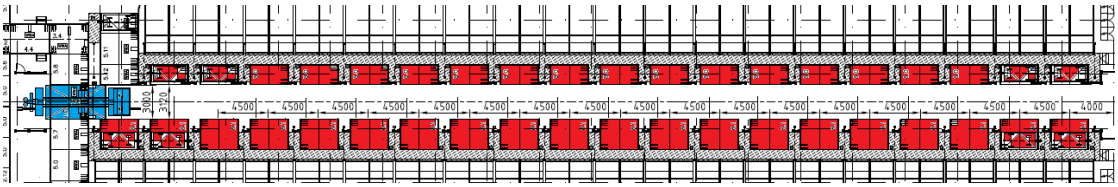

Figure 11: A top view of the store. The blue machine is the ETV and the red blocks are the build-up stations. The distances are given in millimetres.

To calculate the travel distance of the ETV for a certain action, the simulation model keeps track of the current position of the ETV. It directly updates the position when an action has been completed. The position of the ETV consists of a horizontal and vertical component.

The horizontal component is an integer between -1 and 19, where -1 corresponds to the infeed of the store and the other labels are used to indicate in front of which build-up station the ETV stands. The build-up stations on the far left are labelled with a 0, the directly adjacent build-up stations are labelled with a 1. Again, the build-up stations that are adjacent on the right side of the build-up stations labelled with a 1, will be labelled with a 2. This will continue until all build-up stations are labelled. The horizontal component of the ETV can only take the value of the label of a build-up stations due to the fact that the ETV will never stand still in between build-up stations.

Since the distance between adjacent build-up stations is fixed for all build-up stations, the distance between non-adjacent build-up stations can be calculated by determining the difference between the labels and multiplying the difference by the distance between two adjacent build-up stations. This method is also applied in the simulation model.

If the ETV has to store a container, the ETV decides where to put the container in the store using the rules about its behaviour. Therefore, the functions that model the behaviour of the ETV, return the label of the store position to which the ETV has to move. Then, by calculating the difference between the current position of the ETV and the label of the destinations, the horizontal travel distance can be calculated.

The vertical component consists of an integer between 0 and 2. In this case, 0 corresponds with the ground floor of the store, 1 with the first store layer and 2 with the second store layer. At the ground floor, the build-up stations are located, while at the other layers the store positions are located. The vertical travel distance is calculated in the same way as the horizontal travel distance.

B.2 Service time

In this paragraph, an algorithm to calculate the service time will be presented. An action of the ETV consists in most cases of moving to a certain location to retrieve a container and thereafter, move to a certain location in the store to deliver the container. Therefore, this algorithm requires a list as input containing the vertical and horizontal travel distances per movement. For example, suppose that the ETV has to travel from the infeed to label 10 at layer 1 and back to label 4 at layer 0. In this case, the input has to be: $[[11, 1], [6, 1]]$. The first and second component of the elements in the list consists of respectively the horizontal and vertical distance. The first and second element correspond to respectively the first and the second movement of the action.

To compute the travel time, it is assumed that the ETV slows down at the same rate as it accelerates. Moreover, it is assumed that both the acceleration and the slowing down are uniform. Therefore, two situations can occur; in the first situation the ETV reaches its maximum velocity, where the second situations occurs if the ETV does not reach its maximum velocity. These situations are sketched in Figure 12.

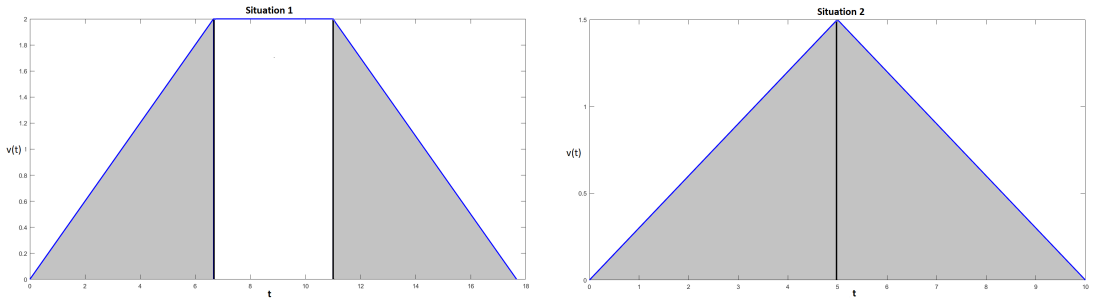

Figure 12: Velocity-time graphs of the ETV. The grey shaded areas in a graph are equal.

