APRIL 12, 2019

SCHEDULING ALGORITHM FOR AUTONOMOUS ROBOT CYCLE COUNTING

Author:
Thom Veneman

Supervisors:
prof. dr. Bayu Jayawardhana
prof. dr. Ming Cao

MSc. Industrial Engineering & Management

rijksuniversiteit groningen

Content

Abstract

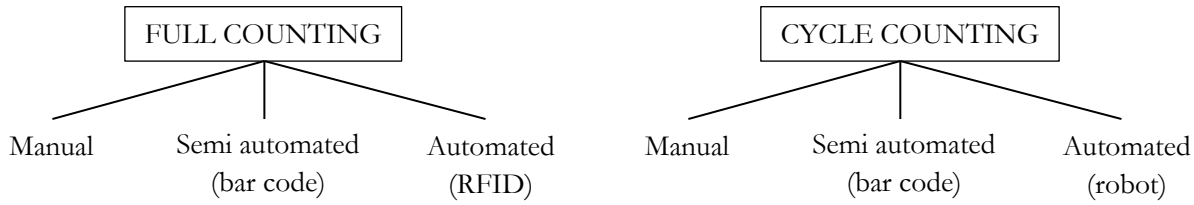
Many companies and universities are developing fully automated cycle counting systems, which make use of autonomous robots, because manual cycle counting is labour intensive. A challenge for achieving fully automated robot cycle counting is the scheduling of the robots to maintain a certain level of inventory accuracy. Therefore, in this thesis a generic scheduling algorithm is developed for planning the routes of the autonomous robots for cycle counting, such that a high level of inventory record accuracy is kept. The algorithm is built by approaching the cycle count scanning problem from a Vehicle Routing Problem perspective. Accounting for the battery depletion, the algorithm schedules multiple trips for the robots each day. This was done with the use of an Adaptive Memory Programming and Tabu Search hybrid metaheuristic algorithm. In addition, the algorithm accounts for recharging times, fixed scanning times per product and a heterogeneous fleet of robots. Furthermore, the algorithm was successfully validated in a case study of a company, that is developing a drone cycle counting system. Lastly, it is shown that four cycle counting methods are compatible with the robot cycle counting system: random sample, ABC, location-based and supervised learning-based cycle counting. The location-based cycle counting method performed best in the case study. However, the supervised learning-based cycle counting method, based on a logistic regression model, would be a better choice in large warehouses with low cycle counting capacity.

Acknowledgements

Firstly, I want to thank my friends and family. During the writing of this thesis, I encountered some though times, which I would not be able to endure without their support, let alone be able to finish this thesis. Furthermore, I want to thank my supervisor prof. dr. Bayu Jayawardhana for the guidance and support during the writing of this thesis.

1 Introduction

To maintaining a high level of inventory accuracy, but also to find and correct conditions of inventory errors, warehouses use inventory counting. In addition, a low inventory record accuracy can cause operational and financial problems for a company. There are two types of inventory counting: full inventory counting, known as wall-to-wall counting, and cycle counting. Full inventory counting requires every product in the warehouse to be counted and recorded [1]. This can be done manually, semi-automated with bar code scanners or automated with Radio Frequency Identification, seen in the scheme below. In full counting with RFID, all products are labelled with RFID tags, which can be read from a distance. On the other hand, cycle counting is counting a sample of the total amount of products more frequently [2].



Although, full counting is more precise than cycle counting, albeit it is also more costly. Manual and semi-automated wall-to-wall counting are very costly, because every product needs to be counted or scanned by an employee and the warehouse needs to be shut down for a certain period of time. Full counting with RFID is also costly because every product needs to have a RFID tag [1]. Furthermore, in practice the RFID tags have proven to be unreliable, resulting in inventory errors [3].

Therefore, cycle counting may pose a better alternative to wall-to-wall counting. Moreover, if cycle counting is done properly it can maintain an inventory record accuracy of 95% [4], [5]. In addition, a study, where 410 manufacturing companies where surveyed, showed that there is a trend towards cycle counting. The results showed that 36% of the companies use only cycle counting, 50% use both cycle counting and periodic full inventory counting and 14% using only periodic full inventory counting. Moreover, of the companies using cycle counting 60% achieved an inventory record accuracy of 90-97% and 20% achieved 98% or higher inventory record accuracy [2]. In this study, the inventory record accuracy was defined as the ratio between the accurate records against the total of records examined.

However, manual and semi-automated cycle counting are still labour intensive. Therefore, many companies and universities are developing fully automated cycle counting systems, which make use of autonomous robots. Most of the companies are pursuing cycle counting with drones [3][6]–[9]. Others, want to use ground robots, for the lower half of the warehouse shelves [10], [11]. These robots can be applied to perform cycle counts outside of working hours. A challenge for achieving fully automated robot cycle counting is the scheduling of the robots to maintain a certain level of inventory accuracy, which is equally as important as the physical and scanning capabilities of the robots for realizing a fully autonomous cycle counting system. Firstly, because most robots have very limited battery capacities. For example, most commercial drones have a maximum of 30 minutes of fly-time and need double as much recharging time. Secondly, due to a high throughput of products in warehouses it is important to scan the right products. Lastly, maximizing the erroneous products found in the warehouse, due to the correct scheduling of the robots, will help improve the business case of the autonomous robot cycle counting system.

Therefore, in this thesis a generic scheduling algorithm is developed that plans the routes of the autonomous robots for cycle counting, such that a high level of inventory record accuracy is kept. The algorithm is built by approaching the cycle count scanning problem from a Vehicle Routing Problem perspective. Accounting for the battery depletion, the algorithm schedules multiple trips for the robots each day. In addition, the

algorithm accounts for recharging times, fixed scanning times per product and a heterogeneous fleet of robots. Furthermore, the algorithm is tested in a case study of the company Arox. Arox is a logistic software developer and is developing an autonomous cycle counting system with drones. In the case study the functioning of the algorithm and the best cycle counting method are tested.

1.1 Higher level context

Warehouses have become an important part of any supply chain [12]. Therefore, inventory record accuracy also has become increasingly important. Inventory record accuracy is the difference between physical inventory held in storage and the record of the inventory stored in an information system of the company [13]. The units of measure are either dollar-based or Stock Keeping Units-based (SKU-based) [14]. There are many financial and operational reasons for maintaining high inventory record accuracy. Financial reasons are [14]:

- Investors want an accurate book value.
- Inventory is one of the primary indicators of financial health and value.
- Conventional lenders need assurance the inventory record is accurate, since this is often collateral.
- Taxation is often based on inventory value. Therefore, underpayment can incur penalties and overpayment of taxes of course reduce profits.

Operational reasons for manufacturing, distributors and retailers are [14]:

- Out of stock products create delivery delays.
- Employees waste time on missing or misplaced products.
- Frequent stock outs result in inventory overcompensation. Therefore, unnecessary inventory is held, which cause unnecessary space to be held and capital to be spend.
- Eliminating physical inventory audits. These types of audits are required when the inventory record accuracy is questionable.

Operational reasons for manufacturing companies specific are [14]:

- Out of stock products interrupt production delays.
- Items that are missing cause delays and idle time, which reduces manufacturing efficiency.
- The turnover of inventory expresses the overall manufacturing efficacy.
- Manufacturing Resource Planning (MRP) and Enterprise Resource Planning (ERP) control systems require inventory record accuracy's of between 95% and 99% to function well.

In addition, high inventory record accuracy is very important in lean environments. Just-in-time, lean and continuous improvement are powerful tools that improve the operations of companies in a comparative environment. These tools drastically lower the inventory that is held by the company, which increases the need for inventory record accuracy (IRA), because this increases the chance of stock-outs [15].

However, most of all inefficient inventory information can lead to incorrect inventory acquisition decisions, which result in poor customer satisfaction and high inventory costs [13]. Inaccurate inventory records can reportedly lead to a loss of profits of around 10%, because of lost sales of products that are out of stock (stock outs) and unnecessary inventory carrying. Furthermore, research has shown that a single company lost a staggering 25% profitability, because of products that were misplaced and reported out of stock [16]. Lastly, it was shown that when the IRA of a manufacturing company was increased from 65% to 95% the company saved approximately 3 million dollars per year. In addition, implementing cycle counting saved 30.000 dollars in counting cost [2].

1.2 Research questions

This research focuses on three main research questions, mentioned below, in the areas of: the cycle counting method, the scheduling algorithm and the performance of the cycle counting system. Each of the research questions contains several sub-questions. The sub-questions are answered throughout the thesis.

- What is the best combination of cycle counting method combined with autonomous robot cycle counting in terms of inventory record accuracy?
    - What is cycle counting?
    - What types of cycle counting methods are there?
    - When can results of a cycle count be inferred for the rest of the population?
    - How much products need to be scanned each day to guarantee a certain level of inventory accuracy?
    - How can the cycle counting methods be implemented in combination with the robot routing algorithm?
    - Which type(s) is/are compatible with the autonomous robot cycle counting system?
- How can an algorithm be constructed that schedules robots for the purpose of cycle counting?
    - What types of Vehicle Routing Problems are there?
    - What types of Vehicle Routing Problems algorithms are there for the autonomous robot cycle counting system?
    - What should the algorithm be able to do?
    - What is the best algorithm for the purpose of cycle counting, in terms of speed, feasible solutions and amount of customers?
    - How can the algorithm be constructed?
- How can the robot routing algorithm be validated and how does it perform in the case study of Arox?
    - How can the case study of Arox be simulated?
    - How should the algorithm be adapted for the case study?
    - How should the Monte Carlo simulations be constructed?
    - How can the Monte Carlo simulation be validated?

1.3 Thesis outline

In Chapter 2, a literature study is performed for the cycle counting method and the vehicle routing algorithm. The scheduling algorithm will be explained in Chapter 3. Furthermore, a case study of the company Arox will be performed and evaluated in Chapter 4 and 5 respectively. Chapter 6 will be a discussion of the results and the future work will be discussed. Lastly, the main research questions will be answered in the conclusion in Chapter 7.

2 Literature review

The literature search contains three main search areas: cycle counting, statistical consideration of cycle counting and the vehicle routing problem. Firstly, it is researched what cycle counting is and what kind of cycle counting methods exist. Secondly, it is discussed when a cycle counted inventory accuracy can be inferred to the rest of the product population and how to determine the cycle counting sample size. Lastly, it is researched what types of vehicle routing problems (VRP's) there are and what methods there are to solve them. Therefore, variations of the following main search words were used: cycle counting, robot cycle counting, cycle counting sample size, inventory record accuracy, vehicle routing problems, heuristic VRP, metaheuristics VRP.

2.1 Cycle counting

As mentioned previously, inventory record accuracy is the difference between physical inventory held in storage and the record of the inventory stored in an information system of the company [13]. Therefore, an inventory record should at least contain: location, stock number, possessed quantity and a condition code. The record can be defined as incorrect if there is a discrepancy with one of the categories mentioned [2]. Any discrepancy measured should be considered seriously, because these can have large consequences, as was explained in the Introduction.

In addition, to maintaining a high level of inventory accuracy, as was mentioned in the Introduction, cycle counting has three other goals: identifying causes of discrepancies, to correcting the conditions causing the discrepancies and providing a correct statement of assets [17]. However, in this research it is assumed that the robots do not physically correct the discrepancies. There are four types of discrepancies, or errors, that can occur: Shrinkage errors, transaction errors, inaccessibility errors and supply errors [18].

