university of groningen / faculty of science and engineering

Master Design Project

Hybrid simulation of internal logistics based on outbound logistics to improve service level: a case study at Stiho DC Utrecht

Final Thesis

MSc. Industrial Engineering and Management

Author:

R.F.E. (Roel) Steggink S3148572

Supervisors:

dr. ing. A. Hübl	(first supervisor)
dr. A.K. Cherukuri	(second supervisor)
R. Wester	(company supervisor)

January 24, 2023

Abstract

De Stiho Groep (hereafter: DSG) is a large family owned distributor of all kinds of construction materials. Their distribution center (hereafter: DC) in Utrecht has 140 FTE and 90 motorized units together with an automated conveyer belt. Together, they daily collect 600 customer orders with materials in different weights and sizes from all departments and bring them to the shipping area where the materials will be loaded into a truck or trailer. The trucks travel to several construction sites or customer warehouses to release their cargo and then come back to the DC.

The main problem considered in this thesis is the insufficient First-Time-Right (hereafter: FTR) KPI performance indicating there are too many customer complaints. This value is currently about 97% where 98% or higher is desired. About 40% of these complaints were caused by an order not being received by the customer on the agreed date. Hence, the service level is insufficient. This had three main underlying causes:

1. The planned orders are not available in the expedition hall at the moment of loading the truck. They either still have to be picked or are in transit to the expedition hall.
2. The planned orders do not fit in the truck's trailer as maximum weight or volume is exceeded.
3. The truck arrives too late at a customer.

With a hybrid model combining an Anylogic discrete event simulation of the internal logistics with a capacitated vehicle routing problem optimization of the outbound logistics modeled in Python, a solution is presented to improve the service level and overcome the underlying causes.

Most orders that are not delivered on the agreed date to the customer are in the routes of the trucks that depart early. In the hybrid simulation, these orders were rescheduled to a later truck without the need of deploying an additional truck. This improved the service level from 97.7% to 99.8% which is above the desired goal of 99.5%. Other options on improving only the internal logistics were also considered, namely: decreasing batch size, using more employees, earlier order intake and batch picking. However, these options could not reach the desired service level.

By considering all customers instead of making a pre selection by the planner in the truck planning process, a reduction in driven distance of 14% could be achieved. Moreover, if administratively the maximum capacity of every truck is reduced by 5%, only 7km of additional travel distance was needed to deliver all orders. Hence, by being more on the save side regarding the capacity of the truck, the chance increases that all planned orders fit in the truck and thus more customers receive their orders on the agreed date. The main hurdle preventing the implementation currently is the correct weight and volume information of all customer orders.

Contents

1 Introduction to Stiho and their problems

De Stiho Groep is a large family owned distributor of all kinds of construction materials such as wooden beams, wooden plates, bricks, mortar and doors. They have three distribution centers in the Netherlands. Their DC in Utrecht has 140 FTE and 90 motorized units together with an automated conveyer belt. Together, they daily collect 600 customer orders with materials in different weights and sizes from all departments and bring them to the shipping area where the materials will be loaded into a truck or trailer. The DC has about 8,000 unique products in store. The trucks travel to several construction sites and customer warehouses to release their cargo and then come back to the DC. All movements are tracked in a Warehouse Management System (hereafter: WMS) and this system is also responsible for the ranking in which all orders are collected. This system was implemented about one and a half years ago and is still not functioning optimally. Customers can place orders until noon and they will be delivered next day. A customer order can consist of multiple order lines with different products. Order lines can consist of multiple picking lines if the same product is located in different shelves and not the entire order line is picked from the same place.

The thesis is structured in the following way. First, the system and scope from which the problem is observed is introduced. Then, the main problem is explained and underpinned with the underlying causes for which a solution will be found. Then, the approach to solve the problem is presented. In the second chapter, the first part of the solution is first generally presented and then applied to DSG. In the third chapter, the second part of the solution is presented. Furthermore, in the fourth chapter, both solutions are combined and multiple solutions to the problem are analysed and compared. The thesis ends with a conclusion including recap, scientific contribution of this thesis and suggestions for further research.

1.1 System description and scope

The system considered in this design project is the 60,000 m^2 DC of Stiho in Utrecht. From this DC, they deliver around 600 customer orders per day with 50 to 60 trucks. There are more than 8,000 unique products in store. In figure 1, the floor map is depicted to get an understanding of the DC's layout. From here, orders are shipped all over the Netherlands and Belgium. The DC consists mainly of two big halls and an outdoor storage. The orderpickzones from where all order lines are picked are indicated by PZ. At arrival new goods, trucks with new stock arrive. At machinery, wood is optionally machined in line with the customers wishes. At expedition storage the finished goods inventory (FGI) is located. At the expedition, trucks are loaded for departure towards the customers.

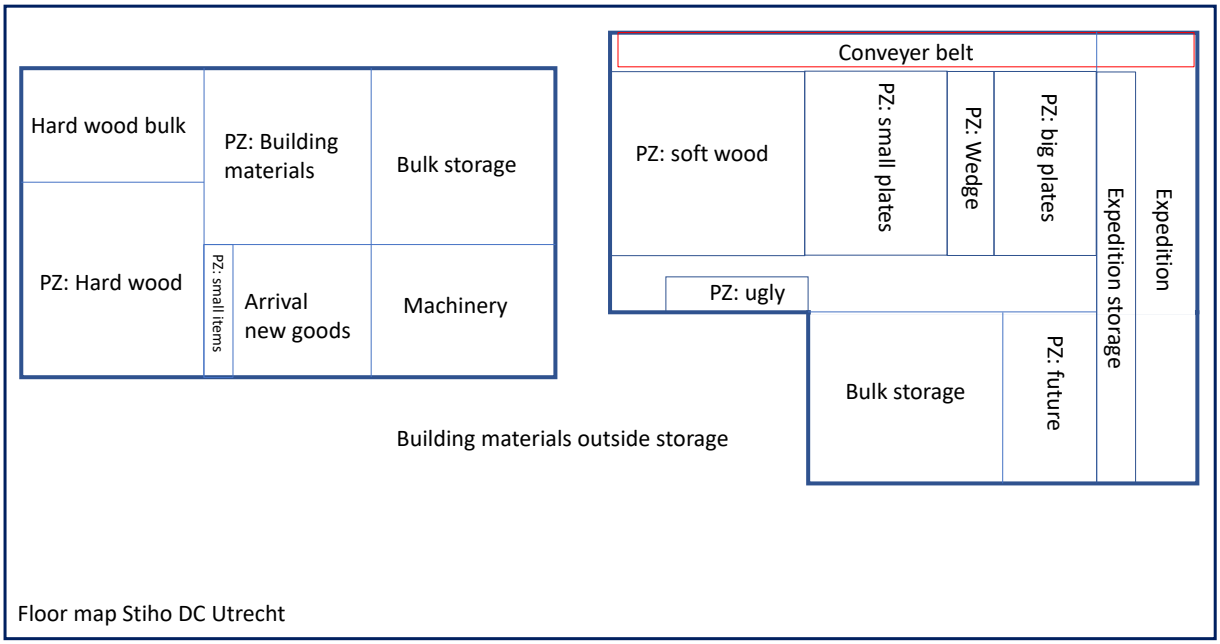


Figure 1: *The floor map of Stiho DC Utrecht indicating the location of all pickzones, arrival goods and expedition.*

Furthermore, the internal logistic process is showed in figure 2. In blue, the flow of materials is given, orange is the flow of information. The red dotted line indicates the pickzones where the order picking occurs. A customer order is received in the software M3 and will only be accepted if there is enough stock available. The planning department schedules the orders for the day ahead in a truck and determines the route. A priority is added to the order such that the logistics cockpit knows when the order is due in the expedition hall and schedules its picking lines. The 27 pickzones mentioned before are, for simplicity reasons, summarised in the 11 pickzones stated in the diagram. The orders are divided among the pickzones and the orderpickers start picking the order which goes per forklift, cart, conveyor belt or a combination of those to the expedition hall from where it will be loaded into the truck. Full packages will be picked up from the bulkstorage immediately and the pickzones are also provided with refills from the bulk storage. Orders with specialty products that are normally not in stock will be stored temporarily at the pickzone called Future until the date they are scheduled to depart. The logistics cockpit can boost the ranking of the picklines if they want to give more priority to certain orders. Once all ordercarriers are inside the truck, it starts the journey towards the customers.

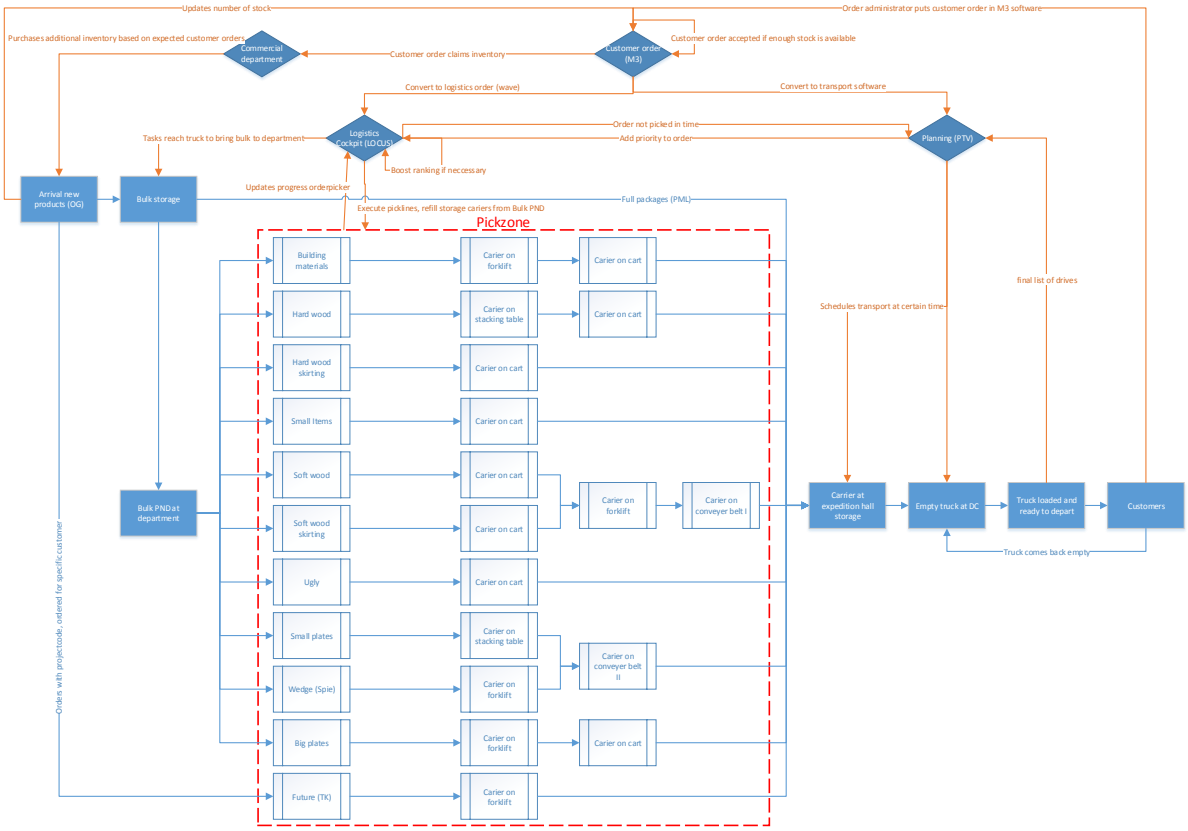


Figure 2: *The internal logistics process of Stiho DC Utrecht.*

The scope is the entire orderpicking and expedition process as indicated in figure 2. It is assumed that there is always stock available for all customer orders since the ordertaker has to check stock before an order can be filed.

1.2 Main problem: insufficient First-Time-Right KPI performance

The key performance indicator of DSG is First-Time-Right (FTR) which is calculated by the number of customer complaints divided by the number of stops. A stop can consist of one or multiple customer orders delivered to the same address by the same truck. FTR is in the core of lean six sigma as it tries to improve the quality of the process by reducing the number of defects doing things right the first time (Yang and El-Haik, 2009). In this case, the defects are the customer complaints. The board of directors has set a target at 98% or higher and measures it on a weekly basis. Currently, this target is not met as it was approximately 96% to 97% the past couple of months. In the past 6 months, about 40% of the complaints are concerning an order (partially) not being received by the customer on the agreed date. Hence, the service level is not sufficient. Service level is measured as number of orders delivered on time divided by the total number of orders. Furthermore, with the current number of employees and productivity, a maximum of 1,400 - 1,500 picklines can be picked in one day. On several days in September, order maximisation had to be instated by DC management. Hence, no orders with next day delivery are accepted anymore since the orders are expected not to be fulfilled in time. Both complaints about orders not delivered and not being able to place an order needed next day might cause customers to make use of the competitors' service. This has an immediate negative effect on revenue and profitability of DSG.

The root causes of this problem can be depicted in the diagram of figure 3 and will be elaborated on in

the next sections.

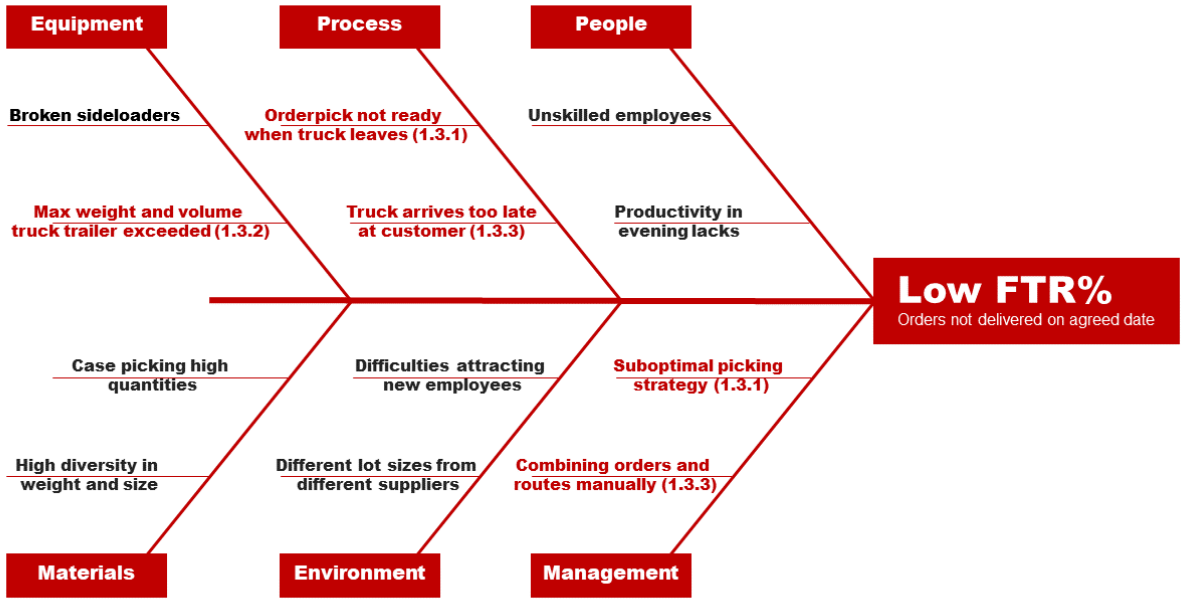


Figure 3: *Ishikawa diagram with the root cause analysis of the problem. The root causes are divided in six categories in which related problems are indicated. At the right, the main problem is placed. The root causes considered in this thesis are indicated in red and the section in which they are elaborated further is indicated between brackets. The black root causes outside the scope of this thesis are further explained in section 1.3.4.*

1.3 Underlying causes

There are three main reasons why not all orders are delivered to the customer on their scheduled date.

1. The planned orders are not available in the expedition hall at the moment of loading the truck. They either still have to be picked or are in transit to the expedition hall.
2. The planned orders do not fit in the truck's trailer as maximum weight or volume is exceeded.
3. The truck arrives too late at a customer.

The first problem is a very complex problem. At first glance, several employees indicated that there are too few orderpickers and the number of orders is too high. However, two months ago, the productivity was on average 7.6 picklines/hour instead of the current month's 6.8 picklines/hour. Productivity is measured as number of picked orders divided by employee hours. The difference is expected to come from the recent large number of new employees. Besides that, some pickzones are already very crowded so additional employees could decrease productivity and increase salary costs. Even if the productivity increases by experience, other problems remain. The first problem is elaborated further in section 1.3.1.

The second problem occurs for about 1% of the orders. If the order cannot be loaded in the truck anymore, the logistic support department together with the planning and the customer tries to come up with a solution. Most of the time, the order will be delivered on the next day. Hence, if the unfitted orders are very large and heavy, the next day there are also limitations on what can be transported. This problem is elaborated in section 1.3.2.

The third problem is mainly the result of the customer not being present anymore, since the delivery exceeds working hours or the customer needed the materials earlier on the day and delivery is not necessary anymore. In that case, the customer already picked up the materials himself by one of the Stiho shops or at a competitor. This problem is additionally explained in section 1.3.3.

1.3.1 WMS orderpick algorithm

The WMS software that DSG uses is called Locus. Locus converts the customer orders into pickorders which is called a wave. An algorithm determines the ranking in which the orders will be picked. It calculates the time required to fulfil a pickorder and calculates back from the time it is scheduled to be needed in the expedition. This time is currently for all kinds of products the same and is set to 3 minutes. However, one can imagine that it is far quicker to bring a box of screws to the truck than 40 beams of wood. In the pickzone soft wood, only 6% of the orders are picked within this time. Hence, the ranking algorithm is not capable to produce a feasible ranking as the initial proces parameters are not valid. Therefore, human interference is necessary. Locus tracks the progress per task and the logistics cockpit can steer by changing the priorities of the pickorders. Currently, the orders that have to be at the expedition with the earliest due date are boosted. This means that a higher priority is allocated to those orders. With this, the cockpit strives to receive them on time at the expedition. However, this causes several local optimums which is not the most optimal solution for the system as a whole as a smaller amount of picklines is considered, the travel times increase and the system becomes less efficient. Although the earliest due dates are then met, not all picklines are collected as the productivity is lower and the next day the DC starts with a backlog of orders. The picking sequence problem is extensively researched and solutions are provided in Moeller (2011).

1.3.2 Truck loading process

About 50-60 trucks are loaded on a daily basis at a certain scheduled time depending on their route. The trucks drive one or two rounds depending on how far their destinations are. The orders that are loaded into the truck are placed on zipcode ranking in the expedition hall temporary storage. However, not all orders that are going into the same truck are placed together as there is also a division between the sizes of the orders. Hence, the forklift that loads the truck has to travel a lot to gather all required items. This leads to a long loading time. In the case of second drives, the truckdriver has to wait longer which costs Stiho more money.

The planner sees the percentage of weight and volume reached when adding orders to a delivery route on the planning software. The planning department schedules too many orders in one truck such that maximum weight or volume is exceeded. This is mainly caused by incorrect weight and volumes in the WMS software which mostly occurs with specialised and/or custom-made items or very big items that have to be transported. Furthermore, the volume of insulation is often underestimated. Sometimes, the dimensions of the orders are larger than the small truck. Hence, the planning department has to schedule a different truck to this location. Furthermore, order lines of the same order in the same pickzone are combined into one pickorder. Hence, materials of different dimensions are combined into one package. If this package is stacked incorrectly by the orderpickers, the dimensions of the orders are larger than taken into account. These problems are also observed in Vélez-Gallego et al. (2020).

If an order cannot be delivered with the scheduled truck, it has to be rescheduled. This puts a strain on next day's planning. The order remains at the expedition hall until shipped.

1.3.3 Outbound logistics

As outbound logistics, the route the trucks drive along their customers and back to the DC is considered. About 600 customer orders are scheduled per day with 1 to 15 stops per route. A customer can order multiple orders which can be combined into one stop. Hence, the number of stops is slightly lower than the number of customer orders. Currently, the orders that are delivered per route are added by some experienced employees based on maximum weight, volume and what they indicate as logical route. The

planning software creates the order in which the selected customers are visited based on shortest distance. The planned departure time of the truck is used as input for the priority ranking in which the orderpickers pick the picklines. For most customers, and especially for contractors at construction sites, there are strict time limits in which delivery is possible. This is usually between 7 am and 4 pm on business days. Hence, it is important for the trucks to be on time, otherwise the order has to be rescheduled to another day. As not all weights and volumes are known, there is currently not an algorithm in place determining which orders are on which route and when to come back to the DC. Hence, the most efficient route is not created. This leads to both additional costs as the total driving time is longer and a lower FTR if the truck driver cannot reach the construction site in time. In Marques et al. (2020), it is stated that integrated planning of supply chain operations and outbound logistics lead to better results than decoupling.

1.3.4 Pickzone specific problems

Out of the 27 different pickzones, the pickzones soft wood and small plates were respectively responsible for 30.00% and 19.55% of the picklines in the past 2.5 months. Furthermore, 58.45% of the unavailable orders in the expedition hall originates from these two pickzones. Hence, these pickzones are identified as the bottlenecks setting the maximum threshold the DC can handle. These problems are included in the ishikawadiagram of figure 3 but will not be specifically solved in this thesis as the pickzone specific problems are too small compared to the other problems presented before. Furthermore, these problems are not part of the underlying causes of the main problem, hence they are outside the scope of this thesis.