These situations give rise to an algorithm that first determines if situation 1 or 2 occurs and then calculates the travel time. Afterwards, it will add a constant amount of time if a container has to be loaded to or unloaded from the ETV.

Figure 12 shows clearly that the ETV can accelerate as long as it does not pass the halfway mark. Otherwise, the ETV will not stand still at the right moment since it takes the same amount of time and distance to slow down as it does to accelerate.

Table 19: Notation

Notation	Definition
v_h^*	Maximum horizontal velocity
v_v^*	Maximum vertical velocity
d_h	Horizontal distance to travel
d_v	Vertical distance to travel
a_h	Horizontal acceleration
d_a	Distance travelled by accelerating and slowing down
$t_{v_h^*}$	Time to reach maximum horizontal velocity
t_a	Maximum acceleration time
t_h	Horizontal travel time
t_v	Vertical travel time

Firstly, using the notation introduced in Table 19, the horizontal travel time will be calculated. Since the velocity of the ETV is zero at the start, the maximum acceleration time can be determined

$$\frac{1}{2}d_h = \frac{1}{2}a_h \cdot t_a^2 \iff t_a = \sqrt{\frac{d_h}{a_h}}.$$

The time to reach maximum speed can easily be calculated by

$$v_h^* = a_h \cdot t_{v_h^*} \iff t_{v_h^*} = \frac{v_h^*}{a_h}.$$

Now, it can be determined which situation applies. If $t_{v_h^*} < t_a$, situation 1 applies, otherwise situation 2 applies.

Suppose situation 1 occurs, then only the time to cruise at maximum speed has to be calculated. To calculate it, determine the distance covered by accelerating and slowing down d_a, since the residual distance is equal to $d_{res} = d_h - d_a$. Since the ETV can accelerate to maximum velocity, d_a can be calculated

$$d_a = 2 \cdot \left(\frac{1}{2} \cdot a_h \cdot t_{v_h^*}^2\right),$$

due to the uniform acceleration. Finally, the residual travel time is equal to $t_{res} = \frac{d_{res}}{v_h^*}$ and therefore

$$t_h = t_{res} + 2 \cdot t_{v_h^*}.$$

Alternatively, suppose situation 2 occurs. Then, the horizontal travel time can be computed directly

$$t_h = 2 \cdot t_a,$$

again due to the assumption of uniform acceleration.

Parallel to these computations, the vertical travel time has to be computed. It is assumed that the vertical accelerations can be neglected, because of the low maximum vertical velocity. Thus, by using the notation introduced in Table 19, the calculation of the vertical travel time boils down to

$$d_v = v_v^* \cdot t_v \iff t_v = \frac{d_v}{v_v^*}.$$

Then, the travel time is equal to $t = \max\{t_h, t_v\}$.

Additionally, to obtain the time to finish one movement, the travel time has to be increased with a constant amount of time if a container has to be loaded onto or unloaded from the ETV. Both the loading and unloading cost approximately 10 seconds, based on my own observations in the the sorter. Thus, the service time per movement is equal to

$$x = t + \mathbb{1}\{\text{container has to be loaded to or unloaded from the ETV}\} \cdot 10,$$

where $\mathbb{1}\{\cdot\}$ is the indicator function, i.e. the indicator function is 1 if a container has to be loaded onto or unloaded from the ETV and 0 otherwise.

To finally obtain the total service time for an action, all service times per movement have to be summed. Thus, the total service time $\sigma = \sum_i t_i$, where t_i is the service time of the ith movement in the action. This method is formalised in Algorithm 1

Algorithm 1 Calculate the service time for an ETV action.

Require:
- L; a list containing the horizontal and vertical distance to travel per movement
- v_h^*; a float that is the maximum horizontal velocity
- v_v^*; a float that is the maximum vertical velocity
- h; a float that is the horizontal distance between two adjacent build-up stations
- v; a float that is the vertical distance between two adjacent build-up stations
- a_h; a float that is the horizontal acceleration
- *type*: a string that indicates the purpose of an action. If the ETV has to re-position without transporting a container, type has to by 'Re-position', otherwise it does not matter.

Ensure: σ; a float that represents the total service time for the action.