Transaction errors include: delivery, shipment, scanning errors and incorrect identification of products. Delivery errors are defined as a mismatch between the required and actually delivered quantity of products by the supplier or delivery of the wrong product or to the wrong directions. When a customer receives the wrong products, a shipping error occurs. These shipping errors can be costly, since the customer can demand a refund or the supplier needs to pay double the transportation costs. Scanning errors most often occur when employees scans one out of two of the products with the same price twice instead of both of them separate. shrinkage errors, also called stock loss, are all errors that cause a loss of products that are ready for sale. Reportedly, shrinkage accounts for 1.69% of retailer sales. These errors include: shoplifting, theft of employees, paperwork and administration errors, and unavailable products for sale and vendor fraud. Products that are unavailable for sale, think of products with damage, out-of-code products or discontinued products. Inaccessible inventory, also called misplaced products, account for the products that are out of place and unavailable for customers. Lastly, supply errors is limitedly researched [18].

For finding and correcting these kinds of discrepancies there are different cycle counting methods. The methods differ in how the sample of products, that are counted, is chosen. The different methods are explained below.

2.1.1 Random sample

As the name suggests, random sample cycle counting is selecting a sample from the population at random. Therefore, every product in the population has an equal chance of being counted. If the sample is sufficiently large and has stability, random sample cycle counting is generally accepted as the best measure for inventory record accuracy (IRA). Performing a cycle count each day ensures a sufficiently large sample. However, when the sample lacks stability, its ability to infer accuracy for the rest of the population diminishes. [15].

In random sample cycle counting there are three techniques to create a sample: constant population, diminished population and diminished population with timing. Constant population sampling returns the sample to the population after counting and diminished population sampling does not. In diminished population, sampling with timing the product is ensured to be counted within the same time interval. This prevents the last product in a diminished population form being the first in the second cycle [15].

2.1.2 ABC cycle counting

In this form of cycle counting the population is classified in three categories according to a Pareto analysis [2]. The Pareto principle states that a minority of the population account for the majority of the result [15]. Therefore, the categories can be determined based on: annual usage dollars, length of lead-time, criticality of equipment usage, frequency of issue or other company relevant criteria. The technique then counts class "A" products more than class "B" products and class "B" products more than class "C" products. Most often, the frequency of counting the products within each group is set, based on subjective organizational priorities. Typically, for class A, this is four times per year, for class B two times per year and class C once each year. Therefore, the number of cycle counts needed per day and the number of counters needed can be calculated given a total number of SKU's. Thereafter, the counts can be scheduled over time. The samples from each group can be chosen using one of the three methods discussed earlier: constant population, diminished population and diminished population with timing, until the number of counts is reached for a year [2].

There are two disadvantages of ABC cycle counting. Firstly, the classification is mostly based on financial measures. However, the price of the product does not determine its importance towards production or shipment. Even the less expensive class C products can cause production delays if not available. Therefore, it can be helpful to consider other categories in classifying the SKU's like: usage, lead-time, criticality or bill of materials level. The products that have other important aspects can then be labelled as class A. However, the classification based on financial measures can be a good starting point. Secondly, the counting workload depends on the number of SKU's in storage [2].

2.1.3 Process control cycle counting

Process control cycle counting [15] entails only counting items that are easy to count. The researchers state that it is "controversial in theory, but effective in practice". This method is based on three criteria: location, ease of counting and obvious errors. The process starts by assigning counters to specific areas in the warehouse. Thereafter, the counters only count the products that are easy to count. This mostly means either products that are low in the rack and easily assessable for the counter or products that are low in quantity. If products are large in quantity there is no physical count made, only compared to the approximation of the counter. Products that are obviously misplaced or misidentified should be added to the sample. Therefore, the counter is free to choose if a product is added to the sample. Then the IRA is measured on the products that were actually counted. Process cycle counting requires inventory records that show the piece count of a product in multiple locations in the warehouse and that the counter has an inventory record listing available of all the products and their quantities and locations.

Process cycle counting is much more cost effective than other forms of cycle counting, because it is faster and requires less counters due to less products that may be counted [2]. In one case this method was a 1000 percent more cost effective than random and ABC cycle counting [15]. However, this method is statistically biased in two ways: the counter decide whether to count a product and the counter does not perform a blind count. In addition, it could be possible that some "hard to count" products are never counted. However, this can be avoided by scheduling a count at the end of the year for items that have not been counted. As was shown in a case study where process cycle counting was used on a population of 30.000

part numbers during 1 year. At the end of the year, only three products were not scanned once in the previous year. In addition, a case study showed that process cycle counting on 30.000 part numbers, of which 300.000 counts were made during the time of observation. Resulted in around 97.2% IRA, while a random sample cycle count yielded a 98.6% IRA [15]. Research has shown that while this method of sampling is biased, it is biased towards products prone to contain discrepancies [2].

2.1.4 Location based

As mentioned previously, location based cycle counting is similar to process control cycle counting. However, it does not differentiate on the ease of counting and the counter does not possess an inventory record list. A specific area in the warehouse is chosen as a sample and every product within that area has to be counted. As well as process cycle counting, a disadvantage is that the samples are not formed using product characteristics. Therefore, location of the products within the warehouse may be irrelevant towards the production or distribution functions [2].

2.1.5 Opportunity based

In this method, counts are performed at specific key events during the process. They can be scheduled for when: a product is stored, a product is reordered, a product is requested or the balance drops below a certain level. Counting can also be scheduled based on the amount of SKU transactions, called transaction based cycle counting. Therefore, counting a product after a certain amount of transactions. Determining the frequency of counts and which products to count are important decision parameters of all the methods mentioned above [2].

2.1.6 Cycle counting based on supervised learning

In addition, to the traditional methods of cycle counting, cycle counting based on supervised learning is a more modern method. In [1] supervised learning is used to determine the products that are most likely to be inaccurate, based on several product characteristics. Supervised learning, in this case supervised classification, is used in "problems with a set of predefined classes and an available dataset with known classes" [1]. Therefore, a classification model assigns a class, in this case accurate of inaccurate, to a data point based on features of the data. These features can be any characteristics of the data, for example: transaction volume, storage height, quantity and product weight. In this research the classification models Logistic Regression and Neural Networks are used, because these models "provide a functional form that predicts the likelihood of an item belonging to a class" [1]. Both models were trained with a random one third of the experimental data set and the rest was used to test the model. Furthermore, the model was trained ten times over the dataset, to increase the validity of the test.

Both the logistic regression model and neural networks model were tested in two company cases against: Transaction-based, Inventory-based, book value based and random sample cycle counting. The results show that both models outperform the random sample cycle counting method in both company cases. Furthermore, in one case they also outperform the book-value based approach. However, they do not outperform the inventory and transaction-based methods. They explain that this could be due to the significant features in the logistic regression model that are also used in the transaction and inventory-based methods. Furthermore, it was found that the logistic regression model is significantly less computational expensive than the neural networks method. Another drawback of their method is that it might not be beneficial to companies that already spend a lot of time cycle counting and therefore count all records frequently, because the method doesn't reduce the amount of records that have to be counted [1]. However, there are also additional benefits for using this approach. Firstly, the logistic regression model assigns weights to the different features of the product that cause inaccuracies. These weights can be leveraged to

do deeper root-cause analyses into inventory record inaccuracies. Secondly, their model does not need any expertise on which features cause inventory inaccuracies, as the traditional methods do, and is applied easily. Lastly, the model is constantly retrained with new counting data. Therefore, the model adapts to ever changing causes of inventory record inaccuracies [1].

2.1.7 Control group cycle counting

Control group cycle counting is counting the same group of products in the same location within a short period of time. Therefore, errors and their causes can be easily found. This type of cycle counting is used to prove that a new counting system works. When the new system fails, control group cycle counting will observe this failure. A reduced interval between counts will reduce the error exposure time and the amount of possible error products, which eases the analyses and correction [15].

2.2 Statistical consideration cycle counting

An important note to make is that the IRA given by any type of cycle counting method can only be inferred to the rest of the population if an unbiased random sample is taken. In general, sampling in a biased manner, like the popular "catch whatever you can in the cheapest way possible " method, can be very dangerous and costly [19]. When the sample is biased, the outcome of the sample analysis is biased too. In the experience of the authors in [19], the biases that these non-probabilistic samples introduce can be up to 1000%. Thus, in the long-term all actions that are based on these biased parameters will always cost money. For example: wrongly estimated market value of products, non-optimal process parameters or highly biased specimens taking from a copper mine. The financial losses incurred, by the use of biased samples, can reach up to millions of dollars per year [19].

This also applies to the inventory record accuracy, if the IRA is underestimated, unnecessary costs can be incurred by adding capacity to the cycle counting system. When the IRA is overestimated, all the reasons for maintaining a high IRA can occur, mentioned in Chapter 1.1, for example: penalties for stating a wrong inventory value for taxation, process delays and delivery costs due to stock-outs, non-functioning control systems (MRP or ERP) etc.

Therefore, an important distinction has to be made in the different goals of cycle counting. The autonomous robot cycle counting system is implemented to find conditions that cause inventory inaccuracy, which helps to maintain a high IRA and to give a correct statement of assets, i.e. a reliable IRA. These two goals are fundamentally different, because for a reliable IRA the samples need to be taken in a random unbiased manner and finding inventory inaccuracies for maintaining a high IRA, as many and most efficiently inaccurate products need to be scanned.

There is a method, described in [5], that is based on the random sample cycle counting method combined with a form of quality control, that can do both. In a static case, the proportion of accurate records X' is equal to

$$\mathrm{X}' = 1 - \frac{\mathrm{Q}}{n}, \tag{1}$$

where Q are the inaccurate products found and then corrected and n is number of records counted in the sample out of the total population N.

The records will be accurate or inaccurate and follow a binomial distribution, with the estimated standard deviation s as

$$s = \left(\frac{X'(1 - X')}{n}\right)^{\frac{1}{2}}. \tag{1}$$

Furthermore, the sample size can be determined to achieve the level of inventory record accuracy R, within a predefined level of error e, with

$$n = \frac{4\,R(1 - R)}{e^2}. \tag{3}$$

When a stratified approach is used, like the ABC cycle counting method based on cost or frequency of errors, the appropriate number of products to count per class n' can be calculated with

$$n' = \frac{n}{\left(1 + \frac{n}{N}\right)}, \tag{4}$$

where N is now the total number of products within the relevant group, i.e. A, B or C. It is then proposed to use a process control chart where the IRA of each sample is mapped. In addition, it can be used to map the IRA per different product groups (ABC) or the inaccurate products per employee performed the transaction. The latter is an interesting approach, which can reveal employees that do not perform well. This can be a great category addition to the cycle counting approach based on supervised learning.

There is also a dynamic approach proposed in [5], which is based on the premise that to maintain a certain level of IRA, as much inaccurate products need to be found and corrected; as are produced by the system. Therefore, this approach can be leveraged to adjust the sample size, i.e. the capacity of the cycle counting system from one period to the next, based on the IRA of the previous period. In this approach, the required improvement rate U' is given by

$$U' = \left(\frac{T'}{Q'}\right)^{\frac{1}{t}} + L' - 1\,, \tag{5}$$

where T' is the target level of accuracy, Q' is the current level of accuracy, L' is the rate at which errors occur and t is the period allowed for the improvement. Furthermore, in this approach the required sample size to achieve this improvement is given by

$$n = \frac{-(U' - L')N}{\left(1 - \frac{(T' + Q')}{2} + U' - L'\right)}. \tag{6}$$

However, this approach is based on estimates for error rates and initial accuracy levels, which may be wrong. Therefore, it is suggested to perform a T-test assessing the current level of IRA. When sample sizes are large or when the error rates are low in comparison to the sample size it is more effective to use a non-random approach [5].