Pickzone soft wood

The pickzone soft wood is mainly made up from six aisles with around 1,200 different shelves where orders are picked from consisting of a majority of beams in different sizes. Orderpickers drive in a cart and pick the right quantity from the different shelves on their wagon. Thereafter, a forklift puts the order on the conveyer belt towards expedition. All products are dynamically stored within the pickzone. Hence, the same product can be located in multiple shelves.

Gathering some picklines require a lot of time because it requires large quantities picked by hand. If the large quantities are exactly equal to one package, it will be shipped directly from the bulk storage to the expedition. However, even if this quantity only deviates slightly, the carrier is shipped via the pickzone. If there are more order lines to the order, the other products are combined with this carrier by hand.

Furthermore, if for example 35 wooden beams have to be collected, the system can direct the orderpicker to three different shelves far apart from eachother to collect smaller quantities that totals 35 whereas one shelve contains more than 35 beams. However, the system is designed to give higher priority to clear shelve locations over picking time. This results in high traveling distance for the orderpicker.

Furthermore, the orderpicker is waiting multiple times per hour on colleagues that either block the shelve with their cart or until a forklift unloads their cart on the conveyer belt. Sometimes, even eight employees are in the same aisle completely blocking eachother.

Moreover, the system should be calculating the route with the least travel time for the orderpicker. However, in practice, the orderpicker is sent back and forth to other aisles and then comes back to the aisle where the the order is put on the conveyer belt. Normally the orderpicker only collects one customer orders (which can consist of multiple picklines) and then unloads it on the conveyer belt.

The orderpick system Stiho uses and the corresponding problem, is also described in section 8 of Boysen et al. (2021).

Pickzone small plates

Small plates are picked with the aid of six stationary tables. Sideloaders bring pallets with plates and doors to the tables where a team of two orderpickers converts the right amount of plates from one pallet to an empty pallet next to it. Once all order lines are picked on the pallet and the order is fulfilled, the

carrier is transported over the conveyer belt to the expedition. Currently, there is a imbalance in the ranking of the orders which causes the sideloaders to wait on the orderpickers and the orderpickers to wait until the sideloaders bring them the right product. Furthermore, if a sideloader is broken, the orderpicking process becomes slower. The ranking problem will probably be fixed in the next couple of weeks and then this pickzone should run more smoothly. If this is the case, no further investigation is required since the problem is solved.

1.4 Approach to solve the problems

From the three underlying causes it can be concluded that the overall problem is human interference in algorithms to overcome a lack of initial correct values and parameters that prevent both the routing and the orderpick algorithm to work properly. This leads to sub-optimal solutions and a lot of additional work and thus salary costs. Hence, this thesis focuses on showing the benefit of considering all customers when setting the route and finding the optimal process parameters in which the orderpick ranking algorithm can operate optimally without much human interference. Furthermore, by letting both systems work more closely with eachother and simulating in hybrid simulation if, with current capacity, all planned orders can be fulfilled in due time, complaints can be avoided and FTR can be increased. Hence, the service level has to be improved. This yields the following goal:

Increase the FTR KPI performance to above 98% by ensuring that 99.5% of the planned orders are delivered to the customer on the agreed date. To achieve this, create a hybrid model with DES of the internal logistics with as input a TSP optimized outbound logistics and perform simulations with it in order to find optimal proces parameters.

1.4.1 Methods and tools

To achieve the service level of 99.5% mentioned in the goal, a model will be created to perform simulations with and obtain possible solutions. To create a conceptualized model of the DC, a digital twin is made to perform simulations with. According to Jia et al. (2022), a digital twin simulates the behavior of a physical system at different levels. They describe four main characteristics a digital twin should have: scalability, interoperability, expansibility, and fidelity. According to Agalianos et al. (2020), the digital twin can be made in a discrete event simulation (hereafter; DES) software. DES is a computer based model of a system as a discrete sequence of events moving forward in time. It can execute simulation software queries on (real-time) data from the physical twin. DES software can be used to enhance logistics processes and the simulations can be used to test virtual models prior to real-world applications. DES is proven to assist with predicting performance, experimenting and evaluating the merits of alternative scenarios (Agalianos et al., 2020). As DES enables experimentation with any element of a business system, it has major advantages over other operational research techniques (Brailsford et al., 2019). The digital twin in the DES model is expanded with an optimization model of the outbound logistics in Python to model the truck routing. When multiple types of simulation are combined, it is called hybrid simulation (Brailsford et al., 2019). In the case of this design project, DES will be combined with optimization. This is known as simulation based optimization. Hybrid simulations can be used to model complex enterprise-wide systems to solve real-world problems (Brailsford et al., 2019).

The main method that will be used is a case study at Stiho DC Utrecht. The data and process outline will be used to create and verify the hybrid model. The processes in the DC of DSG are depending on a previous process step. Hence as a tool, a simulation model can be used. The DES software that will be used is Anylogic. Based on the data from the WMS and a tour through the DC, a conceptualized model of the DC can be made. The data is fitted to a stochastic model.

The main goal is to optimize internal logistics based on outbound logistics. This yields a hybrid model with optimizer and discrete event simulation. The reason why a hybrid model is needed is because the

logistics from warehouse to customer depend on a collaboration between both the internal logistics and the outbound logistics. The process of Stiho is very suitable for hybrid modeling as the trucks depart at different times and have some overlap in the routing. Hence, it is possible to reschedule tardy orders to later departing trucks. The outbound logistics will first be optimized by applying a Traveling Salesman Problem to obtain the shortest route along every construction site. Hence, more customers should be visited within their delivery window. The next trailer should be fully loaded once the truck comes back to the DC in order to leave immediately. The solution of the optimization of the TSP will be used as deterministic input to the stochastic simulation model of the internal logistics on a recurrent basis, thus a hybrid model is created. Hence, different optimization scenarios will be examined. The solutions of the optimizer coded in Python will be tested in Anylogic. An overview of how the models operate together is shown in figure 4. A distance matrix is obtained from putting the customers' coordinates and the DC coordinates in a Anylogic GIS map. The GIS map calculates the distance over the road between two destinations. Then, this distance matrix together with additional customer information and constraints are put into the CVRP algorithm. This yields a routing schedule with due dates for the orderpick when the truck leaves. A DES model of the orderpick system is requiring the pickzone the order has to go through and will yield KPI performance results and the optimal process parameters required to let Stihos orderpick algorithm work smoothly. The hybrid part is if customers whose orders are not ready in time can be rescheduled to a truck that leaves later that day. In this way, the customer still receives his order on the agreed date. Hence, the CVRP model is ran twice, before and during the DES model execution.

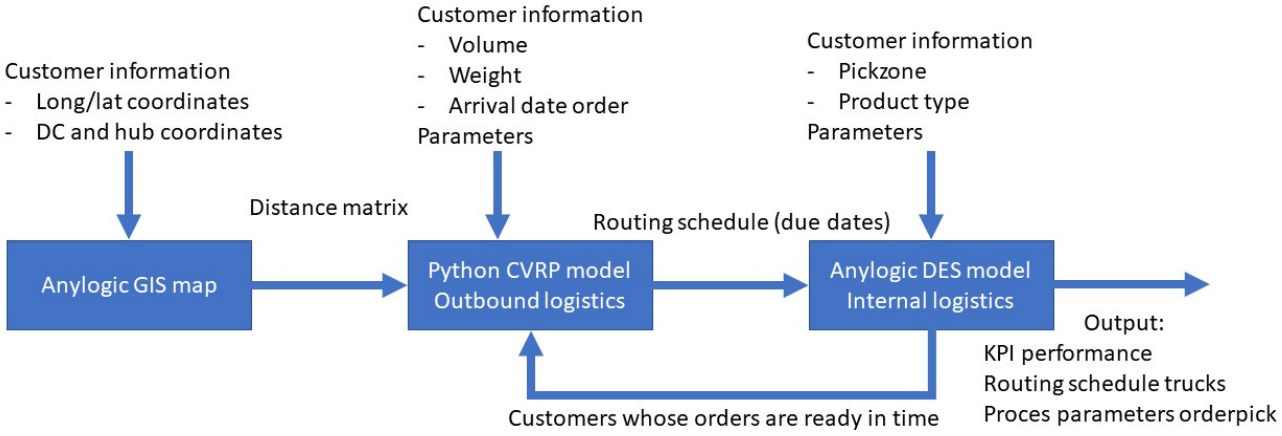


Figure 4: *Hybrid model overview with all the required inputs and outputs.*

The simulation results will be validated by comparing with the existing WMS algorithm and planning to see whether the proposed changes in the scenarios have a positive effect on key parameters such as total costs, labor hours and processing time.

In Burinskiene et al. (2018), a theoretical framework for waste avoidance is presented. This framework is called the waste pyramid which is shown in figure 5. Approximately 10% of the operations costs are located in the receiving process where case picking has 65% of the operations costs. In the case a specific pickzone is the bottleneck and changing the planning parameters is not enough to have most of the orders ready in time, further improvement is needed on a more specific level.

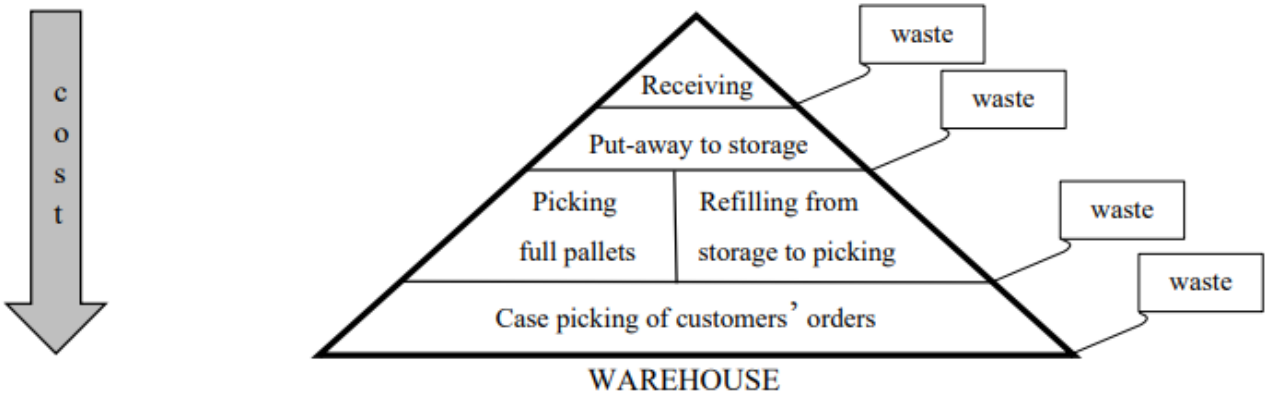


Figure 5: *Wastepyramid (Burinskiene et al., 2018) where the costs are the highest, the highest savings can be reached.*

One relevant scenario to reduce waste presented in Burinskiene et al. (2018) will be investigated in the simulation model of Stiho DC Utrecht. Picking multiple customer orders simultaneously instead of separately reduced the travel distance by more than 21.8% in the reference warehouse. This is called batch picking.

Furthermore, a scenario that will be tested is regarding the trucks routing schedule. As was stated in the problem section, not all orders are ready to be shipped at the time of departure. Hence, these orders can be rescheduled to a later truck who can make a detour in order to deliver the order on the agreed date with the customer. The definitive route will be created at moment of departure. The traveling salesman problem will calculate the optimal route after the additional order is included.

Furthermore, an optimal expedition deadline will be investigated in the simulation model to tackle the manual ranking problem by conducting a parameter variation study.

In consultation with the manager of the distribution center, a very average day is picked to perform the case study on to generate representative results. This is the 7th of October which is between the summer and the winter holidays and has an average number of orders.

1.4.2 Main question and scientific contribution

The main question will aid in solving the problem and reaching the goal. It is formulated as follows:

How can the logistics process be improved with hybrid simulation such that 99.5% of the orders are delivered on time to the customer and an FTR above 98% is achieved whilst having a similar or higher throughput of planned orders?

The service level of 99.5% is chosen such that a maximum of 0.5% of the FTR considered complaints is because orders are not delivered. Hence, there is room for other types of complaints considering FTR has to be higher than 98%.

The main scientific contribution of this thesis will be the hybrid model where Discrete Event Simulation is integrated with the traveling salesman problem. This has not been done before. In this specific case of Stiho DC Utrecht, the inbound logistics DES model will have as input the optimization of the outbound logistics. The TSP application of the outbound logistics considered is the capacitated vehicle routing problem (hereafter: CVRP).

2 Optimization of outbound logistics with CVRP

The first part of the hybrid model is the modelling of the outbound logistics which yields a routing schedule that serves as input for the internal logistics. The outbound logistics of Stiho consists of multiple trucks with trailers that visit customers to deliver customer orders. The trucks can combine orders in one delivery route until a maximum weight, volume and time is reached. Every customer has to be visited only once and the shortest route has to be found. Hence, the algorithm to model this is the capacitated vehicle routing problem which is based on the traveling salesman problem (hereafter: TSP) (Toth and Vigo, 2002). The VRP is a NP-hard optimisation problem. Therefore, it cannot be solved for large instances and a heuristic should be used to guide the process for searching for solutions and approximate a sufficient solution (Alewijnse and Hübl, 2021). The main difference between the current situation and the modelled situation is the consideration of all customers when setting the routes instead of pre selecting customers and creating the optimal route based on this.

In Mohammed et al. (2017), the K-Nearest Neighbor Algorithm (hereafter: KNNA) is used to effectively solve the CVRP by making use of the euclidean distance. The main benefit is that it provides a faster and more accurate result and it is able to solve huge problems with high computational complexity. KNNA are stochastic in decisions which makes them more robust compared to deterministic heuristics. The general idea of the K-nearest neighbor algorithm is a heuristic where the nearest unrouted node is added to a route until a constraint is violated and then a new route is initialized. The new route takes the nearest unrouted node from the depot. This is repeated until all nodes are routed (Rasku et al., 2019).

2.1 The capacitated vehicle routing problem

The mathematical formulation of the Capacitated Vehicle Routing Problem is based on the formulation in (Alewijnse and Hübl, 2021). The CVRP mathematical formulation is defined on a graph $G = (V, A)$ where $V = \{1, \cdots, n + w\}$ is the set of nodes and $A = \{(i, j) : i, j \in V, i \neq j\}$ is the set of edges. V consists of two subsets: the set of n customers $V_c = \{1, \cdots, n\}$ and the set of w hubs $V_h = \{n + 1, \cdots, n + w\}$. A fleet of vehicles, K, with each a maximum capacity in terms of weight W_k and volume Q_k. All vehicles are restricted to a maximum work time T that is based on a maximum distance. For every customer i, a non-negative demand in terms of weight w_i and volume q_i are given. The maximum number of customers visited in one route is indicated by M_k. The travel distance between node i and j is given by $c_{i,j}$. The decision variable for this problem is given by the binary variable $x^k_{i,j} \in \{0, 1\}$, which is equal to 1 if vehicle k travels from node i to node j, and 0 if otherwise. The CVRP can be modeled by a Mixed Integer Linear Problem formulation and is give below.

$$\min \sum_{i \in V} \sum_{j \in V} \sum_{k \in K} x_{i,j}^k c_{i,j} \tag{1a}$$

$$\text{s.t.} \sum_{i \in V} \sum_{k \in K} x_{i,j}^k = 1 \quad \forall j \in V_c \tag{1b}$$

$$\sum_{j \in V} \sum_{k \in K} x_{i,j}^k = 1 \quad \forall i \in V_c \tag{1c}$$

$$\sum_{i \in V_c} \sum_{j \in V} w_i x_{i,j}^k \leq W_k \quad \forall k \in K \tag{1d}$$

$$\sum_{i \in V_c} \sum_{j \in V} q_i x_{i,j}^k \leq Q_k \quad \forall k \in K \tag{1e}$$

$$\sum_{i \in V} \sum_{j \in V} c_{i,j} x_{i,j}^k \leq T \quad \forall k \in K \tag{1f}$$

$$\sum_{i \in V_c} \sum_{j \in V} x_{i,j}^k \leq M_k \quad \forall k \in K \tag{1g}$$

$$x_{i,j}^k \in \{0,1\} \forall i \in V, \forall j \in V, \forall k \in K \tag{1h}$$

Objective function eq. (1a) aims to minimise the total distance needed to visit all customers. Constraint eq. (1b) ensures that all customers are visited exactly once by only on truck. Constraint eq. (1c) ensures that this truck also leaves the customer once. Constraint eq. (1d) and eq. (1e) bound the weight and volume of all customers to the maximum of the truck respectively. Constraint eq. (1f) ensures that the maximum distance one truck can drive is not violated. Constraint eq. (1g) ensures that a maximum number of customers is visited in one route. Constraint eq. (1h) defines the binary variable that can either be equal to 1 or 0 depending on whether the truck travels along that arc on the graph. Solving this model will give a network with a number of routes and the required number of trucks to visit all customers in the shortest possible distance given the constraints.

As explained, the CVRP has to be solved with a meta heuristic and KNNA is chosen to do so. With KNNA, the nearest unrouted node is added to a route until a constraint is violated and then a new route is initialized. This process is repeated until all nodes are routed (Rasku et al., 2019). In general terms, a node is classified based on the classification of the majority of its neighbors. Before a classification is made, the distance must be defined to know who is the nearest neighbor. The distance between two nodes is obtained from a distance matrix that has to be generated before the algorithm can be executed. For the KNNA, the value of K has to be determined. The K is the parameter that determines how many neighbors are considered to add to the route and thus how many routes are constructed in parallel (Rasku et al., 2019). Several values of K have been tested and a K of 1 yielded the best results in terms of minimum distance en minimum number of trucks. Hence, this value is chosen. For a K of 1, the single closest neighbor is added to the route.

2.2 Model of the case study: set-up and assumptions

The Capacitated Vehicle Routing Problem is modeled with the KNNA based on an adapted version of the VeRyPy algorithm that offers multiple heuristic solutions to the CVRP (Rasku et al., 2019). The VeRyPy algorithm only took weight into account as constraint thus the model has been expanded to take volume into account as well. The code is attached in appendix A.1.1.

The data that is necessary to perform the outbound logistics optimization for a given day are based on the customer orders. The routing schedule is always created on a daily basis. The customers and their locations in longitudinal and latitudinal coordinates, the weights and volume of the orders, and the

maximum weight and volume of the trucks are the considered parameters. Since the data is not completely accurate, some data manipulation has to be performed. As explained in the problem section, some special products have no weight and volume information. This occurred for 5.5% of the stops the past year. The weight and volume have been set to the median weight and volume of the truck as an estimation. Furthermore, the trucks start their day from either the DC in Utrecht or from a hub. Stiho mainly uses two hubs in Hoogeveen and Vlijmen where the trucks are going to in the evening and departing in the morning to reduce the travel time during the day. Hence, being sooner at the customer. The coordinates of all planned customers, the hubs and DC Utrecht are converted to a distance matrix by making use of the GIS map functionality of Anylogic. This yields the distance on the quickest road which is more accurate than if the straight line distance was taken. The distance from DC Utrecht to the hubs has been set to 0 in the distance matrix as the distance from DC Utrecht to the hubs driven in the evening before is not considered in the maximum distance the truckdriver can drive during the day. As a driver cannot drive longer than his working day, a maximum traveling distance and a maximum number of stops is assumed. This is not a very strict requirement since overtime is common in the trucking industry. The fleet of trucks is a flexible pool since the trucks are from transportation companies and the number can be altered based on demand. Hence, this is not taken into account as a constraint.

Then, the CVRP algorithm in Python is ran with as input the distance matrix, weights and volume per customer. This yields a routing sequence given the constraints. For every route, the starting point, customers and finishing point is indicated. All trucks start and end in DC Utrecht and if a hub is used, it is entered as the first node to be visited. The routing sequence can be compared to the routing sequence of the planning software to observe whether the CVRP solution is shorter than manual routing.

In consultation with the DC manager of Stiho and based on data from the past year, the following parameters in table 1 are set to resemble an average truck and feasible routing. However, the maximum distance is currently set longer than feasible since not all hubs have been included. The maximum number of customers to be visited in one route heavily correlates with the distance between two customers. The number set in the table below is smaller than the maximum number of customers visited in one route occurring in the data so it compensates for the increased maximum distance.

Variable	**Symbol**	**Value**
Maximum weight per truck	W_k	18,000 kg
Maximum volume per truck	Q_k	30 m3
Maximum distance	T	600 km
Maximum number of customers	M_k	15

Table 1: *Parameters for the CVRP algorithm. One type of average truck is considered en the maximum number of customers is set arbitrarily. The maximum distance is set very large to compensate for the missing small hubs. If this distance is set smaller, there are several routes where only one customer is visited with very small load and this is not happening in reality.*

In figure 6 and figure 7 are respectively the capacity in terms of maximum weight and volume per truck that Stiho could use summarized. The maximum weight and volume in table 1 corresponds with the majority peak in the figures. Some trucks have a larger capacity than used in the model and some trucks have less capacity than used in this model. A sensitivity analysis on the effect of taking other capacities as parameters is included later on in this chapter.