$t_{v_h^*} = v_h^* / a_h$ {Calculate the time to reach maximum horizontal speed}
$\sigma = 0$ {Initialize the total service time}
for $i = 0$ **to** length(L) **do**
  $d_h = L[i][0] \cdot h$ {Determine the horizontal travel distance}
  $d_v = L[i][1] \cdot v$ {Determine the vertical travel distance}
  $t_a = \sqrt{d_h / a_h}$ {Determine the maximum horizontal acceleration time}
  $t_v = d_v / v_v^*$ {Determine the vertical travel time}
  if $t_a <= t_{v_h^*}$ **then**
    $t_s = t_s + \max\{2 \cdot t_a, t_v\}$ {Increase the service time}
  else
    $x = \frac{1}{2} \cdot a_h \cdot t_{v_h^*}^2$ {Calculate the horizontal distance travelled}
    $d_a = 2x$ {Calculate the horizontal distance travelled}
    $d_{res} = d_h - d_a$ {Calculate the remaining horizontal distance to travel}
    $t_{res} = d_{res} / v_h^*$ {Calculate the time to bridge the remaining distance}
    $\sigma = \sigma + \max\{t_{res} + 2 \cdot t_{v_h^*}, t_v\}$ {Increase the service time}
  end if
  if not type = 'Re-position' **then**
    $\sigma = \sigma + 10$ {Increase the service time}
  end if
end for
return σ {Return the total service time}

C Transition rate function

$$
t(s'|s,a) = \begin{cases}
\mu([x,c],1) & \text{if } a = a_{0,0,c} \wedge w'_c - w_c = 1 \wedge \sum_{\substack{i\in I\\ i\neq c}} \mathbb{1}\{w'_i \neq w_i\} = 0 \wedge \\
& q'_{0,0} - q_{0,0} = -1 \wedge \sum_{i=2}^{5} \mathbb{1}\{\bar{q}'_i = \bar{q}_i\} = 4 \wedge x' = c; \\
& \text{for each } c \in I, \\
\mu([|x-c|,c],1) & \text{if } a = a_{0,1,c} \wedge w'_c - w_c = -1 \wedge \sum_{\substack{i\in I\\ i\neq c}} \mathbb{1}\{w'_i \neq w_i\} = 0 \wedge \\
& q'_{0,1} - q_{0,1} = -1 \wedge \sum_{\substack{i=1\\ i\neq 2}}^{5} \mathbb{1}\{\bar{q}'_i = \bar{q}_i\} = 4 \wedge x' = 0; \\
& \text{for each } c \in I, \\
\mu([x,c],1) & \text{if } a = a_{1,0,c} \wedge w'_c - w_c = 2 \wedge \sum_{\substack{i\in I\\ i\neq c}} \mathbb{1}\{w'_i \neq w_i\} = 0 \wedge \\
& q'_{1,0} - q_{1,0} = -1 \wedge \sum_{\substack{i=1\\ i\neq 3}}^{5} \mathbb{1}\{\bar{q}'_i = \bar{q}_i\} = 4 \wedge x' = c; \\
& \text{for each } c \in I, \\
\frac{1}{|I_2(s)|}\ \mu([|x-i|,i],1) & \text{if } a = a_{1,1} \wedge w'_i - w_i = -2 \wedge \sum_{\substack{j\in I\\ j\neq i}} \mathbb{1}\{w'_j \neq w_j\} = 0 \wedge \\
& q'_{1,1} - q_{1,1} = -1 \wedge \sum_{\substack{i=1\\ i\neq 4}}^{5} \mathbb{1}\{\bar{q}'_i = \bar{q}_i\} = 4 \wedge x' = 0; \\
& \text{for each } i \in I_2(s), \\
\frac{1}{n_c}\mu([|x'-x|,c,|c-x'|]),2) & \text{if } a = a_{2,c} \wedge w'_c - w_c = -1 \wedge \sum_{\substack{i\in I\\ i\neq c}} \mathbb{1}\{w'_i \neq w_i\} = 0 \wedge \\
& q'_2 - q_2 = -1 \wedge \sum_{i=1}^{4} \mathbb{1}\{\bar{q}'_i = \bar{q}_i\} = 4 \wedge x' > 0; \\
& \text{for each } c \in I_1(s), \\
\frac{1}{n_c}\mu([|x'-x|,x',x'],2) & \text{if } a = a_2 \wedge \sum_{i\in I} \mathbb{1}\{w'_i \neq w_i\} = 0\ \wedge q'_{0,0} - q_{0,0} = -1 \wedge \\
& q'_2 - q_2 = -1 \wedge \sum_{i=2}^{4} \mathbb{1}\{\bar{q}'_i = \bar{q}_i\} = 3 \wedge x' > 0 \\
\mu([|x-c|],0) & \text{if } a = a_{3,c} \wedge \bar{w}' = \bar{w} \wedge \bar{q}' = \bar{q} \wedge x' = c; \text{ for each } c \in I_0, \\
\lambda_{0,0} & \text{if } q_{0,0} < M \wedge \bar{w}' = \bar{w} \wedge q'_{0,0} - q_{0,0} = 1 \wedge \\
& \sum_{i=2}^{5} \mathbb{1}\{\bar{q}'_i = \bar{q}_i\} = 4 \wedge x' = x \\
\lambda_{0,1} & \text{if } q_{0,1} < M \wedge |I_1(s)| - q_{0,1} > 0 \wedge \bar{w}' = \bar{w} \wedge q'_{0,1} - q_{0,1} = 1 \wedge \\
& \sum_{\substack{i=1\\ i\neq 2}}^{5} \mathbb{1}\{\bar{q}'_i = \bar{q}_i\} = 4 \wedge x' = x \\
\lambda_{1,0} & \text{if } q_{1,0} < M \wedge \bar{w}' = \bar{w} \wedge q'_{1,0} - q_{1,0} = 1 \wedge \\
& \sum_{\substack{i=1\\ i\neq 3}}^{5} \mathbb{1}\{\bar{q}'_i = \bar{q}_i\} = 4 \wedge x' = x \\
\lambda_{1,1}||I_2(s)| - q_{1,1}| & \text{if } q_{1,1} < M \wedge |I_2(s)| - q_{1,1} > 0 \wedge \bar{w}' = \bar{w} \wedge q'_{1,1} - q_{1,1} = 1 \wedge \\
& \sum_{\substack{i=1\\ i\neq 4}}^{5} \mathbb{1}\{\bar{q}'_i = \bar{q}_i\} = 4 \wedge x' = x \\
\lambda_2 & \text{if } q_2 < M \wedge \bar{w}' = \bar{w} \wedge q'_2 - q_2 = 1 \wedge \\
& \sum_{i=1}^{4} \mathbb{1}\{\bar{q}'_i = \bar{q}_i\} = 4 \wedge x' = x \\
0 & \text{otherwise.}
\end{cases}
$$