2.2.1 Sample size determination with non-random cycle counting

In addition, [5] gives an approach to determine the appropriate sample size n when non-random cycle counting methods are used. It is based on the assumption, that if N is the total population within a group, then the period t in which all the products are counted is given by

$$t = \frac{N}{n}. \tag{7}$$

If L' is the error rate, then the IRA of a group of products, one period after being counted, would be given by $(1 - L')$. After t periods the expected IRA will be given by $(1 - L')^t$, given the assumption that the errors will accumulate exponentially over time. Here the maximum amount of periods is given by (7). Therefore, the worst case IRA will occur right before that, at $t - 1$, because this is just before the group is cycle counted again. If no group is allowed to drop below a certain level of inventory accuracy R, then this relation is given by

$$(1 - L')^{t-1} = R\,. \tag{8}$$

Therefore, substituting (7) into (8), the equation can be solved for the appropriate sample size n given by

$$n = \frac{N}{\left(\frac{\ln(R)}{\ln(1 - L')} + 1\right)}. \tag{9}$$

This approach could be used to approximate the sample size for a sought after IRA.

2.2.2 Normal approximation of binomial distribution

Another approach for achieving an accurate IRA, on which actions can be taken, while mainly using a non-random cycle counting method, would be to take random samples at the end of the month. As stated earlier, because the products are either accurate or inaccurate, they will follow a binomial distribution. The binomial distribution can be approximated by a normal distribution, by using the central limit theorem [20]. This states that when a large sample is taken from a population, either finite or infinite, with an unknown probability distribution, it can be approximated with a normal distribution. Therefore, when the sample size is large an approximate 100(1- α)% confidence interval can be constructed on the proportion p of the population that belongs to a certain class by

$$\hat{p} - z\sqrt{\frac{\hat{p}(1 - \hat{p})}{n}} \leq p \leq \hat{p} + z\sqrt{\frac{\hat{p}(1 - \hat{p})}{n}}\,, \tag{10}$$

where $\hat{p}$ is the proportion of observations in a random sample of size n that belong to the class of interest and z is the upper $\alpha/2$ percentage point of the standard normal distribution. Furthermore, this approximation requires $np \geq 5$ and $n(1 - p) \geq 5$ to be a good approximation of the binomial distribution. Typically, in literature, it is common to denote $z_{\alpha/2}$ instead of z.

Therefore, with a chosen confidence interval that the error E in using $\hat{p}$ to estimate p less than a certain level, the appropriate sample size can be calculated using

$$n = \left(\frac{z}{E}\right)^2 p(1-p)\,. \tag{11}$$

However, this can only be used if there is an estimate of p available based on previous $\hat{p}$. Therefore,

$$n = \left(\frac{z}{E}\right)^2 \left(\frac{1}{4}\right) \tag{12}$$

gives a very conservative approach, which gives the maximum sample size n for estimating p with $\hat{p}$ within error E with a certain confidence interval [20], where p in (11) is chosen to be 0.5. Therefore, at the end of each month a random unbiased sample of size (11) or (12) can be taken to assess the IRA at that given point in time. In this approach, the cycle counting system can find the most inaccurate products in the best way possible, without being bound to random samples, while at the end of the month the IRA can accurately be assessed. This properly attained IRA can then be used: to base action on for change in the cycle counting system, to base action on for change in warehouse processes, for accounting and reporting purposes.

2.3 Vehicle routing algorithm

There are multitude of different path planning problems, the most well-known being the Traveling Salesman Problem (TSP). In which, a single salesman has to visit multiple cities, i.e. nodes, and return to his starting point, while minimizing the total travelled distance. When multiple salesman, or in this case vehicles, are used it becomes the Vehicle Routing Problem (VRP). Furthermore, when a fleet of different vehicles, a heterogeneous fleet, is used the abbreviation HVRP is used [21].

2.3.1 Problem formulation

A graph $G = (V, A)$ is given, where $V = \{0,1, \dots . n\}$ is a set of $n + 1$ vertices and A is the set of edges or arcs. The depot is represented by vertice 0 and the set $V' = V \backslash \{0\}$ consists of n customer nodes. Every customer $i \in V'$ has a requirement of q_i units of supply from the depot. A heterogeneous fleet of m different types of vehicles are present at the depot, with $M = \{1, \dots, m\}$. The amount of available vehicles m_k at the depot, for each type $k \in M$, with a capacity of Q_k. The fixed cost of a vehicle, for example the rental of a vehicle, is denoted by F_k. Furthermore, the non-negative dependent routing cost of each vehicle type $k \in M$ for edge $(i, j) \in A$ is denoted by c_{ij}^k. In addition, a route R, of vehicle $k \in M$, is given as $R = (i_1, i_2, \dots, i_{|R|})$, with every route starting and ending at the depot as $i_1 = i_{|R|} = 0$ and $\{i_2, \dots, i_{|R|-1}\} \subseteq \mathrm{V}'$. The route R is a simple circuit in G [21]. Therefore, the Heterogeneous (H) Vehicle Routing Problem (VRP), with Fixed (F) and Dependent (D) cost, i.e. HVRPFD, is formulated as:

$$\min_x \sum_{k \in M} F_k \sum_{j \in V'} x_{0j}^k + \sum_{k \in M} \sum_{\substack{i,j \in V \\ i \neq j}} c_{ij}^k x_{ij}^k \tag{13}$$

Subject to

$$\sum_{k \in M} \sum_{i \in V} x_{ij}^k = 1 \qquad \forall j \in V' \tag{14}$$

$$\sum_{i \in V} x_{ip}^k - \sum_{i \in V} x_{pj}^k = 0 \qquad \forall p \in V', \forall k \in M \tag{15}$$

$$\sum_{j \in V'} x_{0j}^k \leq m_k \qquad \forall k \in M \tag{16}$$

$$\sum_{i \in V} y_{ij} - \sum_{i \in V} y_{ji} = q_j \qquad \forall j \in V' \tag{17}$$

$$q_j x_{ij}^k \leq \; y_{ij} \leq (Q_k - q_i) x_{ij}^k \qquad \forall i, j \in V, i \neq j, \forall k \in M \tag{18}$$

$$y_{ij} \geq 0 \qquad \forall i, j \in V, i \neq j \tag{19}$$

$$x_{ij}^k \quad \in \{0,1\}, \qquad \forall i, j \in V, i \neq j, \forall k \in M \tag{20}$$

where (2) ensures that the customer is visited only once and (3) that if the vehicle visits the customer it must also depart from that customer. In addition, constraint (4) ensures that the number of available vehicles does not exceed the maximum amount of vehicles, for each vehicle type. Furthermore, the demand of the customer should equal the difference in capacity before and after the customer is served by the vehicle (5). Lastly, the vehicle capacity should never be exceeded (6) [21].

There are a number of different types of VRP's and they are denoted with different abbreviations within literature. Whereas, Fleet Size and Mix VRP, is a variant where the number of available vehicles are infinite and is denoted by FSM. The Capacitated ARC Routing Problem (CARP) is a literature section that researches routing problems where edges, or arcs, need to be served instead of nodes. In addition, the Periodic VRP is a class in which customers have specific order frequencies and a planning needs to be made for multiple vehicles for multiple time periods [21].

Furthermore, other variants of the VRP are: the case where the vehicles have limitations on the customers that can be served, are denoted with Site Dependent (SD) VRP; the case when there are multiple depots from which vehicles can depart (MD); the case when vehicles have relatively small capacity and need to make multiple trips to the depot to serve all customers it is with Multiple-trip (MT); and the case when customers have specific time slots in which they want their order it is with Time Window (TW) [21].

Both the TSP and VRP are NP-hard. Therefore, most of the afore mentioned variations of the VRP are NP-hard. NP-hard problems are not solvable in polynomial time [22]. However, exact methods are available, but perform poorly on instances with large sizes. Therefore, many heuristic and metaheuristic algorithms are developed [23]. There are two groups of heuristic algorithms: Construction and improvement procedures. Construction procedures build solutions by adding more nodes to the solution in each step. While, improvement procedures, as the name states, improve feasible solutions [24]. Metaheuristics are generic strategies that efficiently guide a search process, with the goal of finding (near) optimal solutions. Generally, abstract concepts from other research areas are used for the search techniques. Metaheuristic algorithms are not problem specific [25].

2.3.2 Heuristic algorithms

The GENIUS algorithm [24] is a heuristic for the TSP. It consists of two procedures, the GENI procedure and the Unstring and String procedure. The GENI procedure makes the tour by considering inserting customers, arbitrarily chosen, into the route in two ways, see Figure 1 and Figure 2. The first insertion type, is inserting the unrouted customer into the route in-between two adjacent vertices in the route. The second insertion type, is inserting the new customer in-between two customers that are not adjacent. To limit running times only the p closest neighbours in the tour, of the unrouted customer, are considered for inserting the new customer between. After every insertion, the neighbourhood p is updated. The GENI procedure is stopped when all customers are included into the route. The Unstringing and Stringing part is a post optimization procedure. It works with the analogy of unstringing and stringing beads on a cord. The

procedure is similar to the insertion of the GENI algorithm, but reverse. From a complete tour, it considers removing a customer and reconnecting the tour in the two possible ways, see Figure 1 and Figure 2. The customer is removed if it improves the cost of the tour. The procedure stops if there is no further improvement found.

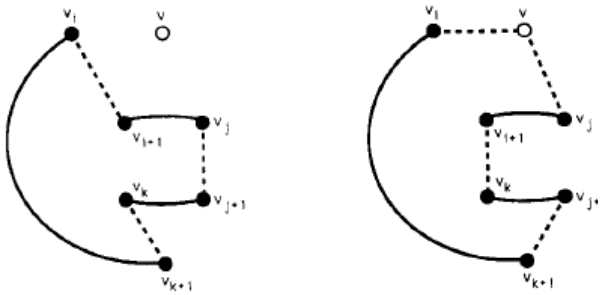


Figure 1: Type 1 insertion or in reverse type 1 unstringing [24].

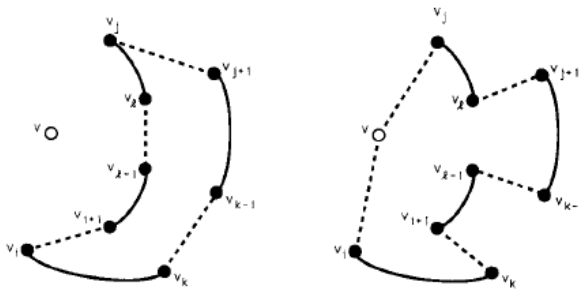


Figure 2: Type 2 insertion or in reverse type 2 unstringing [24].