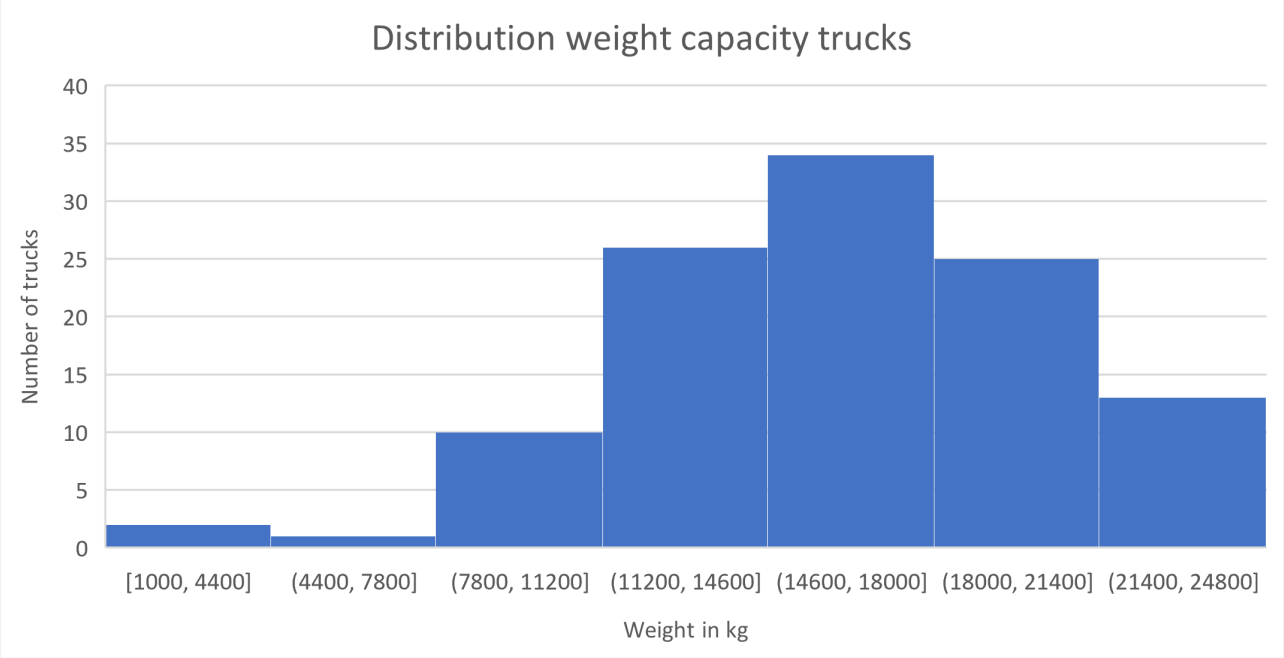


Figure 6: *Division of the maximum weight in kg of trucks Stiho could use.*

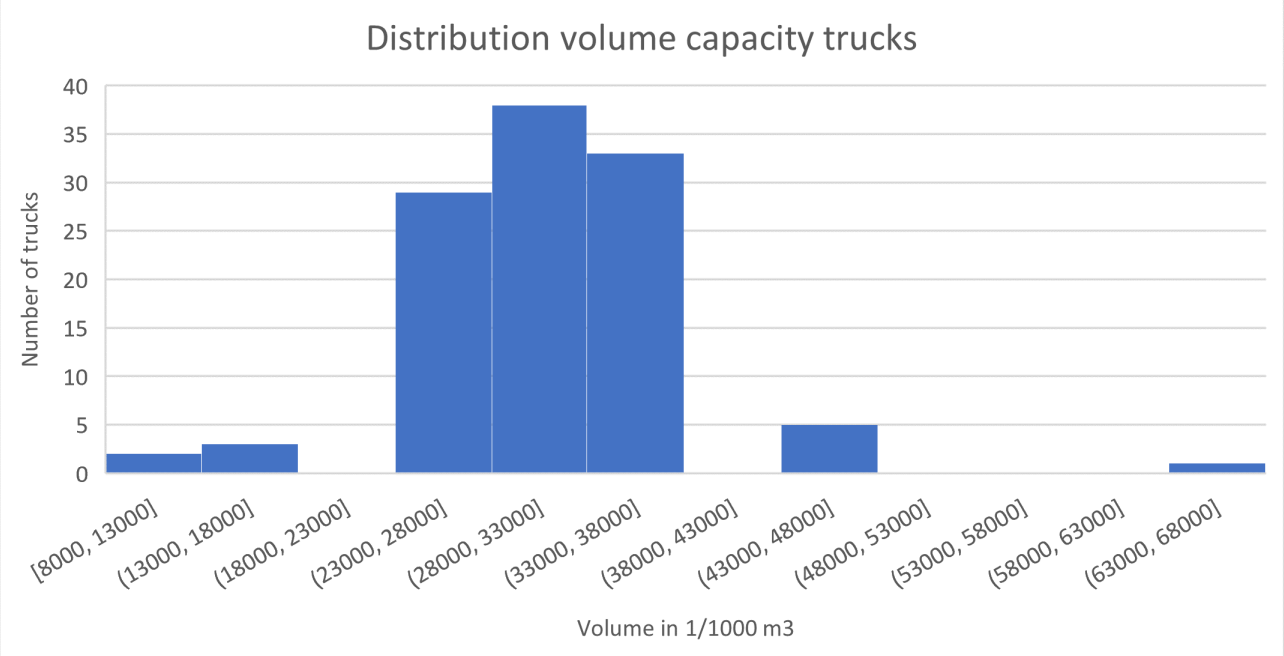


Figure 7: *Division of maximum volume in m3 of trucks Stiho could use.*

The main drawbacks of the model compared to reality are that there are no time delivery windows considered. In reality, some customers indicate that they need their order before a given time on a day. As it occurs not very often, it is not considered in the routing algorithm. Furthermore, it is assumed that all trucks have the same capacity in the model where in reality there are multiple types of trucks.

2.3 Model of the case study: results

The algorithm is ran in python with the parameters of table 1, a K of 1 and the customer data of the 7th of October 2022. The total distance that was traveled that day was 9396.5 km. When aggregating customer orders to one customer order per customer address, 328 unique customers remain. A total distance of 8089 km is driven by 37 trucks resulting a reduction in distance of 14%. On average the trucks reach 54.91% of the maximum weight and 71.08% of the maximum volume. An overview of the percentage weight and volume per route is indicated in figure 8. It can be observed that for five trucks the volume reaches a value higher than 95%. These trucks need special attention when loading. Theoretically, all orders should fit in these trucks. However, due to improper stacking of the orders, it can be the case that in these trucks not all orders can be shipped. On the original data, ten orders did not fit into six different trucks.

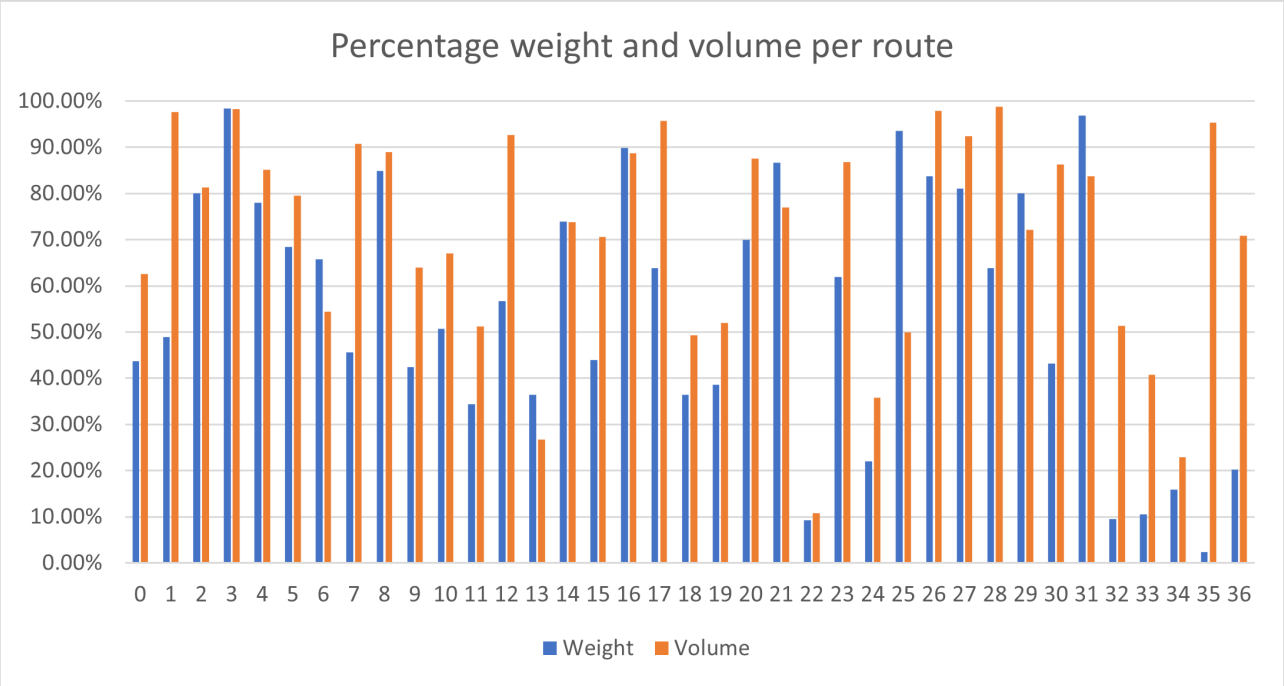


Figure 8: *Percentage of maximum weight and volume that is reached per truck. As one can observe, there are large differences and some trucks leave with less than 50% of the capacity where others are almost full.*

A sensitivity analysis on the maximum weight and volume per truck parameters is performed to find the magnitude in which the arbitrarily set parameters play a role in the results. The results can be found in table 2. A counter-intuitive result is the reduction in total distance when the maximum weight is reduced. Up to route 23, the resulting routes are similar but then the routes are slightly different. This can be explained by the greedy behavior of the k-nearest neighbor heuristic where local optimums are considered.

Table 2: *Sensitivity analysis on the effect of the maximum weight and volume per truck parameters in CVRP algorithm on the total distance. The base case is indicated in bold.*

Max Weight \Max Volume	28.5 m3	**30 m3**	31.5 m3
17,000 kg	8,096 km	7,653 km	7,583 km
18,000 kg	8,096 km	**8,089 km**	7,783 km
19,000 kg	8,034 km	8,059 km	7,781 km

If the maximum volume is reduced by 5%, so Q_k is equal to 28.5 m3, leaving more empty space in a truck and thus increasing the chance that all orders fit in the truck, one additional truck is required. The

total distance driven is 8,096 km which is only 7 km more than in the previous experiment. However, as an additional truck is required, the costs will rise significantly more than the slight increase in distance. However, this case provides a more robust result as the chance increases that all orders are shipped and a trade-off should be made if the additional truck weights up to the lower chance on customer claims for not delivered orders. As in two trucks of figure 8 the volume percentage is very close to 100%, an additional truck to be used seems a wise decision.

When closely examining the results of the route, in some cases the truck does not return to the hub but the DC Utrecht as this distance is shorter and Stiho makes use of more hubs than modelled. Furthermore, in some cases the 15 stops could be too much if there are several long stops combined in one route. A stop can be longer if the products are delivered in the city centre of Utrecht where the truck cannot come very close the the construction sites. Hence, it could be the case that some routes have to be split up again and a couple of additional trucks have to be added. Therefore, it is expected that the distance saved is less than the projected 1300 km but a gap this large seems promising to look into further. Hence, in further research, the model could be refined more to approximate reality even further.

2.4 Managerial advice to Stiho: from current to modelled situation

In the current situation, the planning department puts the orders in a truck and the PTV planning software determines the optimal route based on that selection of orders. In the modelled situation, the CVRP algorithm considers all orders and determines first an optimal division of the orders per truck based where the overall distance is minimised given the weight and volume constraints. Furthermore, the optimal route per truck also results from the total minimum distance. By selecting from a larger set of destinations, the total distance of all routes becomes intuitively smaller as "neighbors" can be more optimally combined.

The main reason why Stiho is planning according to the current situation is that not all weights and volumes are known. This can be solved in two ways. Either it becomes required for the commercial department to enter the correct weight and volume or an approximation of it when placing an order or an approximation of the weight and volume is performed after the order is in the system and just before the planning algorithm is executed. If this is done properly, PTV is able to calculate the optimal route in the same fashion as the CVRP algorithm. The maximum weight and volume can be set to 90% of the capacity to deal with a potential inefficient stacking method or other inefficiency such that the maximum weight or volume is not causing any problems anymore and the chance that all orders fit in the truck increases significantly. Returning goods orders can still be modeled with no weight and volume as they are entering a near empty truck.

The benefit for Stiho is twofold. Not only will the proposed solution reduce the total distance covered which reduces the trucking costs but also the planning department can be reduced from five to two employees. These employees can be deployed elsewhere.

3 Optimization of internal logistics

The second part of the hybrid model is the optimization of the internal logistics. The obtained routing schedule of the optimization of the outbound logistics serves as an input for the DES model of the internal logistics as it determines the due date of when the order should be available in the expedition hall to be shipped to the customer. Recall from figure 4 that additionally the pickzone(s) an order flows through are required. These are obtained from the database of the WMS software for the simulation purposes. In reality, they are obtained from the customer order by the connection with the material that is bought.

3.1 DES model of internal logistics in Anylogic

The orderpick process as mapped in figure 2 is converted to a DES model in Anylogic. The seven most common orderpick processes are modelled individually and a eight process is added for all miscellaneous orderflow. To resemble the algorithm that determines the ranking of Stiho, a backward scheduling algorithm is used in Anylogic. The orderpick process is modelled as a queue with delay blocks for the picking and conveyer belt and batch blocks to simulate the carts. The orders are queued based on their due date and the order with the earliest due date is processed first. Furthermore, if the product of the preceding order is the same, the priority is increased and the processing time to pick the next order is halved since the it can be picked in less time. The orderpicking and transportation times are modelled as a stochastic distribution based on real data from the past month. The order arrives in the model and is split into a customer order and a DC order. The DC order flows through the orderpick process and takes information about the pickzone to be used and due date with it. This determines the flow through the system. The due date is based on the priority of the route gained from the outbound logistics optimization. The internal due date coupled to the DC order is earlier than the customer order to have some room for error. The optimal value for this is yet to be determined but is currently 50 minutes within Stiho. The pickzone to be used is coupled to the customer order. After the due date is reached, the customer order and DC order are matched in the matchblock which checks if the DC order was finished in time to enter the truck on its journey towards the customer. The batch sizes are based on the observed batch sizes in the DC of Stiho. To deal with the last batch that can be less than a full batch size, the model checks every 10 minutes if all orders of that pickzone are finished and then the batch size is set to the current amount on the cart. The number of employees are determining the orderpick tasks that can be performed parallel. It is assumed that if an order is ready at the due date and thus is present at the FGI inventory in the expedition hall, that it will enter the truck. An overview of the Anylogic DES model is shown in figure 9. Further screenshots of the Anylogic model are attached in appendix A.2. The customer orders with due date and pick zone arrive at cust_orders. They are split to a DC order and customer order at the split block. The customer order is waiting in delay_due_date until the due date is reached. The DC order is released into the orderpick process. Here it goes through one of the eight pickzones (PZ) where the orderpick process is modeled by a combination of delay and queue blocks. The orders transport depending on the pickzone through a conveyer belt, modeled as delay block, or cart, modeled as combination of batch and queue blocks. Then the order is exiting the orderpick process and entering the expedition. At the match_orders block is checked if the order is picked at the due date. The parameters and the variables in the top right of the figure are used for the modelling and are determined based on data. The purple blocks are gathering data from the DES simulation for statistical purposes.

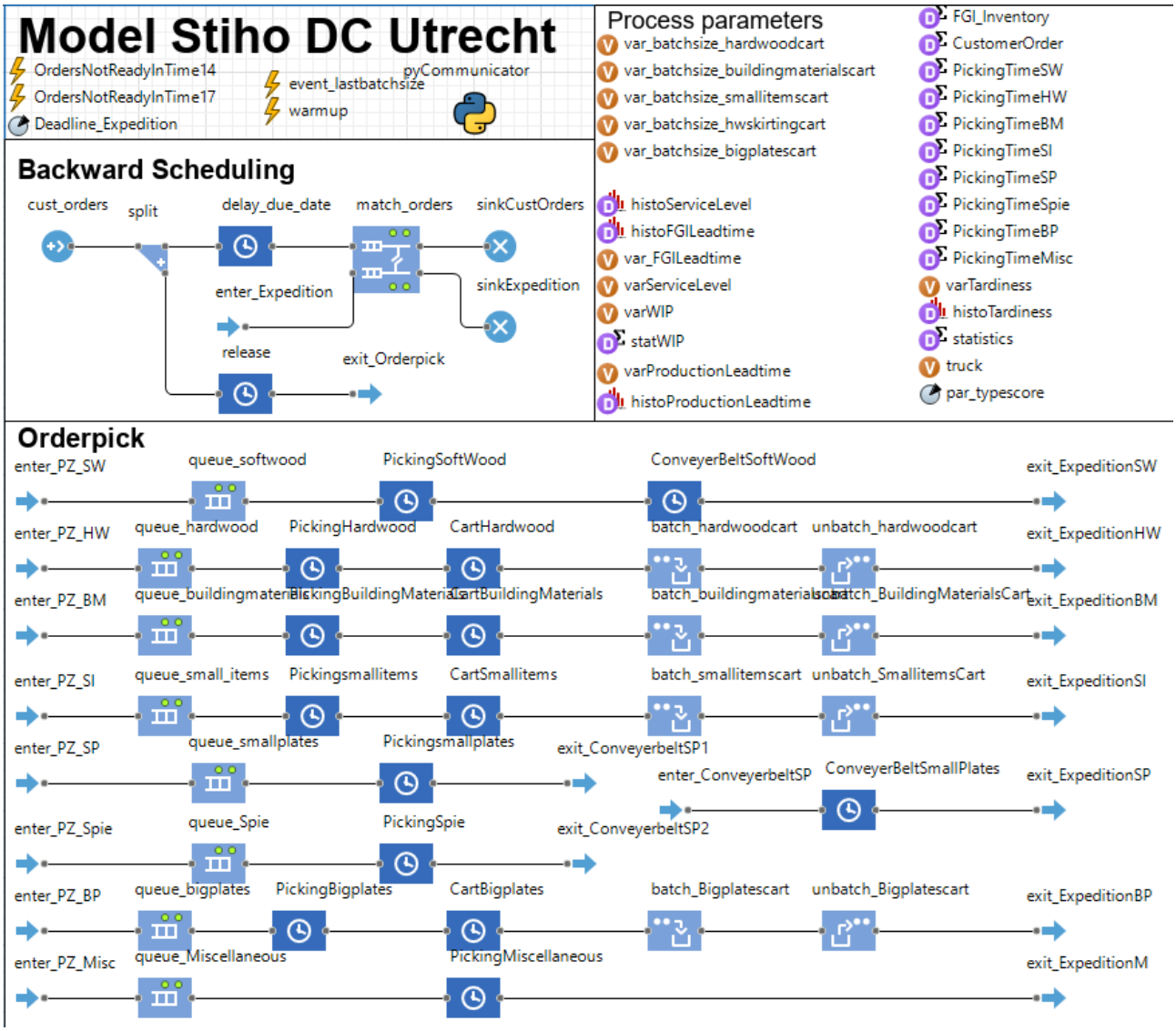
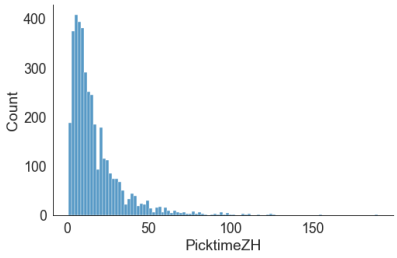


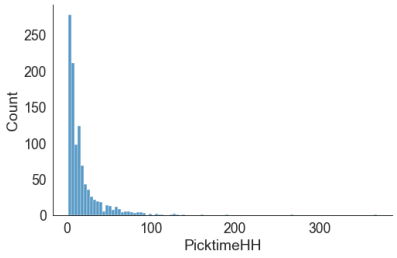
Figure 9: *Anylogic model of the internal logistics of Stiho.*

3.2 Data fitting for distribution of parameters

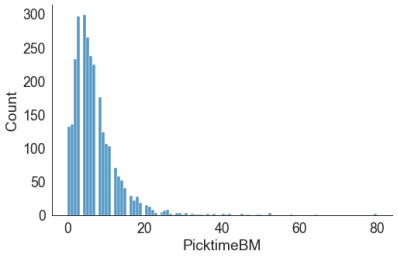
In order to calculate a reliable distribution for the picktimes and the transportation times in the DC, data from the past month is used. By means of a query in the WMS software Locus, the timestamps of the movements of all ordercarriers have been retrieved. Due to some manual errors such as forgetting to scan the final location and progressing through the picking steps all at once instead of indicating its ready when its actually ready, the data has to be cleaned. A boxplot is created in Microsoft Excel to analyse the data outliers. As it is deemed infeasible to pick an order in 0 minutes, those entries are removed from the sample. Furthermore, all times larger than 200 minutes are also removed from the sample. The resulting distribution of the picktimes can be viewed in figure 10. The Python package Fitter from Cokelaer is used to fit a statistical distribution over the data (Cokelaer, 2022). The Python code is attached in appendix A.1.2. The ten most common distributions are fit. This yields a summary of fitting results and the distribution that has the smallest sumsquare error and is available in Anylogic is selected. The smaller the sum square error, the more accurate the forecast. The resulting orderpick time distributions are included in table 3. The distributions for the order picking and transportation times are entered in the respective blocks in Anylogic.