D Tables

Table 20: Part 1 of the results of simulating the CTMC that corresponds to the ϵ-optimal policy for 250,000 days. The table shows how many times the action $a_{0,0,c}$ for any $c \in I$ in a certain category of states has been chosen during the simulation. The categories are defined in (24).

	$a_{0,0,1}$	$a_{0,0,2}$	$a_{0,0,3}$	$a_{0,0,4}$	$a_{0,0,5}$
$[0,0,0,0,0]$	15,728	23,321	0	0	0
$[0,0,0,0,1]$	8,303	8,456	0	0	0
$[0,0,0,0,2]$	487	1,110	10	0	0
$[0,0,0,1,0]$	4,165	4,103	0	0	0
$[0,0,0,1,1]$	7,072	7,645	0	0	0
$[0,0,0,1,2]$	475	504	0	0	0
$[0,0,0,2,0]$	569	1,394	10	0	0
$[0,0,0,2,1]$	851	960	0	0	0
$[0,0,0,2,2]$	370	903	0	0	0
$[0,0,1,0,0]$	1,344	3,462	0	0	0
$[0,0,1,0,1]$	1,858	0	0	0	0
$[0,0,1,0,2]$	72	180	0	0	0
$[0,0,1,1,0]$	1,832	0	0	0	0
$[0,0,1,1,1]$	10,409	0	0	0	0
$[0,0,1,1,2]$	23	251	0	0	0
$[0,0,1,2,0]$	190	266	0	0	0
$[0,0,1,2,1]$	118	527	0	0	0
$[0,0,1,2,2]$	57	220	0	0	0
$[0,0,2,0,0]$	1,083	2,759	0	0	0
$[0,0,2,0,1]$	1,400	1,293	0	0	0
$[0,0,2,0,2]$	120	343	0	0	0
$[0,0,2,1,0]$	780	908	0	0	0
$[0,0,2,1,1]$	1,252	1,886	0	0	0
$[0,0,2,1,2]$	391	401	0	0	0
$[0,0,2,2,0]$	529	1,230	0	0	0
$[0,0,2,2,1]$	727	702	0	0	0
$[0,0,2,2,2]$	538	986	0	0	0
$[0,1,0,0,0]$	3,487	0	12,153	0	0
$[0,1,0,0,1]$	306	0	4,429	0	0
$[0,1,0,0,2]$	56	0	235	325	0
$[0,1,0,1,0]$	296	0	3,736	0	0
$[0,1,0,1,1]$	816	0	7,570	0	0
$[0,1,0,1,2]$	76	0	438	0	0
$[0,1,0,2,0]$	35	0	602	0	0
$[0,1,0,2,1]$	159	0	641	0	0
$[0,1,0,2,2]$	116	0	519	0	0
$[0,1,1,0,0]$	258	0	0	5,050	0
$[0,1,1,0,1]$	149	0	0	2,301	0
$[0,1,1,0,2]$	32	0	0	220	0
$[0,1,1,1,0]$	129	0	0	0	2,616
$[0,1,1,2,0]$	24	0	0	0	338