The Sweep algorithm [24] is a heuristic capable of solving VRP's. The algorithm makes routes for a homogenous fleet of vehicles by adding customers to the route based on their "polar coordinate angel". The polar coordinate angel is defined as the angel between two coordinates on a two-dimensional plane and the reference point, in this case the depot. The polar coordinate angle $AN(I)$ of location I is calculated as

$$AN(I) = \arctan\left(\frac{Y(I) - Y(1)}{X(I) - X(1)}\right), \tag{21}$$

where location 1 is the depot, X and Y are the respective coordinates, $-\pi < AN(I) < 0$ if $Y(I) - Y(1) < 0$ and $0 \leq AN(I) \leq \pi$ if $Y(I) - Y(1) \geq 0$. Customers will be added to a route, starting from the lowest angle, until the maximum holding capacity or the maximum traveling distance of the vehicle is reached. Thereafter, the routes are improved by deleting the customer KII closest to the depot and closest to the next route, from the current route, and adding the customer JJX closest to the last added customer. Customer KII in the current route is determined by

$$R(I) + AN(I)AVR\,, \tag{22}$$

where $R(I)$ is the radius from the depot to customer I and AVR is the average radii for every location in the route. This customer will be left to be added in the next route construction. If this rearrangement of customers does not violate any constraints, the customer JJI closest to JJX will be added to route as well. This procedure continues until no customers can be added to the route and a new route construction is started. Making routes following this procedure does not produce the best results. However, it is a very fast procedure for selecting customers and produces good results [24]. During each step of the algorithm, the distance of the routes are improved by a TSP heuristic. Lastly, when all customers are routed the algorithm attempts to improve the solution by performing a "backwards SWEEP", where the same procedure is done with the first customer being the last and the last being the first.

2.3.3 Metaheuristic algorithms

There are many different forms of meta-heuristics [21]: Ant colony optimization, genetic Algorithms, Greedy randomized adaptive search procedure, simulation annealing, Tabu search, variable neighbourhood search, Adaptive Memory Programming [26] and hybrids (combinations of the above mentioned). However, not all are explained in depth.

The corner stone of the planning problem is the limited battery capacity of the robots. Therefore, the initial forms of the VRP are explored that can handle this, which is the VRP Multi-trip problem, also called Multi-tour or Multi use. A number of the metaheuristics can be used to derive feasible solutions for this problem. However, for obtaining the most efficient cycle counting system, the best algorithm, i.e. that produces the best solution in the least amount of computational time, has to be used. The best known algorithms are compared in [27], where the algorithms are compared in a number of benchmark incidences. Seen from Table 1, the algorithm of OV07 algorithm has far superior calculation times. Furthermore, from Appendix B, it can be seen that OV07 achieved the most amount of feasible solutions. The OV07 is an Adaptive memory Programming and Tabu Search hybrid meta-heuristic algorithm given in [28].

Table 1: Benchmark CPU times, in minutes, of VRP multi-trip algorithms [28].

Problem	# of cust.	z^*	TLG96	BM98	PS04	OV07	SP07
CMT-1	50	524.61	5	2.5	1.8	0.16	0.26
CMT-2	75	835.26	7	5	5.5	0.33	0.50
CMT-3	100	826.14	24	10	13.8	0.40	1.17
CMT-4	150	1028.42	51	25	16.4	0.88	3.44
CMT-5	199	1291.44	66	62.5	40.9	1.68	8.06
CMT-11	120	1042.11	45	25	40.5	0.37	18.86
CMT-12	100	819.56	23	10	2	0.37	0.75
F-11	71	241.97	26	2.5	4.3	0.13	1.55
F-12	134	1162.96	75	80	13.5	0.50	9.73

The Tabu search meta-heuristic works by modifying a solution locally and repeatedly, while these modifications are memorized to avoid getting the same solution multiple times. Therefore, the characteristics of the modifications are memorized in a "tabu" list, which prohibits the use for a specified amount of iterations. An initial, good or bad, solution is searched for [26]. Thereafter, that solution is improved iteratively by local modifications. However, the modifications do not necessarily have to improve the solution at each iteration, but it directs the search to a good subset of solutions. The basic Tabu Search algorithm looks like this [26]:

1. An initial solution is generated s_0; initialize the memory; $k \leftarrow 0$; $s^* \leftarrow s_0$.
2. While the stopping criterion is not met do:
    - **a.** Choose s_{k+1}, a neighbour solution of s_k, using data stored in the memories;
    - **b.** If s_{k+1} is better than s^*, then $s^* \leftarrow s_{k+1}$;
    - **c.** $k \leftarrow k + 1$ **;**
    - **d.** Update the memories.

The stopping criteria is either a fixed number of iterations or number of iterations without an improvement of the best solution. The above steps of the algorithm can differ in complexity. The tabu search algorithm has a short-term memory, the taboo list, but also can be adapted with a long-term memory by applying Adaptive Memory Programming, making it a hybrid meta-heuristic

Adaptive Memory Programming (AMP) works on the principal that "good solutions may be constructed by combining different components of other good solutions" [28]. It begins with a set of solutions of the solutions that are produced are memorized. Secondly, a solution is constructed using the information in the memory. Thirdly, the solution is improved using a local search algorithm or another metaheuristics. Lastly, the new solution is added to the memory or is used to update the structure that memorizes the search history [26].

In [28] a combination of the AMP and Tabu search metaheuristics is used. The research aims to solve the VRP multi-trip problem, which compared to (13), does not take into account fixed cost in the cost function and only takes one type of vehicle into consideration. The algorithm used to solve the problem contains the following four stages:

1. Memory initialization:
   The memory is initialized by generating I solutions and storing them in M. Every solution is made by applying one iterations of the Sweep algorithm, which will be explained later, starting from a random node. Before storing the solution into M, the solutions are improved by the Tabu Search algorithm, in step 4.
2. Memory update:
   Each solution is labelled with the amount of overtime first and the total rout cost second. Thereafter, the solutions are sorted lexicographically. Therefore, the first slots of the memory will belong to the feasible solutions with the lowest total cost. Only limited amounts of solutions are stored in the memory, because if the memory gets to large the probability of selecting good routes, which will be explained in the next step, is diminished.
3. Solution construction:
   A temporary memory $M' \leftarrow M$ is set. New solutions are built by selecting routes r from the memory M' in a probabilistic manner and deleting routes form M' that have customers in common with r. The best routes will be picked more often, because a higher probability is assigned to the solutions in the first slots of M'. The now un-routed customers are again routed with the Sweep algorithm [29], started from a random customer, and the routes added to the memory. The solution construction procedure is stopped when all customers are routed.
4. Tabu Search procedure:
   The solutions that are constructed are improved by the Tabu Search procedure. In this stage the violation of capacity and overtime constraints are allowed. However, they penalize the objective function with

$$f_p(s) = f(s) + \alpha \sum_{r \in R(s)} (q_r - Q)^+ + \beta O(s)\,, \tag{23}$$

   where α penalizes the capacity contraints and β the total overtime constraints. The parameters α and β change every iteration, using the Taburoute algorithm [30]. In this case, if the solution s satisfies the capacity constraint α is divided by 2 and multiplied by 2 if this is not the case. The same holds for parameter β. However, there are lower and upper bound given for the parameters.

   Each iteration of the Tabu Search begins with assigning routes to vehicles, with a simple bin packing heuristic, the Best Fit Decreasing heuristic [31]. This algorithm assigns the longest route available to the "emptiest" vehicle, i.e. the vehicle with the least total routing times. If the assignment of routes is infeasible, i.e. when there is overtime, another procedure runs. Where the vehicle with the longest total time tries to swap routes with other vehicles such that there is less overtime. This algorithm returns a current best solution with all routes assigned to vehicles.

   Thereafter, for each route it is checked if another route is within their neighbourhood. Routes are within each other's neighbourhoods if one of the following moves can be done: if two customers can be swapped between two routes or if a customer can be inserted into another route. Therefore, it is checked for each node if it can be swapped or inserted into the route of five of the nodes nearest to it. When a move reduces f_p, it is made and that move is then declared "Tabu" for a random amount of iterations. The "Tabu" status can be overridden if the move improves the best-

known solution. This is called "aspiration criteria". In addition, whenever a move is made the assignment heuristic is run again.

When a better solution is obtained, each of the routes that were changed by the last move will be rearranged using a post-optimization procedure called Unstringing and Stringing [24]. Thereafter, the α and β parameters are updated. The Tabu Search procedure is run TS^{iter} amount of times and the best solution is returned. With "the best" meaning the least infeasible or of all the feasible solutions the solution with the least total cost.

The authors state that their algorithm distinguishes from others by performing the assignment of routes to vehicles in every iteration of the Tabu Search. In the article, the algorithm is tested against 52 benchmark instances. In which, it produced 43 feasible solutions out of 52 instances, when the maximum time horizon for all vehicles was relatively low. However, when the maximum time horizon for all vehicles was increased it produced feasible solutions for all instances.

As can be seen from Appendix B, the AAB07 algorithm performs considerably worse than the OV07 algorithm discussed previously. It obtains less feasible solutions and the lengths of the infeasible solutions are higher. However, the AAB07 algorithm described in [32] is capable of solving more complex problems. The algorithm is designed to solve Site Dependant (SD), Multi trip (MT), Periodic (P) Vehicle Routing Problems. Site dependant means that some or all vehicles have restrictions on which customers they can serve. They propose a modified Tabu Search algorithm to solve this large problem and call it TS-ABB. The algorithm consists of two stages.

1. The initial solution
   First, the customers are sorted in an ascending order in a similar manner as the polar coordinate angles as discussed previously. However, the authors use the Euclidian coordinates of the customers. If the angle between customers is similar, an arbitrary order is assigned. In addition, a random delivery pattern is assigned to each customer. Thereafter, the initial solution is derived using the GENIUS heuristic [24] for each sequence of customers for each day. The GENI part of the algorithm creates a solution by taking a tour of three random vertices and adding vertices by one of the 2 insertion types, as discussed previous previously, that adds the least amount of time to the total tour. When the time constraint of a vehicle for a day is violated by inserting a new customer, it is assigned to the next vehicle. If the capacity constraint is violated the customer is left for another day on the same vehicle or for another vehicle on the same day. This procedure does not guarantee a feasible solution.

2. Search procedure
   Here two objective functions are used. The first function denotes the sum of the routing cost and two penalty term for the over-capacity and over-time, just as in (23), and the second the routing cost of the respective solution. In the first function, α and β are, just as in the previous algorithm, parameters that adjust during the procedure. If the current solution has a feasible capacity, then α is divided by $1 + \delta$ and otherwise multiplied by it, where δ is a positive paramter. The parameter β is adjusted in the same manner. An advantage using this formulation is that the algorithm will evolve through infeasible and feasible solution. The second advantage is that the search can run with relatively simple moves, which gives algorithmic advantages.
   From the initial solution, the search moves to neighbouring solutions, for a chosen number of iterations. The neighbouring of the solution comprises of solutions that make these moves:

1. If a customer can be **removed** from one route on specific day and be **inserted** into another route on the same day using the GENIUS heuristic. The route can belong to the same vehicle or another vehicle of the same type (because of the site dependency).
2. If a pattern of a customer can be **replaced** with another pattern. The customer is removed from each route in the first pattern and added to that route for each day in the new pattern, such that the f(s) function is minimized using the GENIUS heuristic. For similar days between patterns the customers are not moved.

Every solution has an attribute set comprising of the customer i, with vehicle k, in route r, on day l. If a move has been made, it is declared "Tabu" for a number of iterations. In addition, every attribute (i , k, r, l) is assigned an aspiration level, as the minimum cost of the solution. Furthermore, a subset of attributes, that are not Tabu and have a lower cost than the current solution, is made. For diversification, each solution that increases the objective function is penalized with $\gamma\sqrt{nm\omega t}$. The $\sqrt{nm\omega t}$ term compensates for the problem size, where n is the amount of customers, m the amount of vehicles, ω the maximum number of daily routes per vehicle and t the amount of days per period. Furthermore, γ is a constant factor to adjust the intensity of the diversification.

In [23] a MDVRP with Inter-depot trips (MDVRPI) is solved using a Tabu Search algorithm and a form of AMP. In which, trips between neighbouring depots are allowed, where the capacity is recharged, as long as the vehicle ends at the depot where it started.