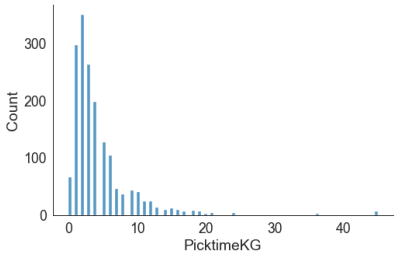
(a) Dist. picktimes softwood



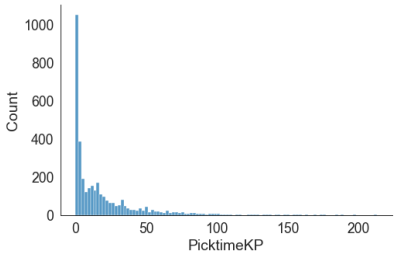
(b) Dist. picktimes hardwood



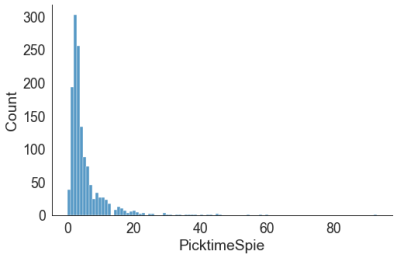
(c) Dist. picktimes building mat.



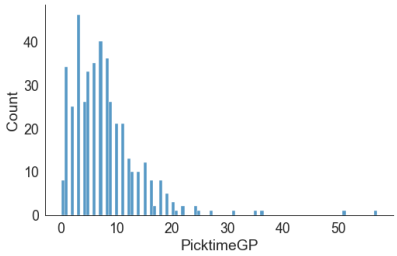
(d) Dist. picktimes small items



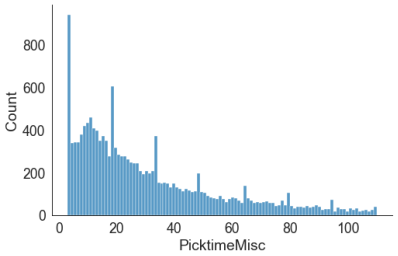
(e) Dist. picktimes small plates



(f) Dist. picktimes spie



(g) Dist. picktimes big plates



(h) Dist. picktimes misc.

Figure 10: *Distributions of the picktimes of the pickzones*

Table 3: *Stochastic distributions of the picktimes per pickzone.*

Pickzone	Distribution type	Loc parameter	Scale parameter	Shape parameter
Softwood	Lognorm	-0.9988	12.9676	0.7959
Hardwood	Lognorm	1.4153	8.3024	1.2504
Building Materials	Cauchy	5.1274	2.5041	-
Small Items	Cauchy	2.6580	1.4338	-
Small Plates	Exponential	0.0	18.8972	-
Spie	Cauchy	2.6983	1.3734	-
Big Plates	Gamma	-0.2973	4.4045	a = 1.8388
Miscellaneous	Exponential	3.0	27.6985	-

In order to validate the distributions, the DES model is runned with 600 orders split over the pickzones in the ratio they occured the past month. The mean simulated orderpick times are measured and compared to the real mean productivity and are found to be quite similar.

The average process times could be a good fit in Locus to substitute the current estimate of 3 minutes as they better approach reality. The processing time values are in table 4. Furthermore, if Locus allows this value to be a function, it could be a function of number of products in the order. The more units that have to be picked, the longer the processing time.

Table 4: *Mean processing time per pickzone*

Pickzone	Mean processing time (min)	Standard deviation
Softwood	16.8	15.8
Hardwood	18.3	25.0
Building Materials	7.0	6.4
Small Items	4.3	4.6
Small Plates	17.9	24.1
Spie	5.0	6.5
Big Plates	7.8	6.3
Miscellaneous	30.7	24.8

3.3 Obtaining initial planning parameters from simulation

The parameters that need to be selected are the number of employees per pick zone. A configuration in parameters is examined in multiple replications with different stochastic values. Increasing the number of replications increases the number of observations and thus increases the accuracy of the results. However, more replications take longer computation time. Hence, a trade-off between accurate results and computation time has to be made. From figure 11, the confidence interval of 10 replications falls entirely within the 20 replications and the average is less than 0.1% apart. Hence, 10 replications for all experiments are performed as its results are deemed accurate.

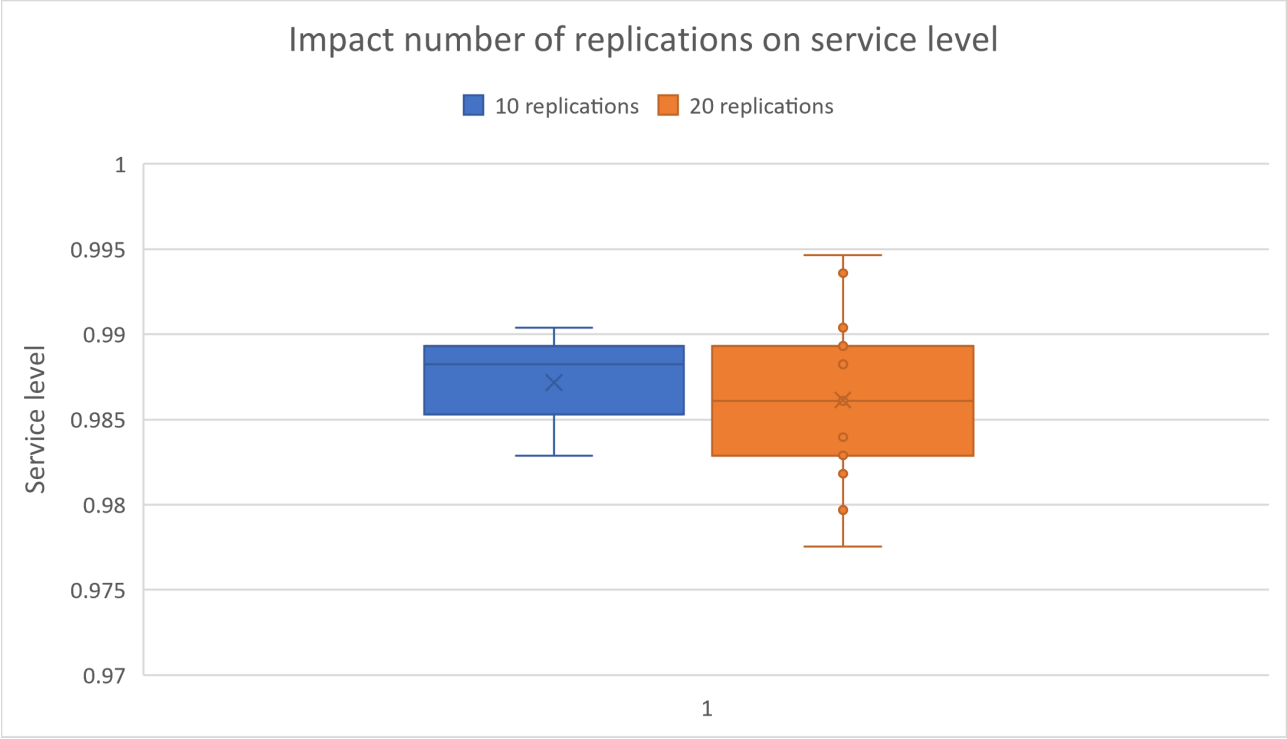


Figure 11: *Impact number of replications on service level*

A parameter variation has been conducted with varying number of employees and 288 experiments have been performed with each 10 replications with different stochastic values. The optimal number of employees with a similar service level as the 7th of October is in table 5. At the 7th of October, 11 orders were not delivered to the customer at the agreed due date. Hence, a service level of 97.9% was reached. Not all employees selected here are occupied the whole day. Hence, the number of employees is less than indicated in the table as some employees could be used in multiple pick zones. Using more employees

increases the service level only marginally.

Table 5: *Optimal number of employees per pick zone that will be used for further experiments.*

Employees Softwood	5
Employees Hardwood	2
Employees Buildingmaterials	3
Employees Small items	1
Employees Small plates	5
Employees Spie	1
Employees Big plates	1
Employees Miscellaneous	4

Other planning parameters are examined in hybrid setting in the next chapter.

4 Hybrid model to reschedule tardy orders

In this section, the hybrid model will be explained and the application of the case at De Stiho Groep will be examined.

4.1 Hybrid model explanation

Recall in figure 12 the hybrid part of the model overview. As it is the main goal to increase FTR by ensuring that 99.5% of the planned orders are delivered to the customer on the agreed date, more orders should be delivered on the agreed date. To achieve this, the hybrid model enables a rescheduling of the orders for a later due date if they are not ready at the earlier due date. In this case, the customer receives his order still on the agreed date and FTR performance increases.

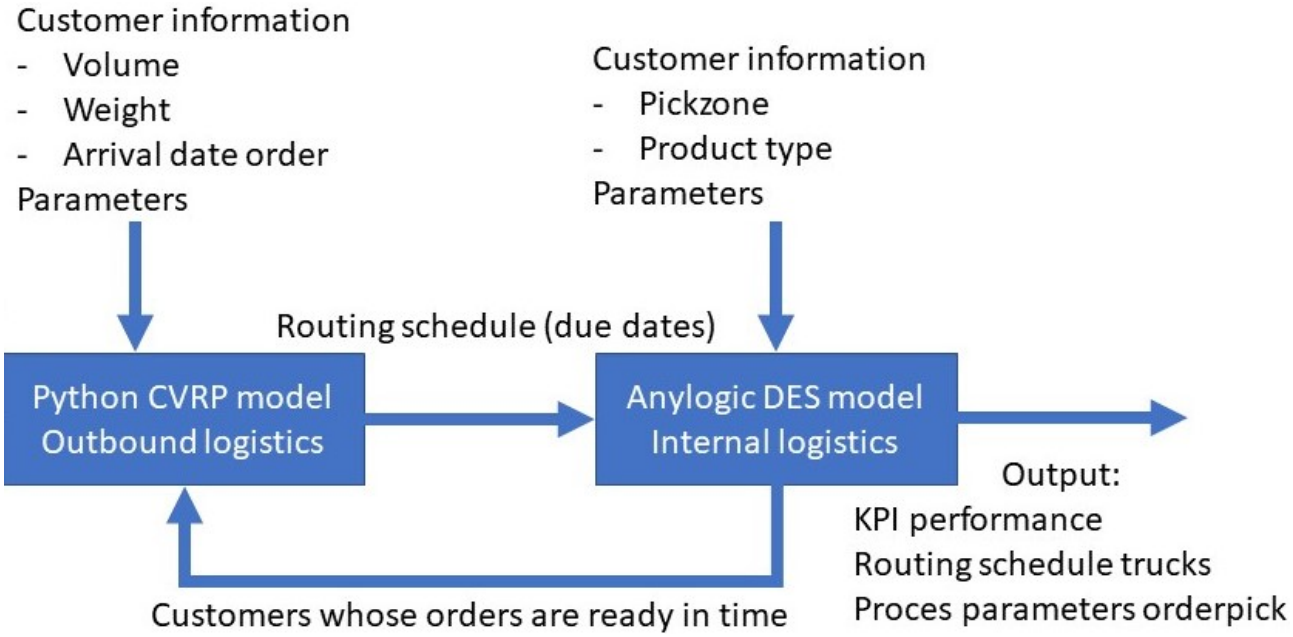


Figure 12: *Recall from figure 4 the hybrid part of the model overview. The hybrid part is if customers whose orders are not ready in time can be rescheduled to a truck that departs later in the day such that the customer still receives the order on the agreed date.*

With the Anylogic package Pypeline, an Anylogic python integrator is added to the Anylogic DES model of the internal logistics. The Pypeline integrator sends data to a python script and receives the results back. In this case, a list of orders that is not ready at 2 pm is exported to excel and then the python CVRP model script is activated. The routing schedule for the trucks departing later than 2 pm is recalculated for the customers that are not visited with the trucks departing at 2 pm and the customers whose deadline at 2 pm were not met. Hence, a new routing schedule including the tardy orders is created. The hybrid model is ran once per day at 2 pm. The new due dates originating from the new routing schedule are sent to Anylogic with the customer ID to match the active agents in the DES model. Hence, the due dates of the orders that are not picked yet are updated and a new ranking is created for the order in which the orders are picked.

4.2 Hybrid model verification

Running the simulation model takes approximately one minute per simulation run. In order to create reliable results, each parameter configuration is ran with 10 replications. In these replications, the stochastic values are different, hence yielding a different result. However, the ten different seed values are constant during all simulations making it possible to compare different parameter settings without

stochastic interference on the results.

First the model results are validated by a sensitivity analysis to see if the real number of tardy orders is within the confidence interval. A confidence interval with significance value alpha of 0.05 is applied in all upcoming simulations. Hence, in 95% of the cases, the mean will be within the range of the sampled mean plus and minus the confidence interval. As can be observed in table 6, the real number of tardy orders (11) is within the confidence interval of the mean tardy orders of the simulation. Hence, the simulation can be used to generate valid results.

Table 6: *Confidence interval analysis on the average tardy orders of the simulation of the logistic system on the 7th of October 2022. For 10 replications with different stochastic values, an average of 12.2 orders was tardy with standard deviation of 2.32. With an alpha of 0.05, the confidence interval is 1.66.*

Average	Count	Stdev	Confidence interval
12.2	10	2.315167	1.656170974

4.3 Hybrid model results

Different simulation scenarios have been examined in the upcoming sections in order to find a solution that improves the service level. In section 4.3.7, the simulation results are compared to see which scenario is the best solution to the problem.

4.3.1 Effect of rescheduling of tardy orders on service level

The first simulation that is performed is the hybrid simulation where the effect of rescheduling the early tardy orders to a later truck on the service level is examined. When the simulation is performed with 10 iterations, on average 11 orders are tardy at 2pm, 1 order is tardy at 5pm and occasionally an order is tardy at 8pm. This can be observed in table 7.

Table 7: *Average tardy orders of 10 simulation iterations per time window*

Time	Tardy orders
Tardy orders 2pm	11.5
Tardy orders 5pm	0.6
Tardy orders 8pm	0.1
Tardy orders 11pm	0

In the hybrid model, the 11 tardy orders at 2pm are rescheduled to a truck leaving at 5pm, 8pm or 11pm. According to the simulation, all orders can be rescheduled without the need for an additional truck. This is possible since most trucks initially have not reached full capacity as can be seen in figure 8. Hence, the rescheduling reduces the number of tardy orders from 12 to only 1. This yields a service level of 99.8% which means that the goal of a service level of 99.5% is reached. If the orders were not rescheduled to another truck as is the case currently, a service level of 97.7% is reached. As can be observed, most tardy orders occur at the routes where the truck departs early. This can be explained by customers entering an order at 12pm at a destination where the truck leaves to at 2pm. This time window is very short and if there are multiple orders for the same pickzone or very large orders, the chance for tardiness is significantly larger than in the case there is more time in between placing the order and truck departure.

4.3.2 Effect of batch picking on performance order picking

If two similar products are picked in sequence, the pickingtime is halved to simulate the reduction in workload. A ranking based on picking multiple products of the same type in sequence is called batch picking. Normally, the ranking is based on due date but an additional score can be added to stimulate batch picking over due date. This score is called TypeScore in the model. If this score is larger than the

difference in due date, orders with a later due date but with the same type of product are picked first. This reduces the overall picking time and increases efficiency. However, the number of tardy orders in the early trucks more than doubles from an average of 11 to an average of 26. Furthermore, 7 orders are tardy at 5pm and 2 orders are tardy at 8pm and 1 order is tardy at 11pm. Although efficiency increases, the service level reduces to 93.2% or 98.1% for the case without the rescheduling and including rescheduling at 2pm respectively.

As can be observed in table 8, on average 1,35% picking time is saved if the current and next delivery window are considered for batch picking. The ranking is calculated as due date in minutes from 6am (e.g. 2pm equals 420) minus the TypeScore which is scored if the order has the same product type as the preceding order. The effect of different values of TypeScore has been examined. As the due dates are 180 minutes apart, a TypeScore of 200 makes the orders of the same type but with a later due date more important than orders with the earlier due date. As can be observed, some pickzones respond better than others. This could be caused by the stochastic behavior.

Table 8: *Analysis on the effect of batch picking on the average picking time per pick zone. All times are in minutes.*

TypeScore	**0**	**50**	**200**	**400**	**800**
Avg PicktimeSW	12.3	12.3	12.5	12.7	12.8
Avg PicktimeSP	14.4	14.4	14.3	14.2	14.4
Avg PicktimeHW	13.9	13.9	12.8	13.5	13.2
Avg PicktimeBM	6.0	6.0	5.5	6.0	6.0
Avg PicktimeBP	5.0	5.0	4.7	5.7	5.2
Avg PicktimeMisc	23.0	23.0	24.0	24.4	23.2
Avg PicktimeSI	2.8	2.8	3.0	2.8	2.9
Avg PicktimeSpie	4.1	4.1	3.6	3.9	4.1
Avg Picktime	**10.2**	**10.2**	**10.0**	**10.4**	**10.2**
% change	0.00%	0.00%	-1.35%	2.12%	0.38%

In table 9, the effect of TypeScore on tardiness is examined. The higher the TypeScore, the more tardy orders occur. This can be explained by the change in the order of picking which lets orders from later delivery windows be picked before the earliest due date. Hence, especially in the earlier due date windows, more orders are tardy. However, the tardy orders of 2pm are rescheduled to another truck in the hybrid model so the number of tardy orders excluding 2pm is still lower than in the current situation. Hence, a trade-off can be made between shorter and thus cheaper picktimes or more happy customers that got their orders delivered on the agreed date.

Table 9: *Analysis on the effect of batchpicking on tardiness of orders. The average number of orders that is tardy per delivery window is indicated. With the hybrid model, the tardy orders at 2pm are rescheduled to another truck that departs later.*

TypeScore	**0**	**50**	**200**	**400**	**800**
Tardy orders 2pm	11.5	11.5	20.1	25.8	26.7
Tardy orders 5pm	0.6	0.6	3.8	7.4	6.5
Tardy orders 8pm	0.1	0.1	0.5	2.4	1.6
Tardy orders 11pm	0	0	0.3	1.2	1.2

4.3.3 Effect of an earlier planned due date

In the current situation, the planned due date of when the order has to be present at the expedition hall is 50 minutes before departure of the truck. It is examined if increasing this 50 minutes has an effect on the number of tardy orders. It has been found that there is no effect on the number of tardy orders

since the planned due date of all orders is adjusted and it has no influence on the ranking in which the orders are picked. Hence, it is not advised to change the current settings regarding to difference between planned due date and original due date.

4.3.4 Effect of increasing the number of employees

An other option to be investigated is if the increase in the number of employees as was suggested in the beginning of the design project by some company employees would reach the desired result. In the current scarce job market and with the additional wage costs, this is not a desired option. But analysing this case provides a comparison of the impact of the hybrid rescheduling of early tardy orders which additional costs are negligible small because the number of trucks remains the same. As becomes apparent from table 10, increasing the number of employees quickly reaches a maximum service level for order lines of 99.1% which is 0.4% higher than the initial configuration. Hence, this is not a solution which solves the problem. The maximum service level of 99.1% could be explained by some large orders that are entered just before noon and have to be ready at 2pm. Furthermore, it can be the case that the transportation time is too long for these orders or that they have to wait until their cart reaches the required batch size. If the total orderpick and transportation time exceeds this window of two hours which can be the case for some large orders, the order will never be finished in time.

Table 10: *Analysis of the effect of increasing the number of employees per pickzone on the service level of order lines. The service level of order lines is different from the service level of orders as orders can be split into multiple order lines. CI indicates the confidence interval*

Employees Pickzone	**Initial number**	**Increased number**	**Increased number**
Employees Softwood	5	8	10
Employees Hardwood	2	4	10
Employees Buildingmaterials	3	4	10
Employees Small items	1	3	10
Employees Small plates	5	7	10
Employees Spie	1	3	10
Employees Big plates	1	3	10
Employees Miscellaneous	4	8	10
Service level order lines	98.7% (CI 0.178)	99.1% (CI 0.145)	99.1% (CI 0.145)

4.3.5 Effect of an earlier order placing deadline

Currently, orders placed before noon have to be delivered next day. In the case of the trucks that depart early, this might cause problems for the orders that are placed very late in this time window. In this subsection, the service level of the base case where orders can be placed until noon is compared with a case where orders for next day delivery can only be placed until 11am. The initial order line service level is again 98.7%. The service level decreases to 98.5%. This can be explained by a different queue as more products have the same rankingscore where some order lines requiring longer picking time are in the same cart making more orders miss their due date.

If the orders are only accepted until 11am and more employees are working, the service level can improve to 99.2%. This is still not the desired level. Furthermore, from a commercials perspective, the longer next day delivery orders are possible, the better the brand position for the customer. Hence, this is not a desired path to proceed.

4.3.6 Effect of decreasing batch size transportationcarts

As became clear from the previous sections, so far only the hybrid model with rescheduling of the tardy orders to a later truck yielded the required service level result. As no further improvement in service level was achieved with more employees, the processing time is expected to be too long. As the processing time also is depending on the transportation time with as major impact the waiting time until the batch is

full, a decrease in batch size of the transportation cart is also examined. The batch size is halved and the simulation is ran. The results are summarized in table 11. As one can observe, a service level improvement of 0.2% can be achieved if the batch size is halved. This means that it costs twice as much time to bring the carts to the expedition hall which reduces productivity. In the case that there are more employees, a maximum service level of 99.3% can be achieved. This is still not enough to satisfy the required service level of 99.5%.