Table 21: Part 2 of the results of simulating the CTMC that corresponds to the ϵ-optimal policy for 250,000 days. The table shows how many times the action $a_{0,0,c}$ for any $c \in I$ in a certain category of states has been chosen during the simulation. The categories are defined in (24).

	$a_{0,0,1}$	$a_{0,0,2}$	$a_{0,0,3}$	$a_{0,0,4}$	$a_{0,0,5}$
$[0,1,2,0,0]$	165	0	0	1,790	0
$[0,1,2,0,1]$	72	0	0	719	0
$[0,1,2,0,2]$	8	0	0	157	0
$[0,1,2,1,0]$	94	0	0	0	1,232
$[0,1,2,2,0]$	42	0	0	0	649
$[0,2,0,0,0]$	2,254	0	4,329	0	0
$[0,2,0,0,1]$	3,034	0	474	0	0
$[0,2,0,0,2]$	87	0	184	0	0
$[0,2,0,1,0]$	1,044	0	167	0	0
$[0,2,0,1,1]$	4,384	0	0	0	0
$[0,2,0,1,2]$	175	0	187	0	0
$[0,2,0,2,0]$	200	0	509	0	0
$[0,2,0,2,1]$	377	0	358	0	0
$[0,2,0,2,2]$	115	0	287	0	0
$[0,2,1,0,0]$	1,001	0	0	1,161	0
$[0,2,1,0,1]$	328	0	0	271	0
$[0,2,1,0,2]$	35	0	0	78	0
$[0,2,1,1,0]$	37	0	0	0	644
$[0,2,1,2,0]$	373	0	0	0	200
$[0,2,2,0,0]$	2,455	0	0	964	0
$[0,2,2,0,1]$	2,623	0	0	0	0
$[0,2,2,0,2]$	177	0	0	414	0
$[0,2,2,1,0]$	987	0	0	0	0
$[0,2,2,2,0]$	1,529	0	0	0	558
$[1,0,0,0,0]$	0	74	41	0	0
$[1,0,0,0,1]$	0	0	167	0	0
$[1,0,0,0,2]$	0	10	1	4	0
$[1,0,0,1,0]$	0	0	108	0	0
$[1,0,0,1,1]$	0	0	1,056	0	0
$[1,0,0,1,2]$	0	0	19	0	0
$[1,0,0,2,0]$	0	0	26	0	0
$[1,0,0,2,1]$	0	0	47	0	0
$[1,0,0,2,2]$	0	0	50	0	0
$[1,0,1,0,0]$	0	0	0	81	0
$[1,0,1,0,1]$	0	0	0	618	0
$[1,0,1,0,2]$	0	0	0	20	0
$[1,0,1,1,0]$	0	0	0	0	545
$[1,0,1,2,0]$	0	0	0	0	31
$[1,0,2,0,0]$	0	55	0	95	0
$[1,0,2,0,1]$	0	0	0	124	0
$[1,0,2,0,2]$	0	0	0	0	0
$[1,0,2,1,0]$	0	0	0	0	104
$[1,0,2,2,0]$	0	41	0	0	83

Table 22: Part 3 of the results of simulating the CTMC that corresponds to the ϵ-optimal policy for 250,000 days. The table shows how many times the action $a_{0,0,c}$ for any $c \in I$ in a certain category of states has been chosen during the simulation. The categories are defined in (24).