The TS procedure is similar of that of the previous algorithm, where: the GENI algorithm is used to obtain solutions, a cost function similar to (23) is used and diversification penalty function is used. Furthermore, when the route starts and ends at the starting depot, the distance of the tour is obtained solving the TSP. However, if the route ends at an intermediate depot, the distance of the route is obtained by solving for the shortest Hamiltonian path.

1. Generating a solution pool
   Three types of sub-problems are solved: VRP, inter-depot sub-problems and MDVRP. For the VRP sub-problem, the customers are assigned to depots by drawing a circle around the depot (in a planar case). The customers assigned to the depot are: those within the circle and those who have the depot as its closest depot. This can be seen Figure 3, where the black squares represent the depos, the plus symbols customers and the small circles the customers that are assigned to the depot, because of proximity to the depot. The inter-depot sub-problem is solved for every pair of depots. Customers inside of an ellipse, centred on the middle of the line between the two depots, are assigned to the pair of depots for the inter-depot sub-problems, see Figure 4. In addition, customers with close proximity, outside of the ellipse, to the pair of depots are included for the inter-depot sub-problems as well.

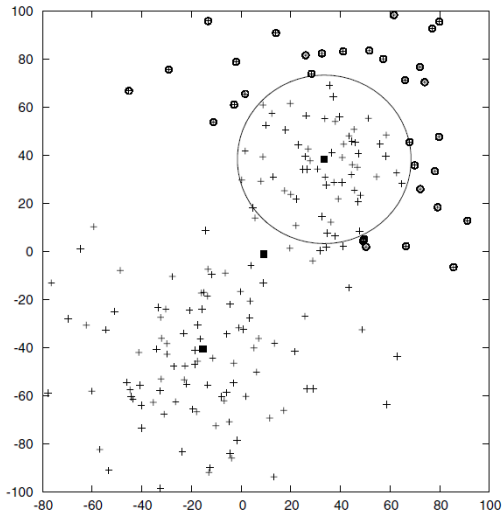


Figure 3: Assigning customers for VRP sub-problems [23].

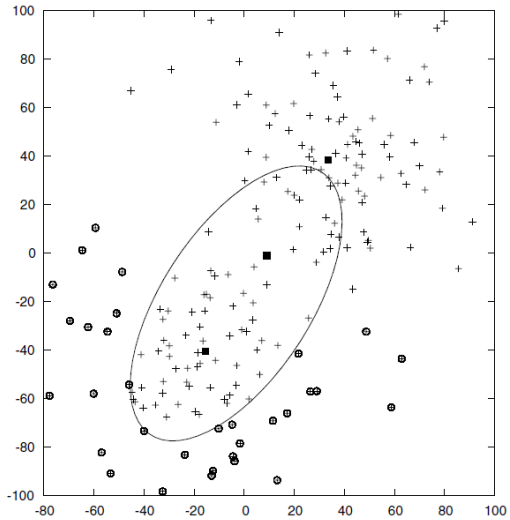


Figure 4: Assigning customers for Inter-depot sub-problems [23].

One of the main reasons for including customers in the sub-problems by this procedure is that the depot closest to the customer has a high probability of serving the respective customers in an optimal solution.

2. Route generation
   The initial routes are generated within the sub-problem are generated using the SWEEP algorithm of [33], which is similar to the SWEEP algorithm used in the OV07 algorithm. However, here the customers are added using the GENI heuristic. Infeasible solutions are also considered. Therefore, similar as (23) the routes are labelled with a cost function, where alpha and beta are updated dynamically. However, only a specific portion of feasible solutions is kept.
3. Set partitioning
   With the portion of feasible routes, it is proposed to solve the MDVRP with a set portioning algorithm.
4. Post optimization
   Firstly, the Tabu search procedure is applied to the solution, the set of feasible routes from the set partitioning algorithm. Secondly, empty routes are removed, meaning the routes where a vehicle only travels between depots without visiting customers. Thirdly, for all non-empty routes, the edges connecting to a depot are removed and a new least cost rotation is identified. Lastly, the TS procedure is applied to the new solution. This procedure is repeated until no empty routes exist in the solution.

3 Robot/cycle counting planning algorithm

In this chapter, the robot routing algorithm is constructed. Therefore, it is first described what the algorithm should be able to do. Thereafter, an algorithm is chosen which fulfils the requirements and is best suited in terms of running time, feasible solutions and amount of customers it can schedule. Furthermore, the algorithm is constructed and each step is explained. Lastly, it is determined which cycle counting methods are compatible with the robot routing algorithm and for those methods procedures are constructed.

Firstly, the requirements of the robot routing algorithm are described. The planning algorithm needs to generate routes to scan products, for the purpose of cycle counting, as efficiently as possible. In addition, the algorithm needs to be generic enough that it can be implemented with any combination of different robots and warehouse layouts, with minimal adjustments. Therefore, the problem is defined in the following statements and assumptions:

1. Multiple scanning vehicles
2. Heterogeneous fleet of scanning vehicles
3. Limited battery capacity
4. Only 8 hours to perform cycle counting each day
5. 3D x, y, z-coordinates, with obstacles
6. No external factors limiting the speed of the robot (no drift fields)
7. Symmetric undirected graph
8. The robot moves at a constant speed
9. No dynamism in the planning algorithm
10. The barcode scanning of the products can be performed from a central point of the shelf
11. Possibly groups of products need to be scanned first, but no sequence constraints
12. No online corrections

(1) is defined, because some warehouses are very large and it would be unrealistic to assume that one drone can scan sufficient products for cycle counting purposes. Furthermore, (2) is defined, because the planning algorithm has to be generic and there are companies that pursue scanning drones and scanning robots that drive, as suggested in the introduction. Therefore, a combination of scanning robots that drive or fly, with different traveling capacities, must be possible within the planning algorithm. However, it's assumed that all the robots can scan all products.(3) the planning algorithm has to account for the limited active duration and recharging times, because most robots have a very limited battery capacity and long recharging times. As can be seen as an example in Appendix A, most commercial drones nowadays have a maximum flying duration of 30 minutes and a recharging time of 60 minutes. This research is based on the idea that the robots will start cycle counting after business hours and stop before the next business day starts (4), as mentioned in the introduction. Assuring no employees are present at the time of cycle counting it is assumed that the system will count 8 hours per day. Most large warehouse have rows of storage racks. Therefore, the distance between the products that need to be scanned have to be considered in 3 dimensions and with obstacles (5). However, since the system operates in a closed environment, the warehouse, it is assumed that there are no environmental factors limiting the speed of the robots (6) and that the shortest path between any 2 products can be travelled in the same time from both directions (7). Therefore, it is also assumed that the robots travel at a constant speed (8). In addition, it is assumed there is no dynamism in the planning algorithm, i.e. no moving obstacles or probabilistic events (9). Moreover, the company is assumed to be out of operation during the cycle counting process. Furthermore, it is assumed that the products can be scanned from a central point of the rack (10), with the use of barcode scanners. In light of stratified forms of cycle counting, like ABC or supervised learning cycle counting, it is possible that product groups will need to be scanned first (11). However, there are no sequence constraints for scanning the

products. Lastly, since there are no specific scanning sequences and mission critical tasks, it is assumed efficient to not perform online corrections in the planning during the cycle counting process (12).

3.1 Robot routing algorithm

From the assumptions and system characteristics the robot cycle counting system is defined as a Heterogeneous Vehicle Routing Problem with Fixed cost, Dependant cost and Multiple-trips, because the corner stone of this planning algorithm is the limited battery capacity of the robots, which forces the system to make multiple-trips within the available cycle counting time. Furthermore, VRP problems are NP-hard. Therefore, exact algorithm will not be feasible. Moreover, the amount "customers", in this case products, will be high in large warehouses. Lastly, metaheuristic algorithms perform better than heuristic algorithms. Therefore, a metaheuristic algorithm is chosen for the robot routing algorithm.

Therefore, the Adaptive Memory Programming and Tabu Search hybrid metaheuristic algorithm [28] is chosen as the basis for the robot routing algorithm, because from the comparison in [27] it obtained the most feasible solutions in the least amount of computational time.

However, the algorithm in [28] can only solve VRP-MT problems. Therefore, the algorithm is adjusted to be able to solve the HVRPFD-MT needed for the robot cycle counting system. Firstly, the maximum scheduling time **T** for the day and the maximum traveling distance of the robots **MaxD** are set to be different for every robot, such that a heterogeneous fleet of robots can be scheduled. Secondly, fixed costs per product scanned is added to the cost function in the Tabu Search procedure, to account for the time it takes to scan a product. Furthermore, the capacity constraint in (23) is removed, because the robots do not physically carry the products. The structure of the cost function and the procedure for updating the penalty coefficient α is kept the same, since the Tabu Search procedure in the other two algorithms, discussed in the literature review, use similar techniques and no benefit is seen in adding a second cost function as in [32]. Furthermore, in the best-fit decreasing procedure for assigning routes to robots, also recharging times are packed into the schedule of the robot, after every route assigned to the robot.

The SWEEP algorithm [29] was designed for 2D problems. However, it is still applicable in this 3D case, because the products within the warehouse are stored in racks. Products on the same side of the rack will have the same polar angle and will most likely be added to the same route. In addition, the GENIUS algorithm, used in the TS-ABB algorithm for creating routes, assigns the customers arbitrarily. Therefore, SWEEP algorithm [29] is still an effective way of creating efficient routes, because the products are in close proximity of each other in the rack and is therefore used in the robot routing algorithm.

Within the SWEEP algorithm, single routes, which are separate TSP's, are improved using the 2-opt algorithm [34], explained, after the SWEEP algorithm**.** Although, the 2-opt heuristic is one of the most basic algorithms for solving the TSP, it is widely used and "achieves amazingly good results on "real world" Euclidian instances both with respect to running time and approximation ratio" [34]. Therefore, the 2-opt heuristic is used in this research for improving the separate TSP's in the SWEEP heuristic and as post-optimization in the Tabu Search procedure.

Lastly, inspired by the GENI algorithm [24] a third type of action is added to the Tabu Search procedure. In the GENI algorithm two types of insertions are considered, where the tour orientation is different in both cases. The Tabu search algorithm in [28] only tries to insert a customer before its p closest neighbours and applies post optimization if the action improves the solution. Therefore, if the insertion after one of its p closest neighbours in the tour would improve the solution; it is not considered, because the post optimization of the route is not applied, since its insertion before did not improve the solution. For this

purpose insertion before and after its p closest neighbours is considered in the Tabu Search procedure, next to swapping customers with its p closest neighbours.

The outline of the AMP and Tabu search algorithm is already explained in the Chapter 2.3.3. Therefore, the pseudo code of the algorithm is given, with a short discerption. The real code is shown in Appendix C and was coded using MATLAB.

Initialization algorithm

Firstly, the algorithm is initialized by constructing a traveling time matrix d, which indicates the traveling time of the robot from the location of every product to the location of all other products. In cases without obstacles, each element in d is calculated by dividing the 3-dimentional Euclidian distance by the traveling speed S

$$d_{ij} = \frac{\sqrt{x_j^2 - x_i^2 + y_j^2 - y_i^2 + z_j^2 - z_i^2}}{S}, \tag{24}$$

where x, y and z are the product coordinates of the respective dimensions; from a central point in the warehouse and $d_{ij} = d_{ji}$. The same is done for the traveling time from the recharging depot to all products $d0$. Secondly, the **p** closest neighbours, in traveling time, of every product are determined, to be used later in the Tabu Search procedure.