Table 11: *Analysis on the impact of a smaller batch size and a combination with more employees and a smaller batch size in the transportation carts on the service level. CI stands for confidence interval.*

Pickzone	**Initial batch size**	**< batch size**	**< batch size and > employees**
Softwood	1 (conveyer belt)	1 (conveyer belt)	1 (conveyer belt)
Hardwood	8	4	4
Buildingmaterials	5	3	3
Small items	30	15	15
Small plates	1 (conveyer belt)	1 (conveyer belt)	1 (conveyer belt)
Spie	1 (forklift)	1 (forklift)	1 (forklift)
Big plates	8	4	4
Miscellaneous	1 (forklift)	1 (forklift)	1 (forklift)
Service level order lines	98.7% (CI 0.178)	98.9% (CI 0.185)	99.3% (CI 0.201)

If four additional employees are deployed to continuously drive the cart, hence set all batch sizes to 1, an average service level of 99.5% (Confidence Interval 0.157) has been achieved. In some stochastic seeds with low processing times, a service level of 100% has been reached. Hence, the required service level has been reached. The remaining orders that are tardy are orders that took longer to be picked and transported than the time between order arrival and the due date. Therefore, to deliver these orders on the agreed date to the customer, very large orders that require long processing times should not be placed just before noon for trucks that depart at 2pm.

4.3.7 Comparison scenarios

In figure 13 the service levels from the analysed scenarios in the previous sections are summarized. As can be observed, the rescheduling of tardy orders to a later truck, which is the main purpose of the hybrid model, provided the best solution in terms of service level to the problem. Also the entire confidence interval with alpha 0.05 of this solution is above the confidence interval of the other solutions. Hence, it can be concluded that for all simulations, the resulting service level is the best.

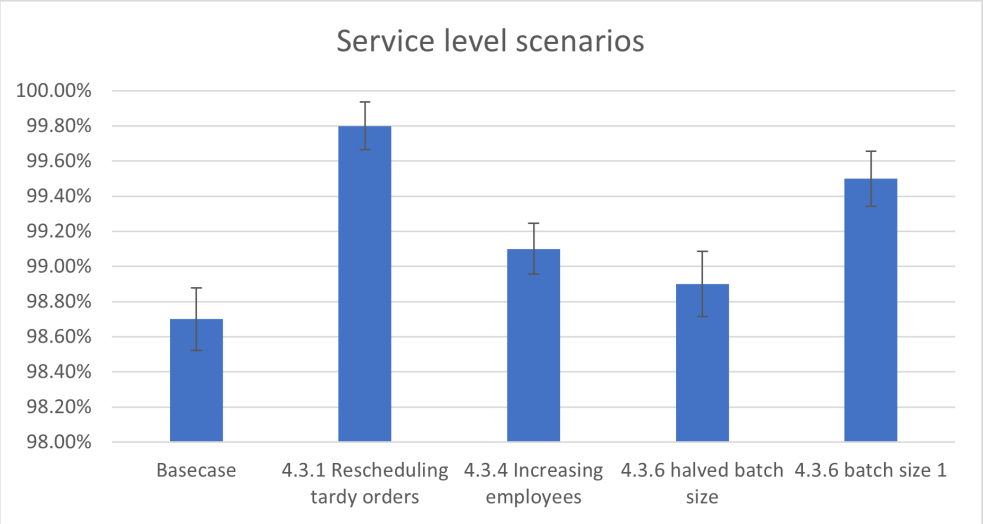


Figure 13: *Improvement of order line service level for the analysed scenarios compared to the basecase.*

5 Conclusion

In this section, the conclusion, scientific contribution and suggestions for further research will be provided.

5.1 Solution to the problem

The main problem considered in this thesis is the insufficient First-Time-Right KPI performance indicating there are too much customer complaints. About 40% of these complaints were caused by an order not received by the customer on the agreed date. Thus, the service level is insufficient. This had three main underlying causes:

1. The planned orders are not available in the expedition hall at the moment of loading the truck. They either still have to be picked or are in transit to the expedition hall.
2. The planned orders do not fit in the truck's trailer as maximum weight or volume are exceeded.
3. The truck arrives too late at a customer.

With a hybrid model combining an Anylogic DES simulation of the internal logistics with a python CVRP optimization of the outbound logistics, a solution is presented to improve the service level and overcome the underlying causes.

Most orders that are not delivered on the agreed date to the customer are in the routes of the trucks that depart early. In the hybrid simulation, there were 12 tardy orders of which 11 were in the earliest delivery window. These orders were rescheduled to a later truck without the need of deploying an additional truck resulting in only 1 order not delivered to the customer on the agreed date. This improves the service level from 97.7% to 99.8% which is above the desired goal of 99.5% service level.

Other options on improving only the internal logistics were also considered, namely: decreasing batch size, using more employees, earlier order intake and batch picking. However, these options could not reach the desired service level. Only a combination with a batch size of 1 and additional order pickers yielded the desired service level in some cases. However, this requires at least four more order pickers which is a more expensive solution than the rescheduling of early tardy orders. Hence, only adaptions in the internal logistics process are found to be not sufficient to solve the problem.

By considering all customers instead of making a pre selection by the planner in the truck planning process, a reduction in driven distance of 14% could be achieved given all assumptions. Hence, more customers are visited in the assigned time window. Moreover, if administratively the maximum capacity of every truck is reduced by 5%, only 7km of additional travel distance was needed to deliver all orders. Hence, by being more on the save side regarding the capacity of the truck, the chance increases that all planned orders fit in the truck and thus more customers receive their orders on the agreed date.

The advice to the management of Stiho is to implement both strategies in order to deliver more orders to the customer on the agreed date and thus reducing the number of complaints and increasing the First-Time-Right KPI performance. The main hurdle preventing the implementation of the new routing planning method is the correct weight and volume information of all customer orders.

5.2 Scientific contribution

The main scientific contribution of this thesis is the hybrid model combining discrete event simulation with the capacitated vehicle routing problem optimization which was not performed in literature before. It is proven that it is possible to use this combination in the logistics field to improve service level at a low costs. The modeling combination can be expanded to other fields outside the building material handling process to for example factory routing problems. Thus general knowledge is obtained.

5.3 Suggestions for further research

Based on the findings and the limited time of this thesis the following suggestions for further research could be looked into:

- Currently, the CVRP part of the hybrid model is ran twice. At the beginning of the day to set the initial route and at 2pm to reschedule the tardy orders. In further research, the routes could be calculated more often for every time a tardy order occurs to see if it can be delivered by another truck without a great detour. This way, the number of tardy orders could probably be reduced even more.
- In the model, the median weight and volume are taken for the orders where this information was missing. Furthermore, 5% of the capacity was used as gap to the maximum capacity to overcome uncertainty and incorrect weight and/or volume. A better way to estimate weight and volume information could be researched such that more orders can be added to one truck.
- Currently, only one day of simulation was ran. With more time and higher computer power, multiple days/weeks could be simulated to see the impact of decisions on other days.
- The ranking system of the simulation is based on due date and similar product types. The ranking system DSG uses is majorly based on this, but is far more complex and only the basics are known by the employees. A further investigation in how the ranking exactly takes place and how it can be implemented in the simulation is recommended.

References

Agalianos, K., Ponis, S., Aretoulaki, E., Plakas, G., and Efthymiou, O. (2020). Discrete event simulation and digital twins: Review and challenges for logistics. *Procedia Manufacturing*, 51:1636–1641. 30th International Conference on Flexible Automation and Intelligent Manufacturing (FAIM2021).

Alewijnse, B. and Hübl, A. (2021). *Minimising Total Costs of a Two-Echelon Multi-Depot Capacitated Vehicle Routing Problem (2E-MD-CVRP) that Describes the Utilisation of the Amsterdam City Canal Network for Last Mile Parcel Delivery*, pages 603–612.

Boysen, N., de Koster, R., and Füßler, D. (2021). The forgotten sons: Warehousing systems for brick-and-mortar retail chains. *European Journal of Operational Research*, 288(2):361–381.

Brailsford, S. C., Eldabi, T., Kunc, M., Mustafee, N., and Osorio, A. F. (2019). Hybrid simulation modelling in operational research: A state-of-the-art review. *European Journal of Operational Research*, 278(3):721–737.

Burinskiene, A., Lorenc, A., and Lerher, T. (2018). A simulation study for the sustainability and reduction of waste in warehouse logistics. *International Journal of Simulation Modelling*, 17:485–497.

Cokelaer, T. (2022). Cokelaer/fitter: Fit data to many distributions.

Jia, W., Wang, W., and Zhang, Z. (2022). From simple digital twin to complex digital twin part i: A novel modeling method for multi-scale and multi-scenario digital twin. *Advanced Engineering Informatics*, 53:101706.

Marques, A., Soares, R., Santos, M. J., and Amorim, P. (2020). Integrated planning of inbound and outbound logistics with a rich vehicle routing problem with backhauls. *Omega*, 92:102172.

Moeller, K. (2011). Increasing warehouse order picking performance by sequence optimization. *Procedia - Social and Behavioral Sciences*, 20:177–185. The State of the Art in the European Quantitative Oriented Transportation and Logistics Research – 14th Euro Working Group on Transportation 26th Mini Euro Conference 1st European Scientific Conference on Air Transport.

Mohammed, M. A., Ghani, M. K. A., Hamed, R. I., Mostafa, S. A., Ibrahim, D. A., Jameel, H. K., and Alallah, A. H. (2017). Solving vehicle routing problem by using improved k-nearest neighbor algorithm for best solution. *Journal of Computational Science*, 21:232–240.

Rasku, J., Kärkkäinen, T., and Musliu, N. (2019). Meta-survey and implementations of classical capacitated vehicle routing heuristics with reproduced results. *JYU Dissertations 113, University of Jyväskylä*, pages 133–260.

Toth, P. and Vigo, D. (2002). Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, 123(1):487–512.

Vélez-Gallego, M. C., Teran-Somohano, A., and Smith, A. E. (2020). Minimizing late deliveries in a truck loading problem. *European Journal of Operational Research*, 286(3):919–928.

Yang, K. and El-Haik, B. S. (2009). *Design for six sigma: a roadmap for product development*. McGraw-Hill Education.

A Appendix

A.1 Python scripts

Python is used for two purposes, the data to distributions fitting for the anylogic process parameters and the CVRP algorithm.

A.1.1 CVRP algorithm

The CVRP python script is based on Rasku et al. (2019). The initial CVRP solution is calculated with the following algorithm:

```
# -*- coding: utf-8 -*-
"""
Created on Wed Oct 12 11:15:00 2022

@author: rsteggink
"""
import pandas as pd

df = pd.read_excel (r'C:/Users/rsteggink/Documents/VeRyPy-master/test/distanceMatrixExcel.xlsx')
#print (df)
dfnew = df.to_numpy()
#print (dfnew)
CapacityInformation = pd.read_excel (r'C:/Users/rsteggink/Documents/Anylogic/adressen.xlsx')
WeightInformation = CapacityInformation.iloc[:,3]
#WeightInformation = pd.DataFrame(WeightInformation)
VolumeInformation = CapacityInformation.iloc[:,4]
#VolumeInformation = pd.DataFrame(VolumeInformation)
#print (CapacityInformation)

MaxWeight = 18000
MaxVol = 30000
MaxRouteLength = 500*1000 #in km*1000 provides meters

#import cvrp_io
from classic_heuristics.nearest_neighbor import nearest_neighbor_init
from util import sol2routes

#E_n51_k5_path = "E-n51-k5.vrp"

#problem = cvrp_io.read_TSPLIB_CVRP(E_n51_k5_path)

solution = nearest_neighbor_init(
    D=dfnew, #distance matrix
    d=WeightInformation, #customer demands (weight, volume)
    d_vol=VolumeInformation,
    C=MaxWeight,
    C_vol=MaxVol,
    L=MaxRouteLength) # capacity constraint

#solution = parallel_savings_init(
#    D=problem.distance_matrix,
#    d=problem.customer_demands,
#    C=problem.capacity_constraint)

for route_idx, route in enumerate(sol2routes(solution)):
    print("Route #%d : %s"%(route_idx+1, route))

Routes = pd.DataFrame(sol2routes(solution))
#Routes.to_excel(r'C:/Users/rsteggink/Documents/VeRyPy-master/test/Routes.xlsx')
dfsol = pd.DataFrame(solution)

TotalDistance = 0
node1 = 0
node2 = 0

for i in range(0, (dfsol.shape[0]-1)): #voeg in lengte dfsol
    node1 = int(dfsol.iloc[i])
    node2 = int(dfsol.iloc[(i+1)])
    TotalDistance = (TotalDistance + (df.iloc[node1,node2]))

print("Total Distance", TotalDistance/1000, "km");
```

The script above calls upon the nearest_neighbor.py heuristic depicted below to solve the problem:

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
###############################################################################
""" This file is a part of the VeRyPy classical vehicle routing problem
heuristic library and provides an implementation of the basic nearest neighbour
heuristic. Both parallel and sequential algorithms are provided. A description
of the algorithm can be found e.g. in van Breedam (1994), but it seems that the
 idea of adapting the approach for CVRP originates from Tyagi (1967).
```

```
 9
10  The script is callable and can be used as a standalone solver for TSPLIB
11  formatted CVRPs. It has really minimal dependencies: only numpy and scipy
12  are needed for reading and preparing the problem instance.
13  """
14  ################################################################################
15
16  # Written in Python 2.7, but try to maintain Python 3+ compatibility
17  from __future__ import print_function
18  from __future__ import division
19
20  from logging import log, DEBUG
21  from operator import itemgetter
22  from collections import deque
23  from util import objf, without_empty_routes, is_better_sol, routes2sol
24
25  from config import CAPACITY_EPSILON as C_EPS
26  from config import COST_EPSILON as S_EPS
27
28  __author__ = "Jussi Rasku"
29  __copyright__ = "Copyright 2018, Jussi Rasku"
30  __credits__ = ["Jussi Rasku"]
31  __license__ = "MIT"
32  __maintainer__ = "Jussi Rasku"
33  __email__ = "jussi.rasku@jyu.fi"
34  __status__ = "Development"
35
36
37  class _PeekQueue:
38      """ This is a helper data strucure, that allows peeking into a list. It is
39      used to calculate the capacity total with ne next nearest neighbor
40      without consuming it from the iterator.
41
42      """
43
44      def __init__(self, l):
45          self.posleft = -1
46          self.posright = 0
47          self.l = list(l)
48
49      def __len__(self):
50          return len(self.l)-(self.posleft+1)+(self.posright)
51
52      def __getitem__(self, idx):
53          if idx>=len(self):
54              raise IndexError
55          return self.l[self.posleft+1+idx]
56
57      def peekleft(self):
58          if len(self)==0:
59              raise IndexError
60          return self.l[self.posleft+1]
61
62      def popleft(self):
63          if len(self)==0:
64              raise IndexError
65          self.posleft+=1
66          return self.l[self.posleft]
67
68      def peekright(self):
69          if len(self)==0:
70              raise IndexError
71          return self.l[self.posright-1]
72
73      def popright(self):
74          if len(self)==0:
75              raise IndexError
76          self.posright-=1
77          return self.l[self.posright]
78
79  def get_seed_node(seed_mode, D, node_nearest_neighbors, served):
80      N = len(node_nearest_neighbors)
81      seed_node = None
82      while True:
83          if seed_mode=='farthest':
84              seed_node  = node_nearest_neighbors[0].popleft()[0]
85          elif seed_mode=='closest':
86              seed_node  = node_nearest_neighbors[0].popright()[0]
87          elif seed_mode=='nearest':
88              nearest_distance = float('inf')
89              for i in range(1,N):
90                  if  served[i]:
91                      continue
92                  elif seed_node==None:
93                      seed_node = i # in case there is no pair
94                  while not served[i] and len(node_nearest_neighbors[i])>0:
95                      nearest = node_nearest_neighbors[i].peekleft()[0]
96                      if served[nearest]:
97                          #print("pop", nearest)
98                          node_nearest_neighbors[i].popleft()
99                      else:
100                         break
101                 if D[i,nearest] < nearest_distance:
102                     seed_node  = i
103                     nearest_distance = D[i,nearest]
104         else:
```

```
105             raise ValueError("Only 'farthest', 'closest', and "+
106                              "'nearest' route initialization "+
107                              "methods are supported")
108         if not served[seed_node]:
109             break
110     return seed_node
111
112 def nearest_neighbor_init(D, d, d_vol, C, C_vol, L=None,
113                           emerging_route_count=1,
114                           initialize_routes_with="nearest",
115                           add_only_to_end=False,
116                           forbidden_nodes=None,
117                           route_improvement_callback=None):
118     """ A greedy nearest neighbor algorithm for solving symmetric CVRPs.
119     The general idea of the heuristic has been proposed e.g. in (Tyagi 1967).
120     The nearest unrouted node is added to a route until a constraint (C or L)
121     is violated and then a new route (taking the nearest unrouted node from
122     the depot) is initialized. This is repeated until all nodes are routed.
123
124     * D is a numpy ndarray (or equvalent) of the full 2D distance matrix.
125     * d is a list of demands. d[0] should be 0.0 as it is the depot.
126     * C is the capacity constraint limit for the identical vehicles.
127     * L is the optional constraint for the maximum route length/duration/cost.
128
129     * emerging_route_count sets how many routes are constructed in parallel
130     * initialize_routes_with can be "farthest" (default) or "closest", which
131        selects if the unrouted node farthest from the depot or closest to the
132        depot is used to initialize the emerging route. Can also be "nearest",
133        and then the route is initialized with the two unrouted nearest
134        neighbors.
135     * add_only_to_end is same to van Breedam (1994, 2002) "places to add stops"
136        parameter. It sets if the customers are added to the end or at either
137        end of the emering route (default).
138
139     The forbidden_nodes and route_improvement_callback can be used to extend
140     nearest_neighbor_init. forbidden_nodes is a list of nodes that are not
141     considered for insertion and route_improvement_callback is an route
142     improvement heuristic with the signature:
143     route_improvement_callback(route_data, D, d, C, L,
144                                served, node_nearest_neighbors), where
145     Route data is a 4-tuple (or similar), with
146       (route, route_demand, route_cost, None)
147
148
149     Tyagi, M. (1968). A practical method for the truck dispatching problem.
150     Journal of the Operations Research Society of Japan, 10:76-92
151     """
152
153     # build the nearest neighbor lists
154     N = len(D)
155     node_nearest_neighbors = [None]*N
156     for i in range(N):
157         node_nearest_neighbors[i] = _PeekQueue( sorted(enumerate(D[i][:]),
158                                                        key=itemgetter(1)) )
159         node_nearest_neighbors[i].popleft() # pop reference to self
160
161     # bookkeeping on the nodes that we have already served
162     served = [False]*N
163     served[0]=True
164
165     if forbidden_nodes:
166         for fn in forbidden_nodes:
167             served[fn] = True
168
169     sol = [0]
170     route_nodes = [None]*emerging_route_count
171     prev_added = [None]*emerging_route_count
172     route_demands = [0.0]*emerging_route_count
173     route_demands_vol = [0.0]*emerging_route_count
174     route_costs = [0]*emerging_route_count
175
176     try:
177         k = 0
178         while True:
179
180             if not route_nodes[k]:
181                 if sol.count(1) < 10:
182                     served[1] = False
183                 if sol.count(2) < 10:
184                     served[2] = False
185                 # Needs to initialize a new emerging route
186                 seed_node =  get_seed_node(initialize_routes_with, D,
187                                            node_nearest_neighbors, served)
188
189                 # build a route around the seed node
190                 route_nodes[k] = deque()
191                 route_nodes[k].append(seed_node)
192                 prev_added[k] = seed_node
193
194
195                 if C: route_demands[k] = d[seed_node]
196                 if C_vol: route_demands_vol[k] = d_vol[seed_node]
197                 if L: route_costs[k] = 0.0
198                 served[seed_node]=True
199
200
```