	$a_{0,0,1}$	$a_{0,0,2}$	$a_{0,0,3}$	$a_{0,0,4}$	$a_{0,0,5}$
$[1,1,0,0,0]$	0	0	0	2	0
$[1,1,0,0,1]$	0	0	0	31	0
$[1,1,0,0,2]$	0	0	0	24	0
$[1,1,0,1,0]$	0	0	0	0	14
$[1,1,0,2,0]$	0	0	0	0	0
$[1,1,1,0,0]$	0	0	0	0	17
$[1,1,2,0,0]$	0	0	0	0	5
$[1,2,0,0,0]$	0	0	0	39	0
$[1,2,0,0,1]$	0	0	283	0	0
$[1,2,0,0,2]$	0	0	0	8	0
$[1,2,0,1,0]$	0	0	0	0	103
$[1,2,0,2,0]$	0	0	8	0	0
$[1,2,1,0,0]$	0	0	0	0	131
$[1,2,2,0,0]$	0	0	0	267	0
$[2,0,0,0,0]$	0	39,987	0	0	0
$[2,0,0,0,1]$	0	15,746	0	0	0
$[2,0,0,0,2]$	0	1,463	0	0	0
$[2,0,0,1,0]$	0	7,440	0	0	0
$[2,0,0,1,1]$	0	19,786	0	0	0
$[2,0,0,1,2]$	0	668	0	0	0
$[2,0,0,2,0]$	0	2,065	0	0	0
$[2,0,0,2,1]$	0	1,252	0	0	0
$[2,0,0,2,2]$	0	1,228	0	0	0
$[2,0,1,0,0]$	0	4,235	0	0	0
$[2,0,1,0,1]$	0	2,418	0	0	0
$[2,0,1,0,2]$	0	184	0	0	0
$[2,0,1,1,0]$	0	107	0	0	1,990
$[2,0,1,2,0]$	0	348	0	0	0
$[2,0,2,0,0]$	0	4,409	0	0	0
$[2,0,2,0,1]$	0	1,989	0	0	0
$[2,0,2,0,2]$	0	565	0	0	0
$[2,0,2,1,0]$	0	1,267	0	0	0
$[2,0,2,2,0]$	0	1,915	0	0	0
$[2,1,0,0,0]$	0	0	15	11,311	0
$[2,1,0,0,1]$	0	0	5	4,337	0
$[2,1,0,0,2]$	0	0	1	466	0
$[2,1,0,1,0]$	0	0	0	0	4,414
$[2,1,0,2,0]$	0	0	518	0	0
$[2,1,1,0,0]$	0	0	0	0	2,627
$[2,1,2,0,0]$	0	0	0	3,691	0
$[2,2,0,0,0]$	0	0	9,621	0	0
$[2,2,0,0,1]$	0	0	4,184	0	0
$[2,2,0,0,2]$	0	0	436	0	0
$[2,2,0,1,0]$	0	0	1,588	0	0
$[2,2,0,2,0]$	0	0	1,055	0	0
$[2,2,1,0,0]$	0	0	0	2,440	0
$[2,2,2,0,0]$	0	0	0	4,899	0

Table 23: The simulation results of each policy using the High Peak scenario. For each of the performance indicators a 99% confidence intervals is constructed.