Memory Initialization

For the algorithm to start, the memory needs to be filled with routes, from which solutions can be build. A solution is defined as a schedule for all robots, which specifies the products that are scanned in which route, the routes taken by what robot and the recharging time between routes. Therefore, a complete solution contains all the products in the cycle counting sample that need to be routed and multiple routes.

Infeasible solutions are allowed during the main procedure. Therefore, overtime, the time exceeding the maximum schedule time **T**, is allowed, but penalized. However, the routes are ranked in the memory, first on the least amount of overtime and secondly on total cost of the total solution. Based on this ranking a probability is assigned to each route. The adaptive Memory Programming algorithm will pick routes based on this probability and construct new solutions. The probability of choosing the i-th route is

$$\frac{|M2| + 1 - i}{|M2|(|M2| + 1)}, \tag{25}$$

where $|M2|$ is the amount of routes currently in the memory. The minimal amount of initial routes is specified as "**In**". Furthermore, the memory is filled with routes from all different types of robots, because a heterogeneous fleet of robot is allowed. Therefore, routes are made with different maximum traveling times **maxD**, that suit the different types of robots, and are stored in the memory.

```
Memory Initialization
While Amount of routes in memory<In
    For i=1: amount(types of vehicles)
        Main procedure
        Assign all routes with overtime and cost of the solution
        Add routes to memory
    End
    Sort Memory on Overtime descending and cost descending
    Assign probability of getting picked based on amount of routes in memory
End
```

Adaptive memory programming procedure

As explained previously, the AMP procedure creates new solutions based on routes in the memory. Routes are picked based on the assigned probability and all routes that have products in common with the chosen route are deleted. Each iteration, the probability is reassigned to the routes that are left in the memory. If the memory is empty, but there are still products that are not routed, the main procedure only makes routes for the unrouted products. However, all routes are assigned to a robot and improved. The routes of the newly constructed solution are labelled with the overtime and cost of the solution and stored in the memory. The memory has a maximum amount of **M**. The memory is capped, otherwise the better routes have a low probability for being chosen. The main procedure, including the SWEEP algorithm, is explained in the next paragraph.

```
AMP algorithm
For 1:AMPiter
    M2=M
    Assign probability to routes
    While M2 is not empty
        Chose route from M2
        Assign route to solution
        Delete all routes that have products in common with chosen routefrom M2
        Reassign probability on routes left in M2
    End
    If M2 empty and there are customers unrouted
        SWEEP(unrouted products)
    End
    Main procedure (excluding SWEEP)
    If new solution better than best solution
        Set New solution to best solution
    End
    Add routes of new solution to memory
    Sort Memory on Overtime descending and cost descending
    If Amount of routes in memory>M
        Memory=Memory(1:M)
    End
End
```

The solution is perceived better if the overtime of the solution less than the overtime of the best solution or if the overtime is equal to the best solution and the cost is less than the best solution.

Main procedure

The main procedure consists of three algorithms. Firstly, routes are created following the SWEEP algorithm, explained in Chapter 2.3.2. Secondly, the routes are assigned to robots using the Best Fit Decreasing algorithm, for an initial solution. Lastly, the solution is further improved using the Tabu Search algorithm.

SWEEP

As explained previously the SWEEP algorithm adds routes based on the polar angle and each iteration optimize the route with the 2-opt algorithm. If the route exceeds the robot's maximum traveling capacity **MaxD**. Then a procedure starts, which tries to add the nearest neighbour of the last added route and remove the route closest to the depot and to the next route.

SWEEP algorithm

```
Sort products in order of descending polar angle from the depot
Pick random product
Sort products from randomly chosen product -> TOBEROUTED
While there are products to be routed
        Add next product v to route L{r} from  TOBEROUTED
        Optimize route with 2-opt(route L{r})
        If LD>MaxD
           Find closest neighbour JJX to last added product
           Find the closest neighbour JJI to JJX
           Find product KII closest to depot and closest to next route
           D2=L{r} without KII and with JJX -> 2-opt(D2)
           If LD2<=MaxD and JJX is a member of the next 4 products TOBEROUTED
                D3= make route with next 5 products from  TOBEROUTED ->2-opt(D3)
                D4= D3 without JJX and KII added ->2-opt(D4)
                If LD{r}+LD3<=LD2+LD4 and JII is member of next 4 products TOBEROUTED
                        D5=D2 with JII added ->2-opt(D5)
                        D6=D4 without JII ->2-opt(D6)
                End
            End
            If  LD2<=maxD and LD(r)+LD3>=LD2+LD4
                Remove KII from the route
                Add JJX to the route
            Else if LD5<maxD && LD5+LD6<LD(r)+LD3
                Remove KII from the route
                Add JJX to the route
                Add JJI to the route
           Else
                Start a new route r=r+1
           End
        End
End
```

2-opt

This heuristic gradually minimizes the total distance of an initial tour, where every vertice is visited ones and the tour starts and ends at the same vertice. The 2-opt heuristic improves the tour by exchanging two edges if its implementation decreases the length of the tour. Therefore, the algorithm searches through all existing edges of the tour by trying to break two existing edges, $\{w_1, w_2\}$ and $\{v_1, v_2\}$, and reconnecting the vertices as $\{w_1, v_1\}$ and $\{w_2, v_2\}$ [34]. By applying this exchanging procedure, the tour is never broken. The 2-opt heuristic stops, when no further improvement is found.

2-opt algorithm

```
While z<zmin
    For i=1:(all products in route L{r})-1
        Find first two products in route that share an edge (or the depot and a product)
        For j=i+1:all products in route L{r}
            Find next two products in route that share an edge
            Calculate distance difference z, if edges are swapped
            If z<zmin
              Store products that need to be swapped in the route
            End
        End
    End
Swap products in route L{r}
z=zmin
End
```

Tabu Search

The routes are further improved by the Tabu Search procedure. Where, products are either swapped from position, inserted before or inserted after their **p** closest neighbours. After a move is made that improved the solution, the move is declared Tabu for number of iterations. However, a Tabu-move can enhance the best known solution, it can be applied, which is called aspiration [28]. Each iteration the penalty factor for overtime Alpha is updated. Alpha is divided by 2 if the solution is feasible and is multiplied by 2 when the solution is infeasible.

Tabu Search

```
Make action list for every p neighbours of every product: Swap with neighbour, insert before neighbour
in route, insert behind neighbour in route
For 1:MaxIt
    For every action in action list
        If Tabu counter==0
            Do action
            Check new distance of every route
            Assign routes to robots (Best fit decreasing)
            If new solution better than best solution
                New solution= Best solution
                Store action that improved solution
            End
        Else (Tabu counter>0 (Aspiration))
            Do action
            Check new distance of every route
            Assign routes to robots (Best fit decreasing)
            If new2 solution better than best solution ever found in Tabu
                New2 solution= Best solution
                Store action that improved solution
            End
        End
    End
    If Best solution better than best solution ever found
        Optimize all routes with in solution with 2-opt
        Assign routes to robots (Best fit decreasing)
        Update best solution ever found
    End
    Update Tabu counter of best action, draw uniform random Tabu time [thetaMin thetaMax]
    Update Alpha
End
```

Best fit decreasing

The Best Fit Decreasing algorithm bin-packs the routes in the schedules of the robots, following [24]. Where, the largest route is packed into the empties schedule. Furthermore, for every route packed into a schedule, the recharging time of robot is also added to the schedule. It is assumed that the robot always recharges after a route. Furthermore, the last 15% of most batteries take longer to recharge than the first 85% [35]. Therefore, the recharging time **Tr** is calculated by

$$Tr = \begin{cases} LD(i)\ ch_{15} & \text{if} \quad LD(i) < 0{,}15 * MaxD \\ (LD(i) - 0{,}15\ MaxD)ch_{85} + ch_{max15} & \text{else} \end{cases}, \tag{26}$$

where $LD(i)$ is the route traveling time of route i, ch_{15} and ch_{85} the time it takes to recharge the battery for every second travelled for the last 15% and the first 85% of the battery respectively and ch_{max15} the maximum recharging time it takes to recharge the last 15% of the battery. If the solutions has robots going into overtime, the algorithm tries to reduce the overtime of robot, with the highest scheduling violation first, by swapping a route with other robots.

Best Fit Decreasing algorithm

```
List all routes and their distance in LDE
LDE2=LDE
For i=1: all routes
        Assign largest route to emptiest vehicle that doesn't violate the maxD of the vehicle
        Assign recharging time Tr according to distance of route and the vehicle it is assigned to
        Keep track of which route on which robot and how much recharging time
        Remove route from LDE2
End

For i=1:all robots
    Rank the robot schedules in order of the highest schedule time first
        If robot schedule passes max schedule time T
           Best solution =current solution
           For all routes in robot schedule
                 For all other routes
                      If the route does not violate the maxD of the vehicle
                      Swap the route and recharging time with other robot
                      Check overtime improvement
                      End
                      If New solution better than Best solution
                         Keep track of routes to swap
                         New solution = Best solution
                       End
                  End
              End
        Swap routes between robots
        Update which route is on which robot
        End
End
```

3.2 Cycle counting procedures

Not all cycle counting methods mentioned in literature are compatible with the robot cycle counting system. For instance, in this research it is assumed that the robots are not used for handling the products, but rather are dedicated to cycle counting. Therefore, opportunity based cycle counting cannot be implemented for the robot cycle counting system. Furthermore, process control cycle counting is based on: location, ease of counting, obvious errors and the counter is free to choose which products to scan. The latter two concepts make process control cycle counting hard to implement in a robot cycle counting system; and will not be considered in this research.

However, location-based cycle counting could be very compatible with the robot cycle counting system, as this method will yield efficient routes for the robots, as the products will have close proximity to each other. Furthermore, the random sample method can obviously be applied. Moreover, as mentioned in Chapter 2.2, it could be necessary for the system to perform random sample cycle counts to achieve a reliable Inventory Record Accuracy. In addition, ABC and supervised learning cycle counting based on historical data can be implemented as well, because the scheduling algorithm could give priority, within choosing the cycle counting sample, to either highly priced products or products that have a high likelihood of containing an error, respectively.

However, this poses the question: Which method is most compatible with the robot cycle counting system? The location-based method will intuitively be able to scan more products, but this does not necessarily mean it will scan more erroneous products. If there are large amounts of products that need to be scanned or when the rate of change in products is high, it might be more efficient to target likely failure products with supervised learning method. Moreover, if errors of highly priced products are substantially more expensive ABC cycle counting could pose a better solution. Therefore, these four cycle counting methods are tested in combination with the scheduling algorithm.

The implementation of the random sample method is straightforward, where the products in the cycle counting sample are chosen at random. In [2],[14] and [15] it is stated that for implementing ABC cycle counting the products need to be stratified, most often, based on the price of the products using a Pareto stratification and that the expensive product groups need to be counted more often. However, in this research the strategy of diminished population is used for all the cycle counting methods. This implies that all products will be scanned ones, in one cycle counting period. If the product is not replaced in the warehouse, the products will not be counted again until the next counting period. Therefore, the cycle counting sample will be filled by prioritising products. Not a strict priority, that a product needs to be scanned before another, but rather based on a ranking of the products; a sample is chosen of the products that will be scanned. In this case prioritizing on price, location or likelihood of error in the ABC, location based and supervised learning method respectively. Again, the first two are straightforward. However, for cycle counting based on supervised learning, an algorithm needs to be constructed, to be able to predict the likelihood of a product containing an error.