```
201
202                 if __debug__:
203                     if prev_added[k] == 1:
204                         route = [0]+[1]+list(route_nodes[k])+[1]+[0]
205                     elif prev_added[k] == 2:
206                         route = [0]+[2]+list(route_nodes[k])+[2]+[0]
207                     else:
208                         route = [0]+list(route_nodes[k])+[0]
209                     log(DEBUG-1, "Initialize route #%d with n%d, resulting to %s (%.2f)"%
210                         (k,seed_node, str(route), objf(route, D)))
211
212             else:
213                 # add as a new nonserved node after the previous one
214                 first_node = route_nodes[k][0]
215                 last_node = route_nodes[k][-1]
216                 prev_node = prev_added[k]
217
218                 nn_node = None
219
220
221                 while True:
222                     nn_node = node_nearest_neighbors[prev_node].peekleft()[0]
223                     if served[nn_node]:
224                         #print("pop", nn_node)
225                         node_nearest_neighbors[prev_node].popleft()
226                     else:
227                         break
228
229                 if __debug__:
230                     log(DEBUG-2, "Consider to extend route #%d with n%d"%(k,nn_node))
231
232                 Δ = D[0,first_node]+D[last_node,nn_node]+D[nn_node,0]
233                 add_to_end = True
234                 if not add_only_to_end:
235                     Δ_start =  D[0,nn_node]+D[nn_node,first_node]+D[last_node,0]
236                     if Δ_start<Δ:
237                         Δ = Δ_start
238                         add_to_end = False
239
240
241                 # check constraints
242
243                 constraints_violated = \
244                   (C and route_demands[k]+d[nn_node]-C_EPS>C) or \
245                   (C_vol and route_demands_vol[k]+d_vol[nn_node]-C_EPS>C_vol) or \
246                   (L and route_costs[k]+Δ-S_EPS>L) or \
247                   (len(route) > 16)
248
249
250                 if constraints_violated:
251                     current_route = route
252
253
254                     if __debug__:
255                         log(DEBUG-1, "Constraint violated, storing route #%d as %s (%.2f)"%
256                             (k,str(current_route), objf(current_route,D)))
257
258                     sol.extend( current_route[1:] )
259                     route_nodes[k] = None
260                     # Did not add a node (as there was no capacity left)
261                     #  do not increment k, as it is still the turn of this
262                     #  route to get a new node.
263                 else:
264                     if C:
265                         route_demands[k] += d[nn_node]
266                     if C_vol:
267                         route_demands_vol[k] += d_vol[nn_node]
268                     if add_to_end:
269                         route_nodes[k].append(nn_node)
270                         if L:
271                             route_costs[k] += D[last_node,nn_node]
272                     else: # add_to_front
273                         route_nodes[k].appendleft(nn_node)
274                         if L:
275                             route_costs[k] += D[nn_node,first_node]
276                     prev_added[k] = nn_node
277                     served[nn_node]=True
278
279                     if __debug__:
280                         if route.count(1) ≥ 1:
281                             route = [0]+[1]+list(route_nodes[k])+[1]+[0]
282                         elif route.count(2) ≥ 1:
283                             route = [0]+[2]+list(route_nodes[k])+[2]+[0]
284                         else:
285                             route = [0]+list(route_nodes[k])+[0]
286                         log(DEBUG-1, "Extending route #%d with n%d, resulting to %s (%.2f)"%
287                             (k, nn_node, str(route), objf(route,D)))
288
289                     # a node was added to the route. "pop" the peeked value
290                     # off the iterator and continue to the next route.
291                     node_nearest_neighbors[prev_node].popleft()
292                     k+=1
293
294             # Parallel version builts K routes in prarallel
295             if k==emerging_route_count:
296                 k = 0
```

```
297
298         except (IndexError, KeyboardInterrupt) as e:
299             interrupted = type(e) is KeyboardInterrupt
300
301
302             # The algorithm ran out of nearest neighbors, which means that all
303             #  nodes are served. Close the remaining routes with a return to depot.
304             for rcustomers in route_nodes:
305                 if rcustomers:
306                     sol.extend(rcustomers)
307                     sol.append(0)
308
309                     if __debug__:
310                         if not interrupted:
311                             current_route = [0]+list(rcustomers)+[0]
312                             croute_f = objf(current_route,D)
313                             log(DEBUG-1, "Constraint violated, storing route "
314                                 "#%d as %s (%.2f)"%(k,str(current_route),croute_f))
315
316             if interrupted:
317                 if sol:
318                     unroutedr = [[n] for n in range(N)
319                                  if (n not in sol)]
320                     if sol and len(sol)>1:
321                         interrupted_sol = sol[:-1]+routes2sol( unroutedr )
322                     else:
323                         interrupted_sol = routes2sol( unroutedr )
324                     raise KeyboardInterrupt(interrupted_sol)
325
326             elif len(sol)==1:
327                 sol = None
328
329         return sol
330
331
332     # ----------------------------------------------------------------------
333     # Wrapper for the command line user interface (CLI)
334     def get_snn_algorithm():
335         algo_name = "vB95-SNN"
336         algo_desc = "van Breedam (1994) Sequential Nearest Neighbor construction "+\
337                     "heuristic"
338         def call_init(points, D, d, d_vol, C, C_vol, L, st, wtt, single, minimize_K):
339             if minimize_K:
340                 raise NotImplementedError("Nearest neighbor algorithm does "+
341                                           " not support minimizing the number"+
342                                           " of vehicles")
343             return nearest_neighbor_init(D,d,d_vol,C,C_vol,L, emerging_route_count=1)
344         return (algo_name, algo_desc, call_init)
345
346     def get_pnn_algorithm(emerging_route_count="auto"):
347         algo_name = "vB95-PNN"
348         algo_desc = "Parallel Nearest Neighbor construction heuristic"
349         if emerging_route_count=="auto":
350             def call_init(points, D, d, d_vol, C, C_vol, L, st, wtt, single, minimize_K):
351                 if minimize_K:
352                     raise NotImplementedError("Nearest neighbor algorithm does "+
353                                               " not support minimizing the number"+
354                                               " of vehicles")
355
356                 sol_snn = nearest_neighbor_init(D, d, d_vol, C, C_vol, L, emerging_route_count=1)
357                 if single:
358                     return sol_snn
359
360                 auto_route_count = sol_snn.count(0)-1
361
362                 # NN is so fast we can try with several K and take the best
363                 best_sol = sol_snn
364                 best_f = objf(sol_snn,D)
365                 best_K = auto_route_count
366                 for k in range(2,auto_route_count+1):
367                     sol = nearest_neighbor_init(D, d, d_vol, C, C_vol, L, emerging_route_count=k)
368                     sol = without_empty_routes(sol)
369                     sol_f = objf(sol,D)
370                     sol_K = sol.count(0)-1
371
372                     if is_better_sol(best_f, best_K, sol_f, sol_K, minimize_K):
373                         best_sol = sol
374                         best_f = sol_f
375                         best_K = sol_K
376
377                 return best_sol
378         elif emerging_route_count>1:
379             def call_init(points, D, d, d_vol, C, C_vol, L, st, wtt, single, minimize_K):
380                 if minimize_K:
381                     raise NotImplementedError("Nearest neighbor algorithm does "+
382                                               " not support minimizing the number"+
383                                               " of vehicles")
384                 return nearest_neighbor_init(D, d, d_vol, C, C_vol, L, emerging_route_count)
385         else:
386             raise ValueError("Not a valid emerging_route_count value "+
387                              "(%s) for parallel algorithm"%
388                              str(emerging_route_count))
389         return (algo_name, algo_desc, call_init)
390
391     if __name__=="__main__":
392         from shared_cli import cli
```

```
393     cli(*get_pnn_algorithm())
```

util.py script where the sol2routes algorithm is located:

```
1   # -*- coding: utf-8 -*-
2   ###############################################################################
3   """ This file is a part of the VeRyPy classical vehicle routing problem
4   heuristic library and provides shared utility functions for the algoritm
5   implementations."""
6   ###############################################################################
7
8   # Written in Python 2.7, but try to maintain Python 3+ compatibility
9   from __future__ import print_function
10  from __future__ import division
11  from builtins import range
12
13  from itertools import groupby
14  from operator import itemgetter
15
16  __author__ = "Jussi Rasku"
17  __copyright__ = "Copyright 2018, Jussi Rasku"
18  __credits__ = ["Jussi Rasku"]
19  __license__ = "MIT"
20  __maintainer__ = "Jussi Rasku"
21  __email__ = "jussi.rasku@jyu.fi"
22  __status__ = "Development"
23
24
25  def is_sorted(l):
26      """Checks if the list is sorted """
27      return all(l[i] <= l[i+1] for i in range(len(l)-1))
28
29  def produce_nn_list(D):
30      """Produces a list of lists, each list has 2-tupes of node indices and
31      distances and from a node to all other nodes sorted by that distance. """
32
33      n = len(D)
34      # preprocess D to sorted_per_line_D
35      NN_D = [None]*n
36      for i in range(n):
37          # sort each row
38          NN_D[i] = sorted(enumerate(D[i,:]), key=itemgetter(1))
39      return NN_D
40
41  def objf(sol, D):
42      """A quick procedure for calclulating the quality of an solution (or a
43      route). Assumes that the solution (or the route) contains all visits (incl.
44      the first and the last) to the depot."""
45      return sum(( D[sol[i-1],sol[i]] for i in range(1,len(sol))))
46
47  def totald(sol, d):
48      """A quick procedure for calclulating the total demand of a solution
49      (or a route)."""
50      if not d: return 0
51      return sum( d[n] for n in sol )
52
53  def first_valid(l):
54      """Returns the first non-None or otherwise valid (evals to true) item from
55      the list, or None if no such item exists."""
56      return next((item for item in l if item), default=None)
57
58  def is_better_sol(best_f, best_K, sol_f, sol_K, minimize_K):
59      """Compares a solution against the current best and returns True if the
60      solution is actually better accordint to minimize_K, which sets the primary
61      optimization target (True=number of vehicles, False=total cost)."""
62
63      if sol_f is None or sol_K is None:
64          return False
65      if best_f is None or best_K is None:
66          return True
67      elif minimize_K:
68          return (sol_K<best_K) or (sol_K==best_K and sol_f<best_f)
69      else:
70          return sol_f<best_f
71
72  def without_empty_routes(sol):
73      """Removes empty routes from the solution. WARNING: this also removes
74      other concecutive duplicate nodes, not just 0,0!"""
75      return [n[0] for n in groupby(sol)]
76
77  def sol2routes(sol):
78      """Convert  solution to a list of routes (each a list of customers leaving
79      and returning to a depot (node 0). Removes empty routes. WARNING: this also
80      removes other concecutive duplicate nodes, not just 0,0!"""
81      if not sol or len(sol)<=2: return []
82      return [[0]+list(r)+[0] for x, r in groupby(sol, lambda z: z == 0) if not x]
83
84  def sol2edgeset(sol, symmetric=True):
85      """Converts solution to a set of edges (2-tuples). If the problem is
86      symmetric the tuples are directed from smaller to larger node value to
87      avoid duplicates."""
88      edges = set()
```

```
    for i in range(0,len(sol)-1):
        j = i+1
        if symmetric or sol[i]<sol[j]:
            edges.add( (sol[i], sol[j]) )
        else:
            edges.add( (sol[j], sol[i]) )
    return edges

def routes2sol(routes):
    """Concatenates a list of routes to a solution. Routes may or may not have
    visits to the depot (node 0), but the procedure will make sure that
    the solution leaves from the depot, returns to the depot, and that the
    routes are separated by a visit to the depot."""
    if not routes:
        return None

    sol = [0]
    for r in routes:
        if r:
            if r[0]==0:
                sol += r[1:]
            else:
                sol += r
            if sol[-1]!=0:
                sol += [0]
    return sol
```

Altered version of the initial CVRP solution that is called upon by Anylogic at the 2pm recalculation of the routes. It subtracts the customers that are already served from the initial customer base and recalculates the routing schedule for the remaining customers.

```
# -*- coding: utf-8 -*-
"""
Created on Mon Dec 19 09:59:49 2022

@author: rsteggink
"""

def execute():
    import pandas as pd

    df = pd.read_excel (r'C:/Users/rsteggink/Documents/VeRyPy-master/test/distanceMatrixExcel.xlsx')
    CapacityInformation = pd.read_excel (r'C:/Users/rsteggink/Documents/Anylogic/adressen.xlsx')
    Due_dates = pd.read_excel(r'C:/Users/rsteggink/Documents/KlantgegevensAnylogic.xlsx', sheet_name='Anylogic')
    TardyOrder = pd.read_excel (r'C:/Users/rsteggink/Documents/Anylogic/CustomersDelivered.xlsx')
    TardyOrder = TardyOrder.drop_duplicates(keep = 'first');

    DueDate14 = Due_dates[['Klantnr', 'DueDate']];
    DueDate14 = DueDate14[DueDate14['DueDate'] == "2022-10-07 14:00:00"];
    DueDate14 = DueDate14[['Klantnr']];
    DueDate14 = DueDate14.drop_duplicates(keep = 'first');
    CustomersDelivered = DueDate14.append(TardyOrder);
    CustomersDelivered = CustomersDelivered.drop_duplicates(keep = False);
    CustomersDelivered = CustomersDelivered['Klantnr'].tolist();

    #print (df)
    allklantnr = CapacityInformation.iloc[:,6];
    allklantnr = allklantnr.tolist()
    klantnrfilter = [item for item in allklantnr if item not in CustomersDelivered]
    #testklantnr = ["DC Utrecht"];
    #actualcustomers = df[df.columns.intersection(testklantnr)]
    actualcustomers = df.filter(klantnrfilter, axis=0)
    #actualcustomers = actualcustomers.filter(testklantnr, axis=1)
    actualcustomers = actualcustomers.iloc[: , klantnrfilter];
    actualcustomers = actualcustomers.reset_index(drop=True);
    NewCapacityInformation = CapacityInformation.filter(klantnrfilter,axis=0);
    NewCapacityInformation = NewCapacityInformation.reset_index(drop=True); #niet hier


    dfnew = actualcustomers.to_numpy()
    #print (dfnew)
    WeightInformation = NewCapacityInformation.iloc[:,3]
    #WeightInformation = pd.DataFrame(WeightInformation)
    VolumeInformation = NewCapacityInformation.iloc[:,4]
    #VolumeInformation = pd.DataFrame(VolumeInformation)
    #print (CapacityInformation)

    MaxWeight = 18000
    MaxVol = 30000
    MaxRouteLength = 500*1000 #in km*1000 provides meters

    #import cvrp_io
    from nearest_neighbor import nearest_neighbor_init
    from util import sol2routes

    #E_n51_k5_path = "E-n51-k5.vrp"

    #problem = cvrp_io.read_TSPLIB_CVRP(E_n51_k5_path)

    solution = nearest_neighbor_init(
```

```
        D=dfnew, #distance matrix
        d=WeightInformation, #customer demands (weight, volume)
        d_vol=VolumeInformation,
        C=MaxWeight,
        C_vol=MaxVol,
        L=MaxRouteLength) # capacity constraint

    #solution = parallel_savings_init(
    #    D=problem.distance_matrix,
    #    d=problem.customer_demands,
    #    C=problem.capacity_constraint)

    for route_idx, route in enumerate(sol2routes(solution)):
        print("Route #%d : %s"%(route_idx+1, route))

    Routes = pd.DataFrame(sol2routes(solution))
    #Routes.to_excel(r'C:/Users/rsteggink/Documents/VeRyPy-master/test/Routes.xlsx')
    dfsol = pd.DataFrame(solution)

    TotalDistance = 0
    node1 = 0
    node2 = 0

    for i in range(0, (dfsol.shape[0]-1)): #voeg in lengte dfsol
        node1 = int(dfsol.iloc[i])
        node2 = int(dfsol.iloc[(i+1)])
        TotalDistance = (TotalDistance + (df.iloc[node1,node2]))



    Finallist = []
    Routes_list = []
    for i in Routes.keys():
        df = Routes[i].tolist()
        for j in range(0, len(df)):
            Finallist.append([df[j],j])
        #Routes_list.append(df)
    df_neww = pd.DataFrame(columns=['Klant', 'Routes'])


    Due_datesRoutes = pd.read_excel(r'C:/Users/rsteggink/Documents/KlantgegevensAnylogic.xlsx', sheet_name='DueDatesNew',
                                    names=['Routes', 'Due date'])

    for i in range(0, len(Finallist)):
        if Finallist[i][0] >= 0 and Finallist[i][1] >= 0:
            df_ind = str(Finallist[i][0])
            Routes_str = 'Route {}'.format(Finallist[i][1])

            df_neww = df_neww.append({'Klant': Finallist[i][0], 'Routes': Routes_str}, ignore_index=True)
            df_neww = df_neww.sort_values('Klant')
            df_neww = df_neww.reset_index(drop=True)

    df_final = df_neww.merge(Due_datesRoutes, on ='Routes')
    df_final = df_final.sort_values('Klant')
    df_final = df_final[df_final['Klant'] > 2];

    IndexToKlant = pd.DataFrame(klantnrfilter);
    IndexToKlant.columns = ['Klantnr']
    UpdateRouting = pd.merge(df_final,IndexToKlant, left_on='Klant', right_index=True);
    UpdateRouting = UpdateRouting[['Klantnr', 'Routes', 'Due date','Klant']];
    UpdateRouting = UpdateRouting.reset_index(drop=True);
    #UpdateRouting.to_excel(r'C:/Users/rsteggink/Documents/UpdateRouting.xlsx')
    PyKlantnr = list(UpdateRouting['Klantnr']);
    PyRoute = list(UpdateRouting['Routes']);
    PyDueDate = list(UpdateRouting['Due date']);
    PyTruckAmount = len(Routes);
    return [PyKlantnr,PyDueDate,PyTruckAmount];
    print("Total Distance", TotalDistance/1000, "km");
```

A.1.2 Data to distributions fitting algorihm

This algorithm finds the best distribution fitting to a dataset and is used for the Anylogic parameters:

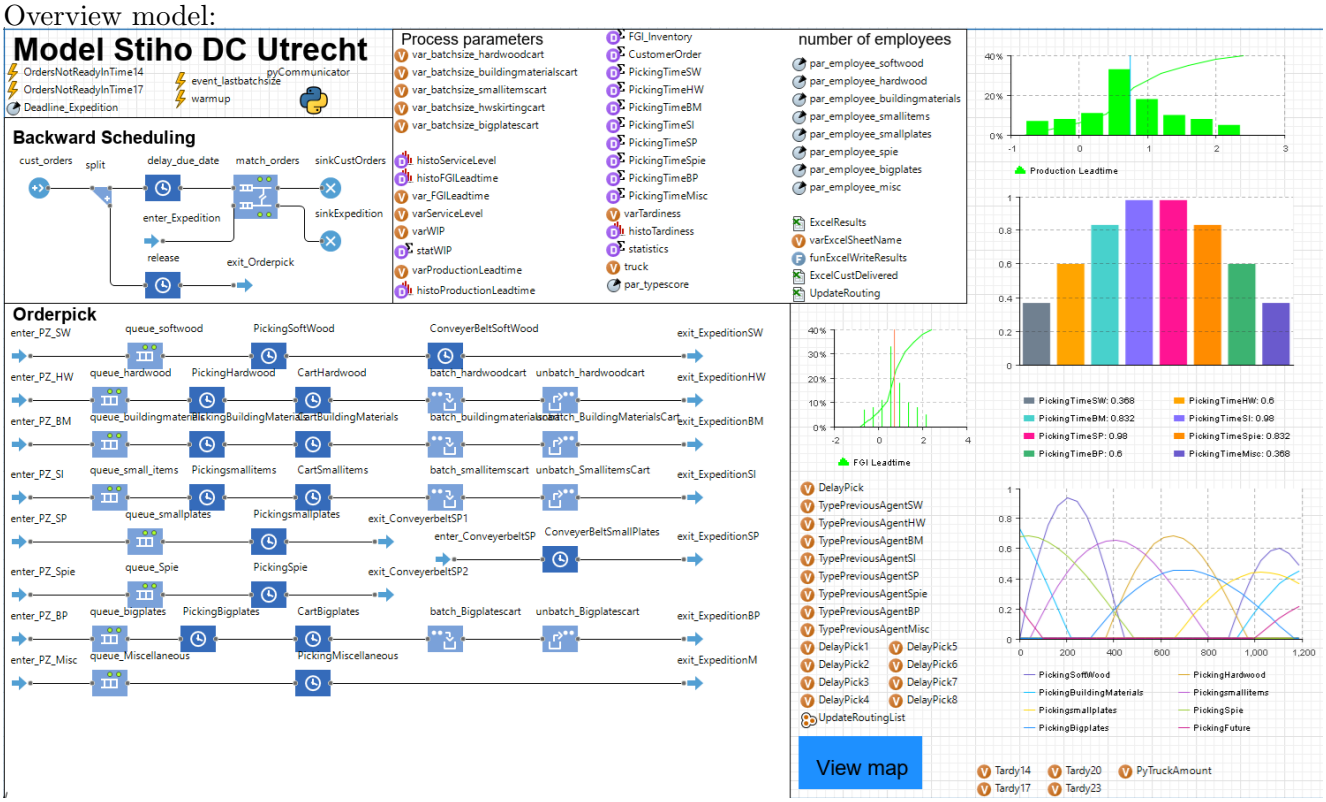
```
# -*- coding: utf-8 -*-
"""
Created on Mon Nov  7 09:38:27 2022