	Random storage policy		Closest open location storage policy		COI policy (random)		COI policy (closest)		Heuristic policy (big)		ϵ-optimal policy	
	Mean	Confidence Interval	Mean	Confidence Interval	Mean	Confidence Interval	Mean	Confidence Interval	Mean	Confidence Interval	Mean	Confidence Interval
Daily weighted waiting time (s)	368,618.18	[356,754.31; 380,482.05]	266,787.53	[265,652.00; 267923.05]	332,398.43	[330,014.00; 334,782.86]	184,741.38	[183,851.29; 185,631.48]	167,527.42	[166,741.02; 168,313.83]	188,227.92	[186,375.84; 190,080.00]
Daily waiting time (s)	349,576.07	[338,165.14; 360,987.01]	251,968.49	[250,891.09; 253,045.89]	315,799.92	[313,516.19; 318,083.65]	174,879.21	[174,039.55; 175,718.87]	158,432.40	[157,688.66; 159,176.15]	179,292.42	[177,506.71; 181,078.12]
Waiting time per container (s)	517.47	[501.87; 533.07]	360.10	[358.85; 361.35]	453.40	[450.49; 456.30]	249.18	[248.12; 250.25]	225.78	[224.87; 226.69]	256.81	[254.36; 259.26]
Service time per container (s)	73.43	[73.08; 73.79]	61.37	[61.36; 61.39]	68.46	[68.44; 68.48]	56.32	[56.31; 56.33]	54.41	[54.40; 54.42]	56.88	[56.83; 56.93]
Percentage waiting time <60 s	0.3762	[0.3712; 0.3812]	0.4470	[0.4464; 0.4475]	0.3874	[0.3867; 0.3882]	0.4874	[0.4870; 0.4879]	0.5045	[0.5040; 0.5049]	0.5449	[0.5378; 0.5520]
Percentage waiting time <120 s ($\times$100%)	0.4508	[0.4444; 0.4572]	0.5230	[0.5224; 0.5236]	0.4692	[0.4682; 0.4701]	0.5830	[0.5824; 0.5836]	0.6002	[0.5996; 0.6009]	0.6446	[0.6387; 0.6504]
Percentage waiting time <180 s ($\times$100%)	0.5009	[0.4939; 0.5080]	0.5810	[0.5803; 0.5816]	0.5243	[0.5232; 0.5254]	0.6541	[0.6535; 0.6547]	0.6711	[0.6705; 0.6718]	0.7102	[0.7054; 0.7151]
Percentage waiting time <240 s ($\times$100%)	0.5468	[0.5393; 0.5544]	0.6298	[0.6290; 0.6305]	0.5746	[0.5735; 0.5758]	0.7119	[0.7112; 0.7125]	0.7275	[0.7268; 0.7282]	0.7560	[0.7525; 0.7595]
Percentage waiting time <300 s ($\times$100%)	0.5881	[0.5803; 0.5958]	0.6707	[0.6699; 0.6715]	0.6191	[0.6179; 0.6203]	0.7574	[0.7568; 0.7580]	0.7715	[0.7709; 0.7722]	0.7935	[0.7928; 0.7982]
Percentage waiting time <360 s ($\times$100%)	0.6253	[0.6175; 0.6331]	0.7071	[0.7063; 0.7079]	0.6588	[0.6576; 0.6601]	0.7952	[0.7945; 0.7958]	0.8080	[0.8074; 0.8087]	0.8167	[0.8142; 0.8192]

Table 24: The simulation results of each policy using the Peak scenario. For each of the performance indicators a 99% confidence intervals is constructed.

	Random storage policy		Closest open location storage policy		COI policy (random)		COI policy (closest)		Heuristic policy (big)		ϵ-optimal policy	
	Mean	Confidence Interval	Mean	Confidence Interval	Mean	Confidence Interval	Mean	Confidence Interval	Mean	Confidence Interval	Mean	Confidence Interval
Daily weighted waiting time (s)	186,154.68	[183,040.29; 189,269.07]	128,671.42	[126,741.85; 130,600.98]	151,108.21	[147,172.16; 155,044.25]	97,588.09	[95,435.02; 99,741.15]	84,721.83	[83,688.01; 85,755.66]	93,376.22	[92,248.09; 94,504.36]
Daily waiting time (s)	174,590.82	[171,616.46; 177,565.19]	120,384.32	[118,557.80; 122,210.83]	141,891.67	[138,131.29; 145,652.06]	91,476.52	[89,432.42; 93,520.61]	79,421.51	[78,443.39; 80,399.63]	88,047.38	[86,959.82; 89,134.93]
Waiting time per container (s)	332.35	[327.20; 337.50]	227.55	[224.71; 230.38]	268.48	[262.09; 274.88]	172.58	[169.25; 175.91]	149.95	[148.50; 151.40]	166.88	[164.96; 168.80]
Service time per container (s)	72.64	[72.39; 72.90]	60.41	[60.28; 60.55]	68.94	[68.88; 69.01]	57.42	[57.22; 57.62]	54.22	[54.07; 54.36]	56.13	[56.02; 56.23]
Percentage waiting time <60 s	0.4650	[0.4622; 0.4679]	0.5419	[0.5397; 0.5441]	0.4823	[0.4793; 0.4853]	0.5683	[0.5652; 0.5713]	0.5904	[0.5889; 0.5919]	0.6130	[0.6056; 0.6204]
Percentage waiting time <120 s ($\times$100%)	0.5633	[0.5598; 0.5668]	0.6314	[0.6294; 0.6334]	0.5932	[0.5897; 0.5968]	0.6756	[0.6731; 0.6781]	0.6951	[0.6940; 0.6963]	0.7150	[0.7099; 0.7202]
Percentage waiting time <180 s ($\times$100%)	0.6255	[0.6219; 0.6290]	0.6969	[0.6952; 0.6986]	0.6615	[0.6579; 0.6651]	0.7456	[0.7436; 0.7476]	0.7637	[0.7628; 0.7647]	0.7782	[0.7742; 0.7823]
Percentage waiting time <240 s ($\times$100%)	0.6764	[0.6730; 0.6799]	0.7448	[0.7434; 0.7462]	0.7164	[0.7130; 0.7198]	0.7990	[0.7973; 0.8006]	0.8147	[0.8138; 0.8155]	0.8186	[0.8159; 0.8212]
Percentage waiting time <300 s ($\times$100%)	0.7178	[0.7145; 0.7211]	0.7807	[0.7794; 0.7819]	0.7599	[0.7568; 0.7631]	0.8381	[0.8368; 0.8394]	0.8510	[0.8501; 0.8518]	0.8515	[0.8496; 0.8534]
Percentage waiting time <360 s ($\times$100%)	0.7516	[0.7485; 0.7547]	0.8101	[0.8090; 0.8111]	0.7948	[0.7919; 0.7976]	0.8672	[0.8661; 0.8683]	0.8781	[0.8772; 0.8791]	0.8685	[0.8667; 0.8702]