In [1] it was concluded that a logistic regression supervised learning model poses a good method for predicting erroneous products . Not only did it outperform a neural network model, it also has the side benefit of the capability to take preventive measures for reducing the product errors. The preventive action can be based on the coefficients the logistic regression model assigns to the product characteristics, as discussed in Chapter 2.1. Therefore, a logistic regression model is chosen in this research to predict failure, i.e. misplacement, in products.

The logistic regression model created using [36][37]. The logistic regression model prediction is based on the log-likelihood of an assumed model of the system as:

$$\log\left(\frac{p_i}{1-p_i}\right) = \beta_0 + \beta_1 x_{i1} + \cdots + \beta_r x_{ir} \tag{27}$$

$$= [\,1 \;\; x_{i1} \;\; x_{i2} \;\; \cdots \;\; x_{ir}] \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \vdots \\ \beta_r \end{bmatrix} \tag{28}$$

$$= x_i^{\mathrm{T}} \beta \;\; , \tag{29}$$

where x_{ij} is a set of $\mathrm{j} = [1,2,\ldots,\mathrm{r}]$ variables, in this case product characteristics, over a set of $\mathrm{i} = [1,2,\ldots,\mathrm{n}]$ events. Furthermore, β_j is a set of $\mathrm{j} = [0,1,\ldots,\mathrm{r}]$ coefficients assigned to the variables, where β_0 is a stabilizing term. Rewriting equation (27) gives the log-odds of "success"

$$p_i = \frac{\exp(x_i^{\mathrm{T}}\beta)}{1+\exp(x_i^{\mathrm{T}}\beta\,)} \tag{30}$$

and "failure"

$$1 - p_i = \frac{1}{1+\exp(x_i^{\mathrm{T}}\beta\,)} \;\; , \tag{31}$$

where p_i is the probability of the event. This is also known as the sigmoid function.

Solving the model for the optimal coefficients β, for the most accurate prediction, several techniques can be used. However, in this research the Newton-Raphson method is used, because this method works well in many situations [36]. Although computationally expensive, this method converges fast to the optimal solution. A drawback of this method is that in more complex models it can get stuck in local minima [38]. However, for the purpose of this experiment this simpler model is accepted. The Newton-Raphson method is explained as follows. The outcome of the model y_i is binary; the product is either correctly positioned or incorrectly and can be described with a binomial distribution. Therefore, the likelihood of the events is given by

$$Pr[Y_1 = y_1, \cdots, Y_n = \; y_n | p_1, \cdots, p_n] = \prod_{i=1}^{n} \left(\frac{m_i}{y_i}\right) p_i^{y_i} (1-p_i)^{m_i - y_i} \;\; , \tag{32}$$

where m_i is the amount of observations at the respective event, i.e. a specific configuration of variables x_{ij}. Therefore, the log-likelihood is given by

$$L = \sum_{i=1}^{n} \left\{ m_i \log(1-p_i) + y_i \log\left(\frac{p_i}{1-p_i}\right) \right\} . \tag{33}$$

Substituting the equations **(16)** for p_i of the logistic regression model, will give

$$L(\beta) = \sum_{i=1}^{n} \left\{ m_i \log\left(\frac{1}{1+\exp\left(x_i^{\mathrm{T}}\beta\,\right)}\right) + y_i \log\left(\frac{\dfrac{\exp(x_i^{\mathrm{T}}\beta)}{1+\exp\left(x_i^{\mathrm{T}}\beta\,\right)_i}}{\dfrac{1}{1+\exp\left(x_i^{\mathrm{T}}\beta\,\right)}}\right) \right\} \tag{34}$$

$$= \sum_{i=1}^{n} \left\{ m_i \log\left(\frac{1}{1+\exp(x_i^{\mathrm{T}}\beta\,)}\right) + y_i (x_i^{\mathrm{T}}\beta\,) \right\} \tag{35}$$

$$= \sum_{i=1}^{n} \{y_i(x_i^{\mathrm{T}}\beta\,) - m_i \log(1 + \exp(x_i^{\mathrm{T}}\beta\,))\}\,. \tag{36}$$

In matrix form, the variables are written as:

$$Y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}, \qquad P = \begin{bmatrix} p_1 \\ \vdots \\ p_n \end{bmatrix}, \qquad M = \begin{bmatrix} m_1 \\ \vdots \\ m_n \end{bmatrix},$$

$$X = \begin{bmatrix} x_1^{\mathrm{T}} \\ \vdots \\ x_n^{\mathrm{T}} \end{bmatrix}, \qquad \log\left(\frac{\mathrm{P}}{1-P}\right) = \begin{bmatrix} \log\left(\frac{p_1}{1+p_1}\right) \\ \vdots \\ \log\left(\frac{p_n}{1+p_n}\right) \end{bmatrix}.$$

Therefore, the log-likelihood in matrix form becomes

$$L(\beta) = Y^T X\,\beta - M^T \log(1 + \exp(X\beta))\,. \tag{37}$$

To find the maximum likelihood estimates the derivative of $L(\beta)$, with respect to β, needs to be 0. The derivative $\dot{L}(\beta)$ is written as:

$$\dot{L}(\beta) = X^T\big(Y - M \circ P(\beta)\big)\,, \tag{38}$$

where the $\circ$ represents the elementwise product.

The Newton-Raphson method is graphically shown in Figure 5, where $\dot{L}(\beta)$ is set out against β. Each iteration of the Newton-Raphson method, a tangent is taken at the current $\dot{L}(\beta)$ and the β coefficient for the next iteration will be; where the tangent is equal to zero.

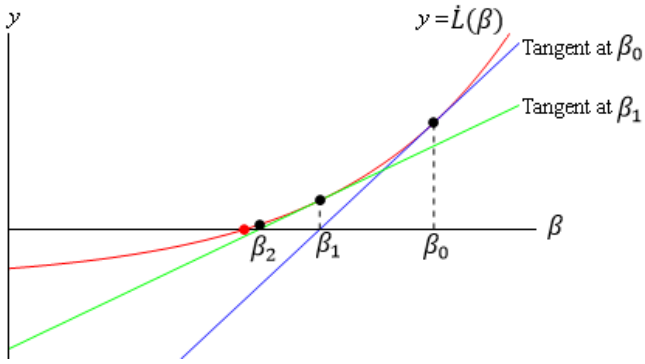


Figure 5: Newton-Raphson method iteratively changes the parameters on the x-axes, based on the first and second derivative of the log-likelihood.

The tangent is the second derivative of L. Therefore, the beta coefficients are updated every iteration of the Newton-Raphson following:

$$\beta_{it+1} = \beta_{it} - \ddot{L}(\beta)^{-1}\dot{L}(\beta)\,, \tag{39}$$

where $\dot{L}(\beta)$ is called the gradient and $\ddot{L}(\beta)$ the Hessian. The Hessian is given by

$$\ddot{L}(\beta) = -X^T v(\beta) X\ , \tag{40}$$

with $v(\beta)$ as a diagonal matrix, with the i-th element being:

$$m_i p_i(x_i, \beta)\big(1 - p_i(x_i^{\mathrm{T}}, \beta)\big)\,. \tag{41}$$

Therefore, the pseudo code for getting the maximum likelihood estimation for the logistic regression supervised learning model is given below and the real code is shown in Appendix D

Newton-Raphson algorithm

Set $\beta_0 \leftarrow$`log(sum(y)/sum(m-y))`
Set other β coefficients to 0
Set $\beta_{OLD} \leftarrow \beta$
While the chance in β > tolerance and the maximum iteration count isn't met
    Count the iteration
    Calculate probability p based on β and X
    Calculate the gradient $\dot{L}(\beta)$ and the hessian $\ddot{L}(\beta)$
    Set $\beta \leftarrow \beta_{OLD} - \ddot{L}(\beta)^{-1}\dot{L}(\beta)$
    Calculate change between β and β_{OLD}
    Set $\beta_{OLD} \leftarrow \beta$
End

It is described in [37] that the convergence of the model is highly dependent on the initial values of β. The best initial value for β_0 is

$$\log\left(\frac{\sum_{i=1}^{n}\{y_i\}.}{\sum_{i=1}^{n}\{m_i y_i\}.}\right). \tag{42}$$

Furthermore, the initial value for the rest of the β coefficients should be set to zero.

4 Case study Arox

In this chapter, it is described how the simulation, of the Arox case study, is constructed and how the robot routing algorithm is adapted for this case. As previously mentioned**,** the purpose of cycle counting is eliminating product related errors from the warehouse, whereas the robot cycle counting system can only detect an error in location of the product. Therefore, the methods ABC, location-based, logistic-regression and random sample cycle counting, are tested in combination with the robot routing algorithm to see which method performs best. The performance here is measured by the percentage of erroneous products the cycle counting method "catches" as opposed to the amount of errors in the system. Lastly, the algorithm is tested based on solution quality and processing time.

The robot routing algorithm and the cycle counting methods are tested in a case study of Arox. Arox is a logistic software developer and makes custom software for inventory, transport and financial management systems. Furthermore, Arox is developing an autonomous cycle counting system with drones. For this purpose, the robot cycle counting algorithm is tested in a simulation with the setup of Arox's hardware and a warehouse layout similar to clients of Arox.

Arox currently uses one kind of drone. In an interview with the CEO of Arox, the following system characteristics became clear. The drones used have a battery capacity of 20 minutes of flight time and due to safety reasons the speed of the drones are capped at walking speed. An empty drone has a recharging time of 2 hours. Furthermore, the warehouses of clients range from 20.000 – 120.000 square meters. Currently, the human based cycle counting systems of clients use 2-3 employees dedicated to cycle counting. Moreover, the counting process requires two employees and a forklift truck. In addition, the majority of companies use ABC cycle counting in combination with opportunity-based cycle counting. Arox was not able to provide historical cycle counting data, nor product characteristics. Therefore, random data is generated for the simulations, as explained in the next chapter.

The simulated warehouse, as seen in Figure 6, is 195 meters long and has 73 rows of product shelfs, which have a length of 52 meters. The shelfs are 0.8 meters in width and the aisles are 3.4 meter apart from each other. Each of the shelfs has 3 levels and it is assumed that each level contains 15 products, equally spaced. Furthermore, it is assumed that the products can be scanned on one point 0.5 meters from the product. Therefore, each of the product shelfs contains 45 products, making the maximum capacity 3285 products. Lastly, the aisles can only be entered and existed from one side, because the shelfs are situated against the back of the warehouse.

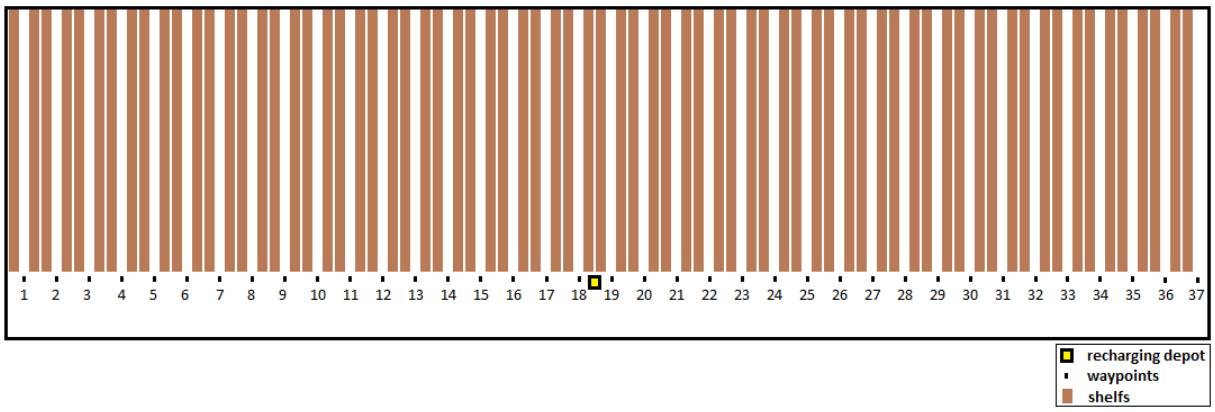


Figure 6: Layout of simulated warehouse.