@author: rsteggink
"""

import numpy as np
import pandas as pd
import seaborn as sns
from fitter import Fitter, get_common_distributions, get_distributions
import math

dataset = pd.read_excel(r'C:\Users\rsteggink\Documents\DistributiePZ.xlsx')
dataset.head()
dataset.info()

sns.set_style('white')
```

```
sns.set_context("paper", font_scale = 2)
sns.displot(data=dataset, x="ZACHTHOUT", kind="hist", bins = 100, aspect = 1.5)

picktime = dataset["ZACHTHOUT"].dropna()
f = Fitter(picktime, distributions= get_common_distributions())
f.fit()
f.summary()
f.get_best(method = 'sumsquare_error')
```

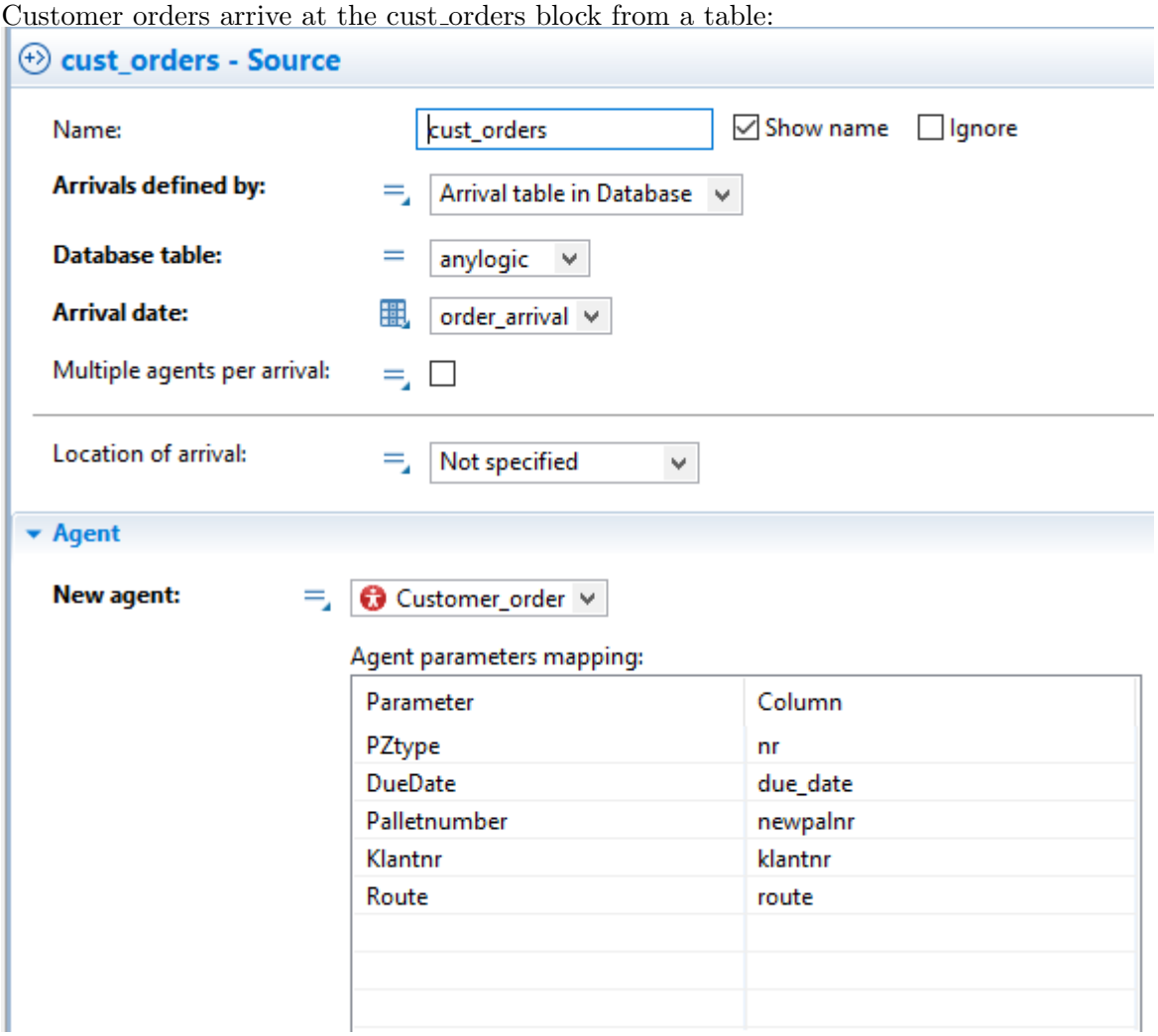
A.2 Anylogic DES model

Anylogic 8.8.0 personal learning edition is used to model the Discrete Event Simulation of the internal logistics. The anylogic file and its code is depicted below:

Overview model:



Customer orders arrive at the cust_orders block from a table:



Code in split block where DC_order is created by duplicating customer_order:

On exit copy:

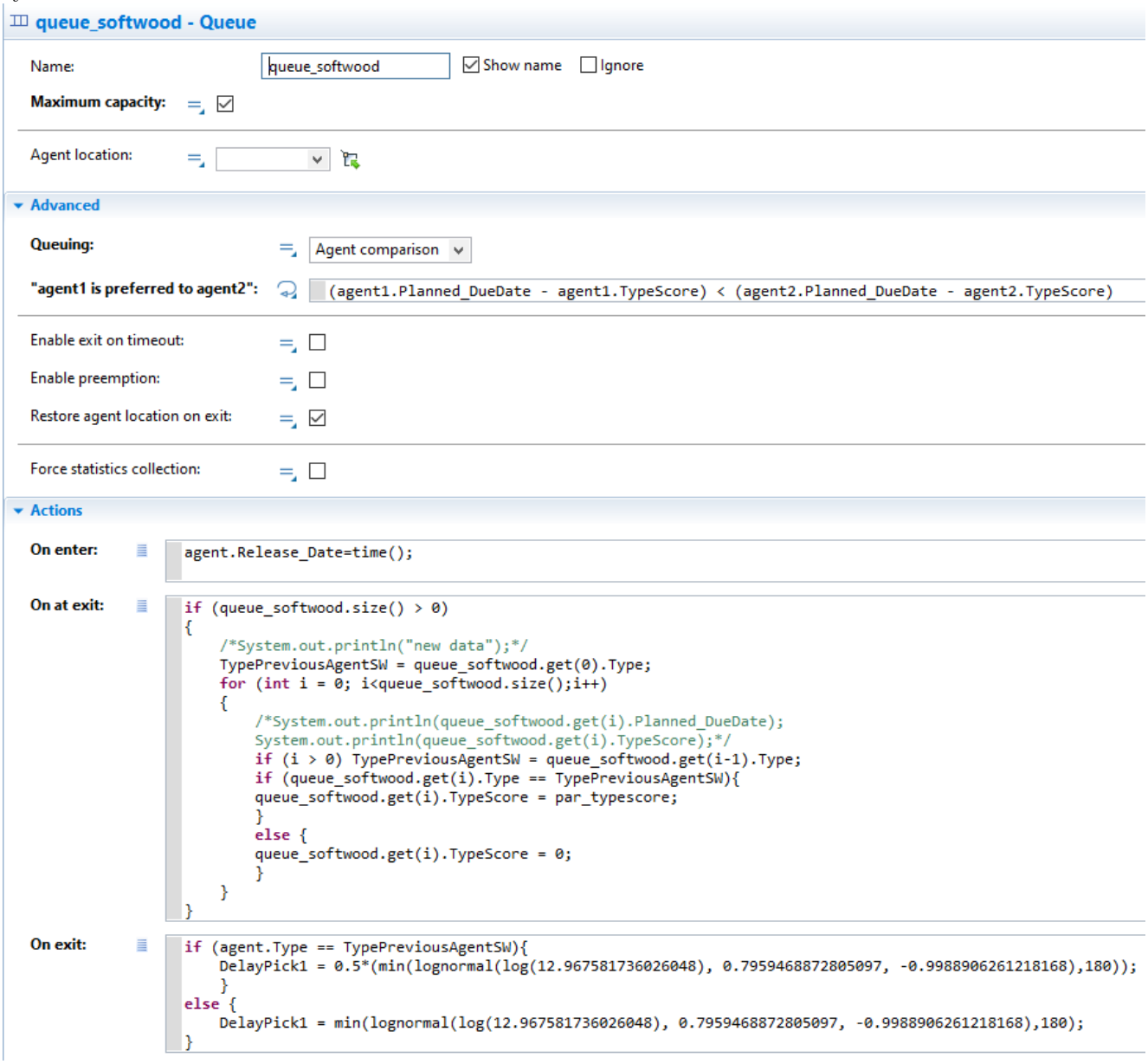
```
agent.Planned_DueDate=dateToTime(original.DueDate)-Deadline_Expedition;
agent.PZtype=original.PZtype;
agent.Palletnumber=original.Palletnumber;
agent.Klantnr = original.Klantnr;
agent.Type = uniform_discr(1,2);
```

Code in exit_Orderpick where the orders are divided over the different pickzones:

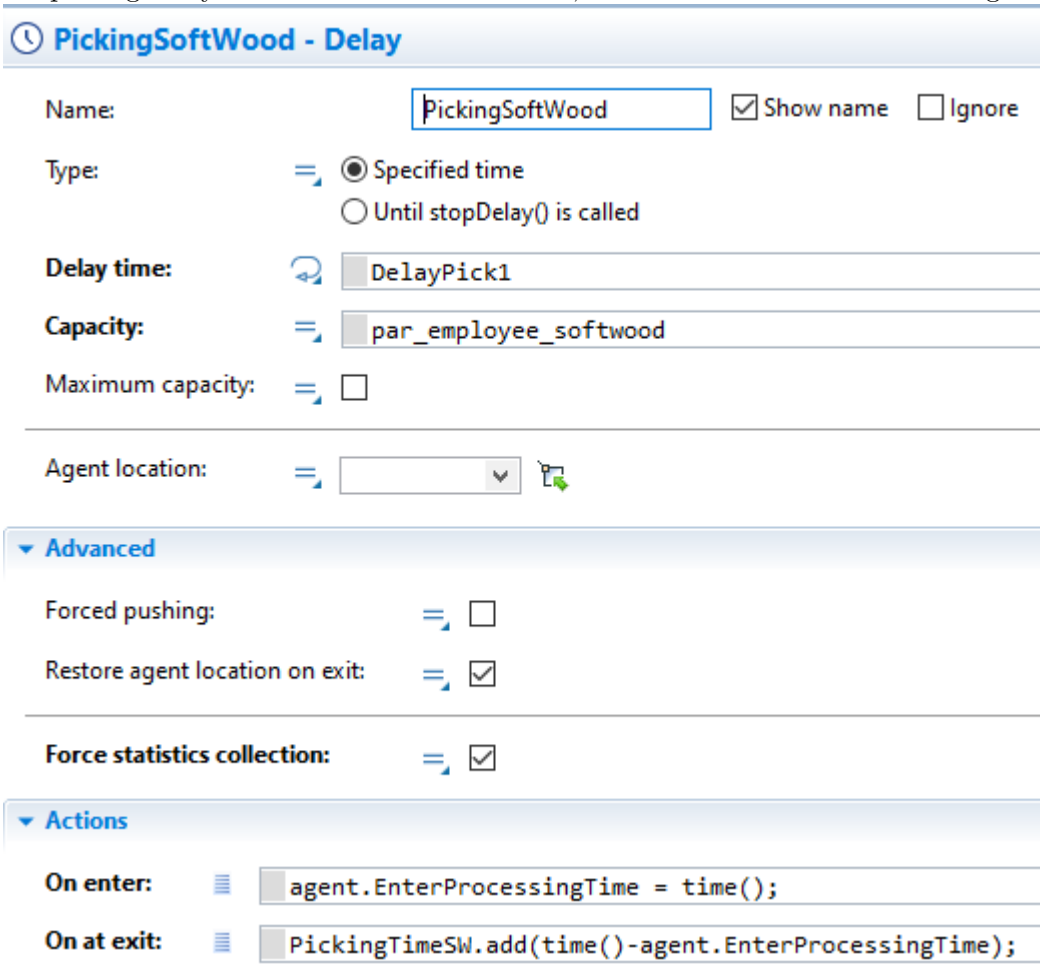
On exit:

```
if( agent.PZtype == 1){
    enter_PZ_SW.take(agent);
    }
if( agent.PZtype == 2){
    enter_PZ_HW.take(agent);
    }
if( agent.PZtype == 3){
    enter_PZ_BM.take(agent);
    }
if( agent.PZtype == 4){
    enter_PZ_SI.take(agent);
    }
if( agent.PZtype == 5){
    enter_PZ_SP.take(agent);
    }
if( agent.PZtype == 6){
    enter_PZ_Spie.take(agent);
    }
if( agent.PZtype == 7){
    enter_PZ_BP.take(agent);
    }
if( agent.PZtype == 8){
    enter_PZ_Misc.take(agent);
    }
```

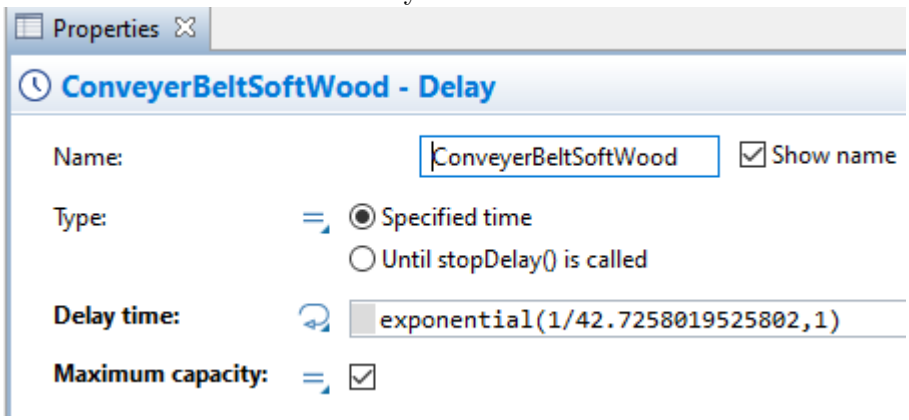
All queues have similar settings, hence only the queue of softwood will be included in this appendix. Here the orders are queued and the ranking is determined in which order the orders will flow through the system:



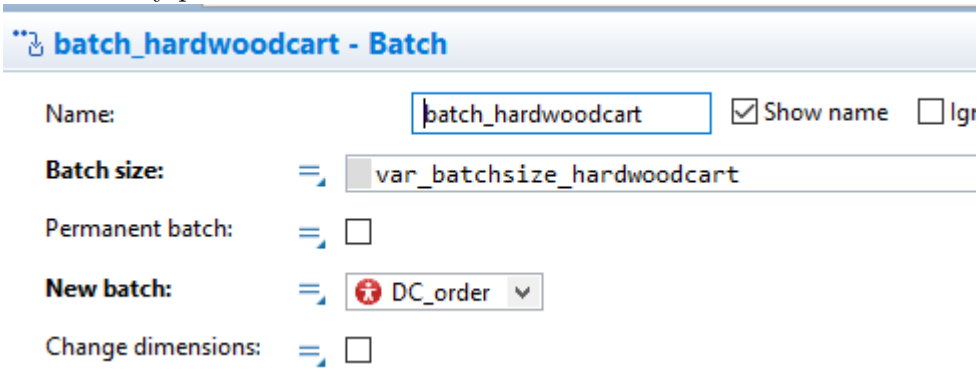
All picking delay blocks are modelled similar, here is the screenshot of PickingSoftWood:



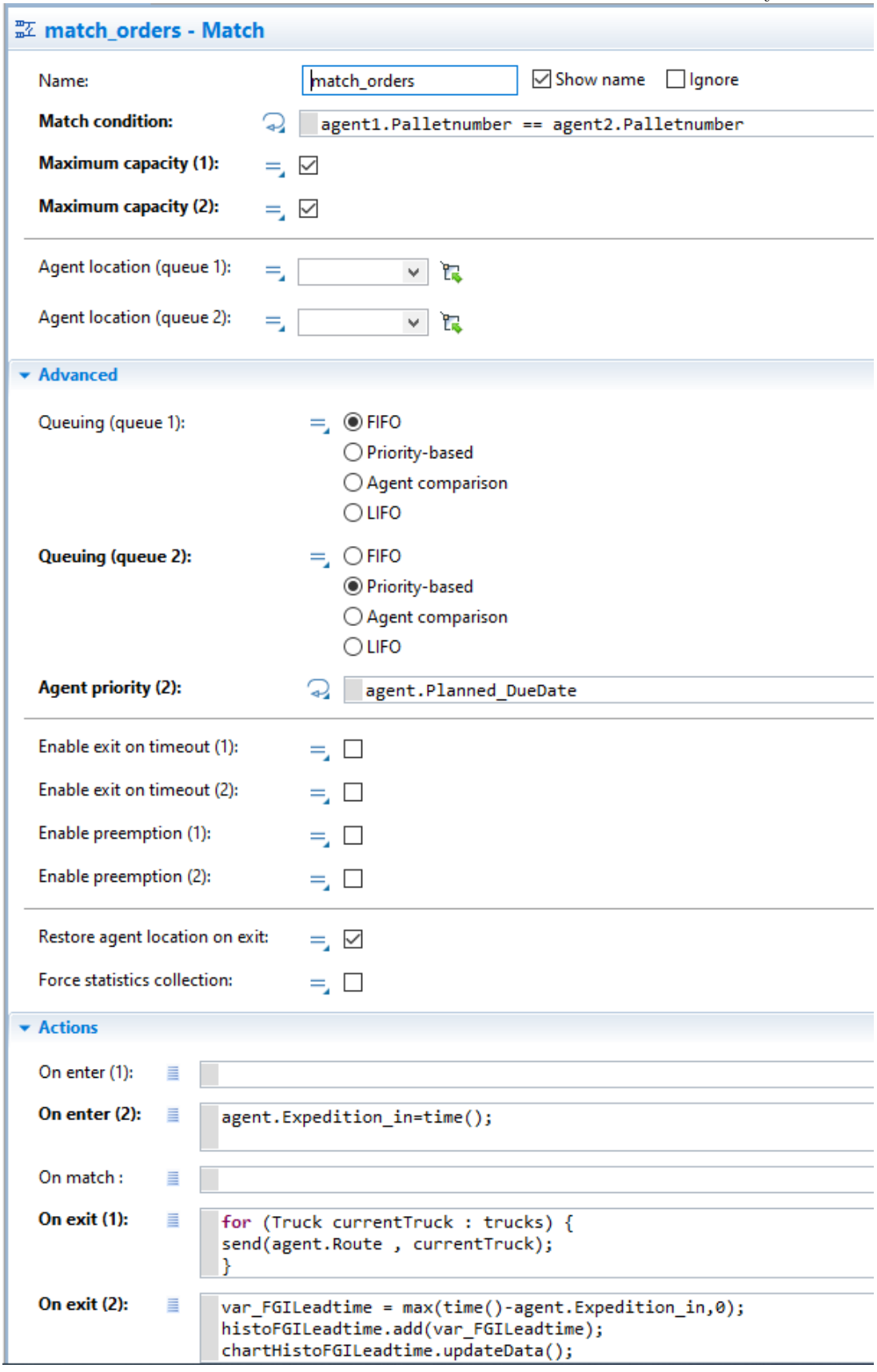
Here is the screenshot of ConveyerBeltSoftWood:



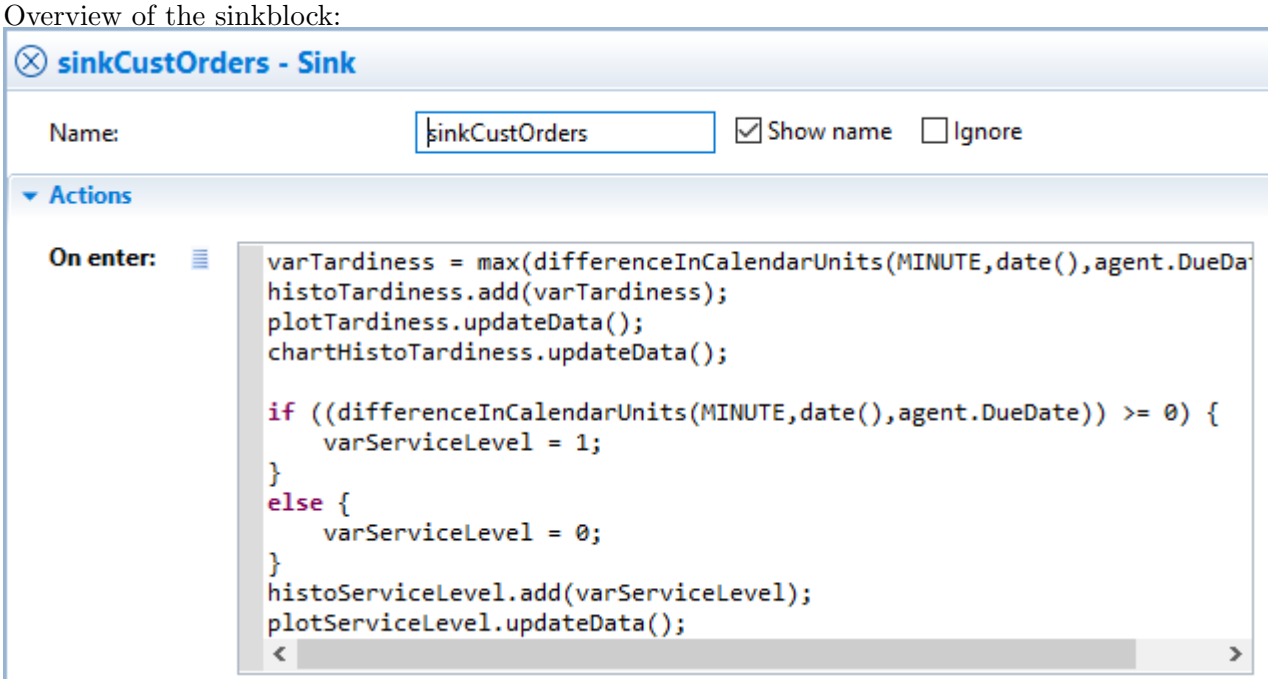
Alternatively product could move in batched carts:



Overview of the match_orders block where is checked if the orders are ready in time:



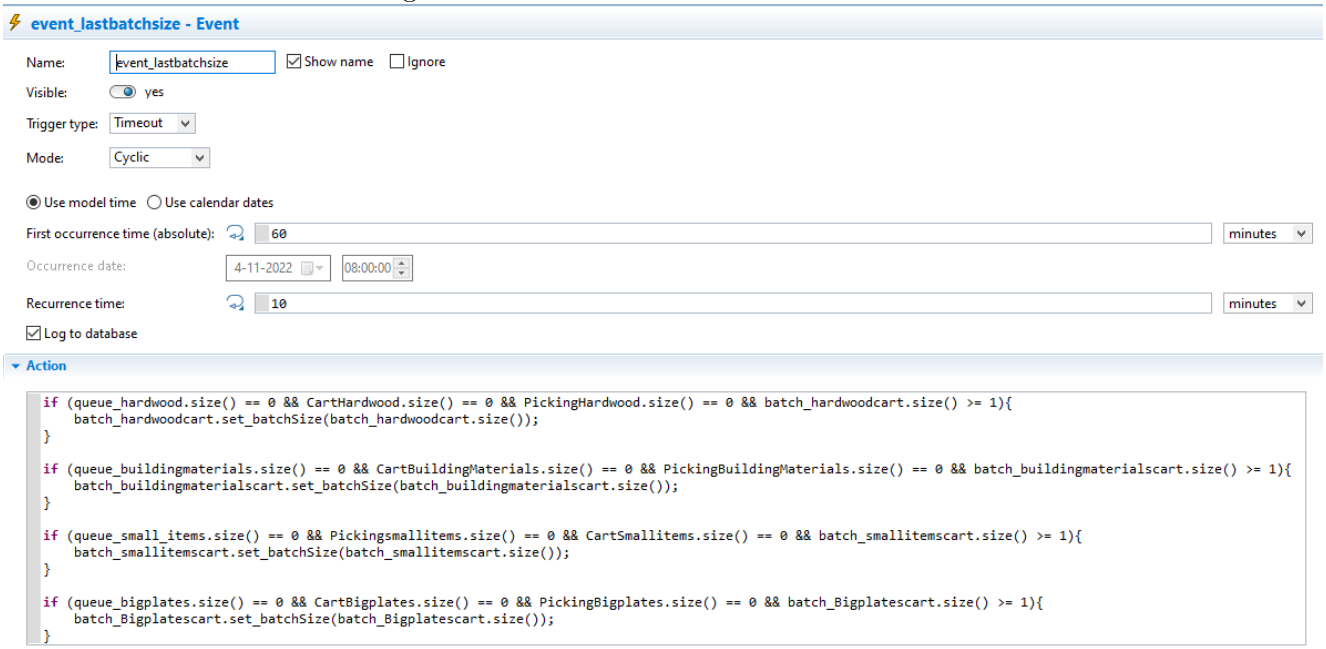
Overview of the sinkblock:



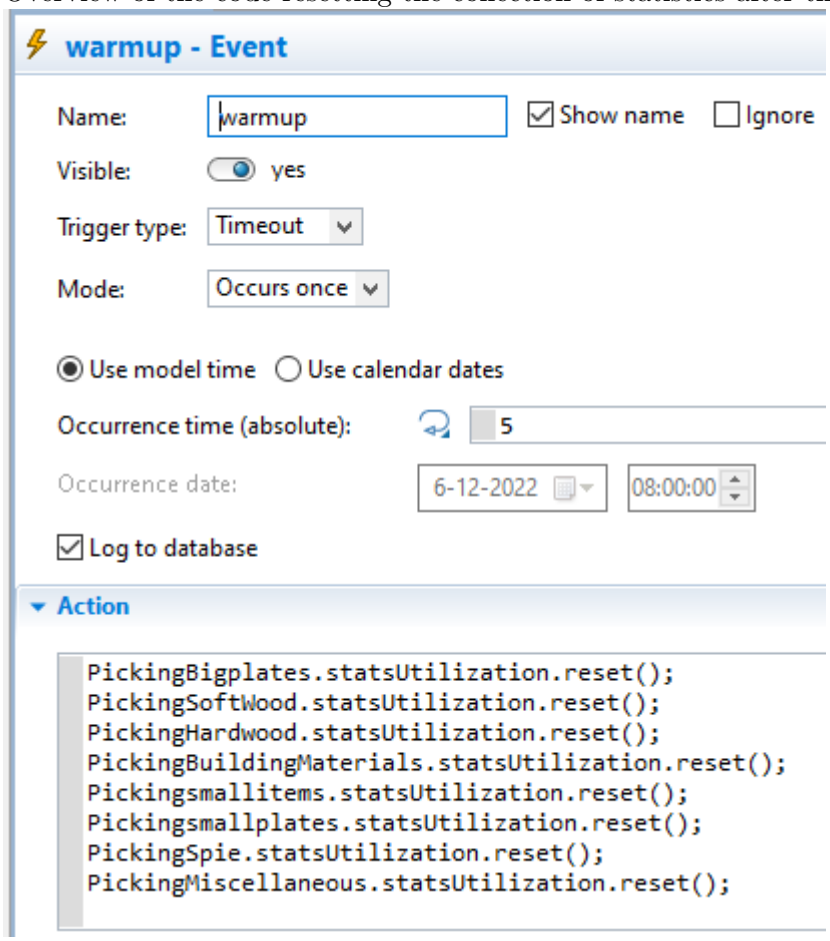
Overview of the code for identifying tardy orders at 2pm and initiating the rescheduling of those orders with the CVRP python algorithm. The tardy orders at 5pm, 8pm and 11pm are identified in similar manner:


```
int ActRow=1;
int ActColumn=1;
/*ActRow=ExcelCustDelivered.getLastRowNum("Blad1");*/
ActRow++;
for (int i = 0; i<match_orders.queue1.size();i++)
    {
        //System.out.println(match_orders.queue1.get(i).Route);
        ExcelCustDelivered.createCell("Blad1", ActRow, 1);
        ExcelCustDelivered.setCellValue(match_orders.queue1.get(i).Klantnr, "Blad1",
        ActRow++;
    }
ExcelCustDelivered.writeFile();
Tardy14 = match_orders.queue1.size();

pyCommunicator.run("import routing");
pyCommunicator.run("import routing2");
pyCommunicator.run("import routing3");
List<ArrayList> results1 = pyCommunicator.runResults(List.class,"routing.execute()")
Pyklantnr = results1.get(0);
PyDate = results1.get(1);
String results3 = pyCommunicator.runResults(String.class,"routing3.execute()");
PyTruckAmount = results3;
String results2 = pyCommunicator.runResults(String.class,"routing2.execute()");
PyRoute = results2;


if (queue_softwood.size()>0){
    for (int i =0; i<queue_softwood.size(); i++){
        DC_order currentorder = queue_softwood.get(i);
        currentorder.Index = Pyklantnr.indexOf(currentorder.Klantnr);
        if (currentorder.Index >= 0){
            Datetranslation = (int) PyDate.get(currentorder.Index);
            if (Datetranslation == 1){
                currentorder.Planned_DueDate = 660;
            }
            if (Datetranslation == 2){
                currentorder.Planned_DueDate = 840;
            }
            if (Datetranslation == 3){
                currentorder.Planned_DueDate = 1020;
            }
            currentorder.markParametersAreSet();
        }
    }
}
```

Overview of the code for dealing with the last batch:



Overview of the code resetting the collection of statistics after the warmup period:



Overview of the code writing the simulation experiment results to Excel:

funExcelWriteResults - Function

Name: funExcelWriteResults ☑ Show name ☐ Ignore

Visible: yes

◉ Just action (returns nothing)

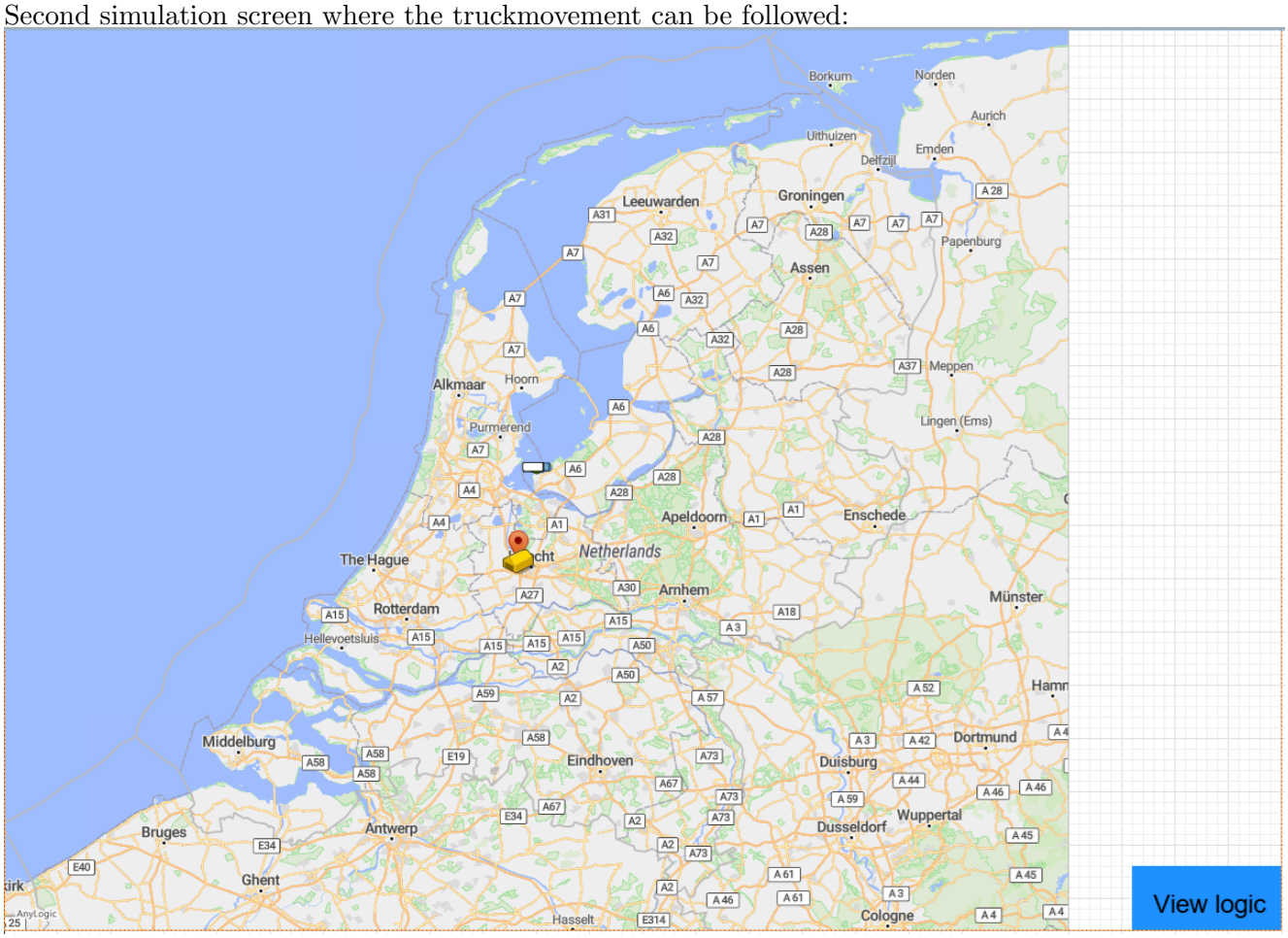
○ Returns value

▸ Arguments

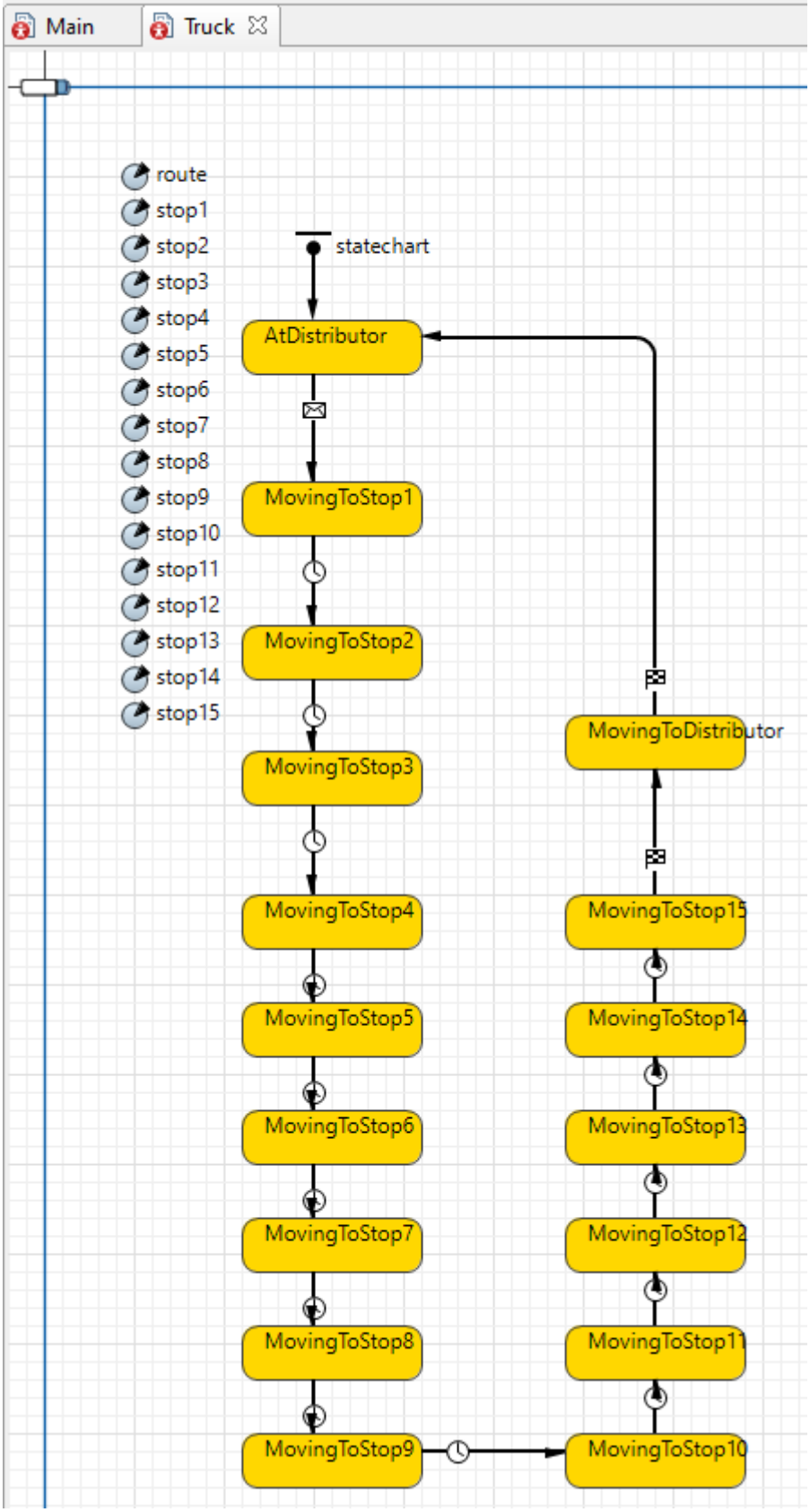
▾ Function body

```
int ActRow=1;
int ActColumn=1;
int MaxColumn=40;
ActRow=ExcelResults.getLastRowNum(varExcelSheetName);
ActRow++;

for (int i = 1; i<=MaxColumn; i++) {
ExcelResults.createCell(varExcelSheetName, ActRow, i);
};

ExcelResults.setCellValue(par_employee_softwood, varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(par_employee_hardwood, varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(par_employee_buildingmaterials, varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(par_employee_smallitems, varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(par_employee_smallplates, varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(par_employee_spie, varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(par_employee_bigplates, varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(par_employee_misc, varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(histoServiceLevel.mean(), varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(PickingTimeSW.mean(), varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(PickingTimeHW.mean(), varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(PickingTimeBM.mean(), varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(PickingTimeSI.mean(), varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(PickingTimeSP.mean(), varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(PickingTimeSpie.mean(), varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(PickingTimeBP.mean(), varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(PickingTimeMisc.mean(), varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(PickingSoftWood.statsUtilization.mean(), varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(PickingHardwood.statsUtilization.mean(), varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(PickingBuildingMaterials.statsUtilization.mean(), varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(Pickingsmallitems.statsUtilization.mean(), varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(Pickingsmallplates.statsUtilization.mean(), varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(PickingSpie.statsUtilization.mean(), varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(PickingBigplates.statsUtilization.mean(), varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(PickingMiscellaneous.statsUtilization.mean(), varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(Deadline_Expedition, varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(Tardy14, varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(Tardy17, varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(Tardy20, varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(Tardy23, varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(PyTruckAmount, varExcelSheetName, ActRow, ActColumn++);
ExcelResults.setCellValue(par_typescore, varExcelSheetName, ActRow, ActColumn++);
```

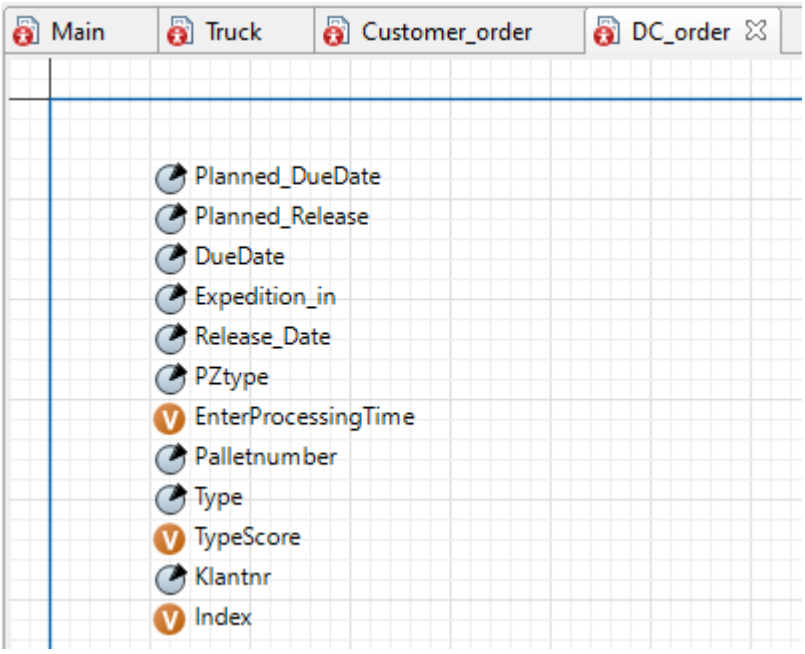
Second simulation screen where the truckmovement can be followed:



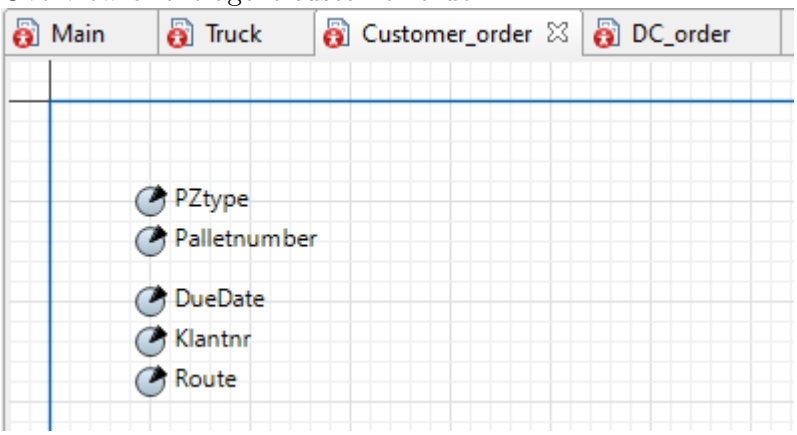
Overview of the logic behind the truckmovement:



Overview of the agent DC_order:



Overview of the agent customer_order:



A.3 Evaluation of working within the company environment

The main difference I experienced between the research environment and the company environment is the money driven approach of the company. Improvements should save money and improve profit margin for it to be implemented. I really enjoyed working within Stiho and talked to many employees to gather information. It is nice to see that the knowledge obtained in the study is also applicable in the real world. As there is in the academic world a more clear true or false, in the company there are multiple perspectives from different employees who have different views on what the problems are and how they should be tackled. It is key to remain objective and do not go with the flow immediately but carefully analyse the situations, problems and perspectives.

A.4 Short analysis of this MDP: 3 pros and 3 cons

Three pros:

- Robust solution that is scalable and can be implemented quite fast without the need for large investments.
- The combination of CVRP and DES as hybrid model is novel. The gained insights can be used by other industries and researchers to build on.
- The questions asked to other employees and discussing the observed problems has influenced other improvement projects within the company outside the scope of this thesis.

Three cons:

- Only one day of simulation was ran whilst a longer time may give a more robust result. The day that was taken was a very average day. Days such as the week of Christmas with only a few order lines and far less trucks might not be suitable for rescheduling orders in a later truck because the detour becomes very large.
- The routing scheduling CVRP algorithm has some assumptions with for example the same type of trucks. This means that the 14% distance reduction will likely not be met in reality.
- The order ranking system could not be recreated on exactly the same grounds as it is too complex and depending on a lot of parameters. Hence, the ranking in the simulation is slightly different than in reality.