Table 25: The simulation results of each policy using the No Peak scenario. For each of the performance indicators a 99% confidence intervals is constructed.

	Random storage policy		Closest open location storage policy		COI policy (random)		COI policy (closest)		Heuristic policy (big)		ϵ-optimal policy	
	Mean	Confidence Interval	Mean	Confidence Interval	Mean	Confidence Interval	Mean	Confidence Interval	Mean	Confidence Interval	Mean	Confidence Interval
Daily weighted waiting time (s)	91,144.43	[90,543.80; 91,745.05]	68,798.39	[68,408.26; 69,188.51]	77,356.55	[76,894.22; 77,818.88]	54,753.35	[54,423.13; 55,083.58]	48,662.48	[48,375.23; 48,949.72]	53,083.54	[52,386.89; 53,780.19]
Daily waiting time (s)	85,070.67	[84,514.72; 85,626.61]	64,162.78	[63,804.02; 64,521.55]	72,310.18	[71,882.46; 72,737.90]	51,175.79	[50,874.18; 51,477.41]	45,515.55	[45,251.05; 45,780.05]	49,677.76	[49,006.92; 50,348.60]
Waiting time per container (s)	214.13	[212.92; 215.34]	161.36	[160.56; 162.16]	181.88	[180.95; 182.81]	128.64	[127.96; 129.32]	114.46	[113.86; 115.05]	125.07	[123.45; 126.68]
Service time per container (s)	70.92	[70.83; 71.00]	59.48	[59.46; 59.50]	69.34	[69.31; 69.36]	57.90	[57.87; 57.93]	54.24	[54.21; 54.26]	55.87	[55.82; 55.92]
Percentage waiting time <60 s ($\times$100%)	0.5475	[0.5467; 0.5483]	0.6066	[0.6061; 0.6072]	0.5568	[0.5562; 0.5574]	0.6246	[0.6242; 0.6251]	0.6414	[0.6409; 0.6419]	0.6586	[0.6518; 0.6653]
Percentage waiting time <120 s ($\times$100%)	0.6578	[0.6569; 0.6586]	0.6993	[0.6987; 0.6999]	0.6778	[0.6771; 0.6784]	0.7309	[0.7305; 0.7313]	0.7447	[0.7441; 0.7453]	0.7570	[0.7537; 0.7604]
Percentage waiting time <180 s ($\times$100%)	0.7224	[0.7216; 0.7232]	0.7644	[0.7638; 0.7650]	0.7452	[0.7446; 0.7459]	0.7968	[0.7964; 0.7973]	0.8097	[0.8090; 0.8104]	0.8171	[0.8147; 0.8195]
Percentage waiting time <240 s ($\times$100%)	0.7715	[0.7707; 0.7723]	0.8075	[0.8069; 0.8081]	0.7956	[0.7950; 0.7963]	0.8445	[0.8440; 0.8450]	0.8553	[0.8546; 0.8560]	0.8537	[0.8523; 0.8551]
Percentage waiting time <300 s ($\times$100%)	0.8086	[0.8078; 0.8093]	0.8381	[0.8374, 0.8387]	0.8328	[0.8322; 0.8334]	0.8777	[0.8772; 0.8782]	0.8856	[0.8850; 0.8863]	0.8817	[0.8807; 0.8828]
Percentage waiting time <360 s ($\times$100%)	0.8369	[0.8362; 0.8376]	0.8621	[0.8615; 0.8628]	0.8611	[0.8606; 0.8617]	0.9006	[0.9001; 0.9011]	0.9073	[0.9067; 0.9079]	0.8966	[0.8956; 0.8975]