4.1 Construction Monte Carlo simulation Arox

The simulations are performed using MATLAB version 9.5.0.944444 (R2018b-educational), on a Rijksuniversiteit Groningen server, with an Intel(r) Xeon(r) CPU e5-2690 v4 @ 2.60GHz processor. Furthermore, the simulation is constructed using the Monte Carlo method. The Monte Carlo methods are a group of methods, which use random samples for solving mathematical problems. Rather than computing the expected value using the, most often complex, definitions; the behaviour of randomly generated variables is observed and the expected value estimated based on these observations [39]. In this case, it is used to determine how the algorithm performs in a practical case. Moreover, the warehouse has many possible configurations of products and possible errors. Secondly, it is used to test which of the cycle counting method is best suited for the robot cycle counting system.

The fixed parameters of the system are as follows:

- The maximum speed of the drones is set at 6 km/h
- The maximum flying duration of the drones 20 minutes
- The maximum cycle counting duration is 4 hours (maximum schedule length)
- The time it takes the drone to scan a product in 3 seconds
- The first 85% of the battery capacity takes 1 hour and the last 15% takes 1 hour to recharge

The maximum speed and flying duration are assumed based on the interview with the Arox CEO. The maximum scheduling duration is set at 4 hours, because for the routing algorithm to take large amount of products into consideration the processing time for each simulation will be too large to perform a reasonable amount of simulations. The results of the simulations will be extrapolated for longer cycle counting periods. Furthermore, it is assumed that the drone needs to position itself to be able to scan the product properly, or finding the barcode. Lastly, the robot routing algorithm forces the robots to recharge after every route. Therefore, the recharging time is assigned based on the battery depletion. However, the last percentages of the battery capacity takes considerably longer. The total recharging time is 2 hours, it is assumed the last 15% takes an hour., as the last percentage of capacity take longer in most batteries [35].

A number of simulations are randomly created and each cycle counting method will be tested on the same instances. The amount of products and the percentage of products in failure are chosen as follows:

- The amount of products present in the warehouse is uniformly drawn from **n2**=[0.2n n]
- The amount of failures is uniformly draw from **FL=n2**[0.1 0.4]

As previously mentioned, no product or cycle counting data was available. Therefore, the missing system parameter are randomly generated. To test the logistic regression model within this simulation, product characteristic data needs to be created with some correlation to the failure of products. In this simulation, it is assumed that the products stored on the highest shelf have a higher probability of failure, i.e. being misplaced, than the products on lower level. Therefore, firstly the amount of failures within each simulation is randomly drawn from a uniform distribution. Thereafter, products, i.e. the position within the warehouse, are assigned to the failures. A failure has 50% chance of being assigned a product on the highest level, 25% chance for each middle and lowest level products.

A random product that resides in the respective level is assigned and removed from being chosen again. If **FL** products are assigned to failures, "non-failure" products are assigned randomly from the products that are left to be chosen until **n2** products. In addition, two more random variables are assigned to the chosen products in the simulation. These variables can represent any product characteristic, for example transaction volume, quantity or product weight. In this experiment, variable 1 will be assigned the transaction volume

and variable 2 the products weight. Therefore, the logistic regression model will have 3 variables to base a prediction model on. However, these other two random variables, transaction volume and product weight, need to have some correlation to the failure products. The transaction volume has a value randomly drawn from two different normal distributions depended on failure $400 + a200$ or no failure $300 + a5$, where a is a random number draw from a standard normal distribution. The standard deviation is chosen to be high in case of failure, to avoid a perfect prediction. The product weight is described as $200 + a10$, where a is randomly drawn from a standard normal distribution with correlation factor of 0.6 to the transaction volume. In addition, the price of every product is randomly drawn from a normal distribution with a mean of 700 and a standard deviation of 200. Following this principle, 200 simulations are generated, from which 100 are randomly removed for training purposes.

As described in Chapter 3.1**,** the algorithm requires a distance matrix from all products to all other products. However, the drones cannot pass through the shelfs. For this purpose, a rule is implemented that the drones need to travel to a point in front of the aisle, before traveling to the product within the aisle that needs to be scanned. For example, if a drone has to scan a product in aisle 37 and is currently positioned in aisle 1, it has to travel to a given point in front of aisle 1 and then to point a given point in front of aisle 37, before entering aisle 37. The distance from the recharging depot to the products works with the same principal.

Since every product has a specific x, y, z, location the traveling time between products within the same aisle can be calculated conform formula (24). While, the traveling time between products in different aisles is derived as the Euclidian distance to the point in front of the aisle plus the distance between the way-points in front of the aisles, plus the Euclidian distance from the second aisle way-point to the new product. In the simulation, the recharging depot is placed in front of the aisles, near the 18th aisle. In addition, the cases will have two drones cycle counting.

4.2 Cycle counting method implementation

The cycle counting methods are implemented as described in Chapter 3.2. The random sample method is implemented by filling the sample with random products drawn from the available products in the respective simulation. For the ABC method, the products are ranked based on product price and the sample is filled with the highest value products first. Furthermore, the location-based method randomly choses an aisle and adds the products present in the aisle to the cycle counting sample. This is repeated until the required sample size is met. Lastly, the logistic regression model, described in Chapter 3.2**,** is trained on the 100 training simulations, based on the product characteristics, i.e. the transaction volume, product weight and the level in the shelfs, representing historical cycle counting data. A third of training data is used to train the model the and rest is used for testing the prediction [1]. The maximum amount of iteration was set at 20 and the convergence tolerance on beta at 10^{-9}. The graph of the cost function for each iteration is shown below. The cost function is described as $-\left(\frac{1}{m}\right) L(\beta)$.

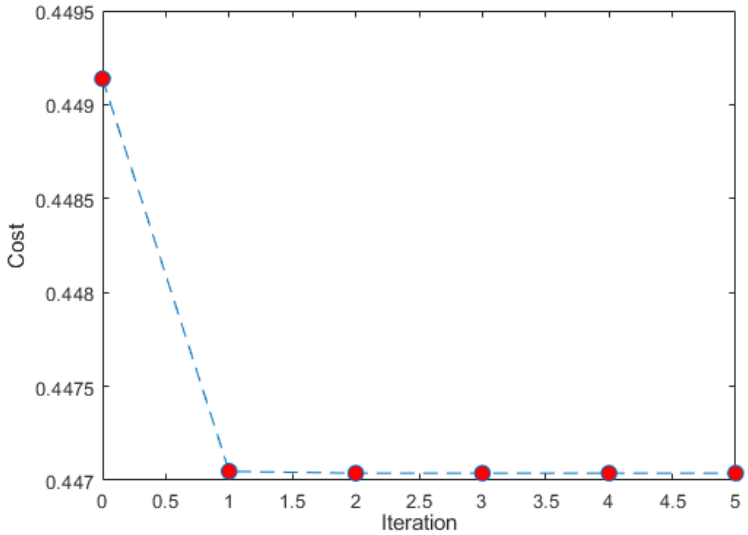


Figure 7: convergence of the cost function in the Newton-Raphson method for calculating the maximum likelihood estimates of the logistic regression.

It can be seen that the algorithm converged extremely fast. The algorithm was stopped based on the convergence criteria. The final beta coefficients are shown in the table below.

Table 2: Logistic regression product parameter coefficients.

Product parameter	Coefficient value
β_0	1.158
Transaction volume	0.011
Product weight	-0.033
Level in rack	0.205

The tested accuracy, on the other two thirds of the training data, was 86.36%. Furthermore, the trained model will be used to classify the products in the simulation and the cycle counting sample will be filled with the highest probability of failure products first.

The routing algorithm needs to be given a cycle counting sample size, from which the methods described above, can chose products to scan. Therefore, for each simulation the sample size is chosen sufficiently large, that the algorithm will not be able to obtain a feasible solution. After the algorithm obtains the least infeasible solution, the schedule and the routes within are "cut-off" at the maximum scheduling time. The reason being that a maximization of erroneous products found is sought after. If the sample size is chosen such that the algorithm will obtain a feasible solution, it is impossible to know if the cycle counting method could have scanned more erroneous products. Moreover, the algorithm assigns recharging time after each route. Therefore, the schedule could be infeasible due to the recharging time of the last route, which is incorrect since the robots are allowed to recharge after the cycle counting schedule ends.

4.3 Parameter tuning

The robot routing algorithm has some parameters, as explained in Chapter 3.1**,** that need to be tuned to get the best combination of least infeasible, least cost (total scheduling time) and running time of the algorithm. Obtaining the right parameters is done by applying the random sample cycle counting method in combination with the robot routing algorithm on a portion of the 100 training instances. The random sample method is chosen, as it is an unbiased method. Each experiment was done on 10 simulation instances, with a sample size of 300 products and the same fixed variables of the simulation described previously. Furthermore, the simulations are run with the use of pseudo random numbers. Therefore, each experiments runs exactly the same instances. In experiment 1-6 one of the parameters is changed to see the effect on the solution and running time of the algorithm, seen in the table below. In addition, experiment 7-12 are performed with combinations of the six initial experiments, to find the best configuration.

Table 3: Robot routing algorithm parameters.

	Exp1	**Exp2**	**Exp3**	**Exp4**	**Exp5**	**Exp6**	**Exp7**	**Exp8**	**Exp9**	**Exp10**	**Exp11**	**Exp12**
M	100	100	100	100	**200**	100	**200**	**200**	**200**	**200**	**200**	**200**
AMPiter	10	**20**	10	10	10	10	10	10	10	**20**	**15**	10
In	10	10	**20**	10	10	10	10	**20**	10	10	10	10
MaxIt	10	10	10	**20**	10	10	10	10	10	10	**15**	**20**
thetha min	1	1	1	1	1	1	1	1	1	1	1	1
thetha max	5	5	5	5	5	5	5	5	5	5	5	5
p	5	5	5	5	5	**7**	**7**	**7**	**9**	**7**	**7**	5
Products in sample	300	300	300	300	300	300	300	300	300	300	300	300
Cost	484,38	**483,27**	489,98	**478,51**	484,38	**481,86**	**481,86**	488,00	**483,49**	**481,40**	**480,26**	**478,51**
Overtime	10,15	**7,87**	12,50	**3,53**	10,15	**7,81**	**7,81**	10,86	**8,89**	**4,01**	**4,92**	**3,53**
Running time	97,32	171,56	116,50	191,03	97,50	138,58	137,93	159,36	177,62	241,13	282,44	190,08

The first experiment is baseline for the rest of the experiments, with a maximum memory capacity **M** of 100, 10 AMP iterations **AMPiter**, 10 initial routes **In** in the solution, 10 Tabu Search iterations **MaxIt**, a Tabu length drawn from 1 to 5 and a neighbourhood **p** of 5 products. The cost, overtime and running time are mean values from 10 simulation cases. Furthermore, the cost and overtimes are displayed in minutes and the running time in seconds. It can be seen form the table above that all other experiment are equal to or improve the total cost of the first experiment, except case 3 and 8. Moreover, the overtimes are increased as well. Here the initial solution **In** are increased to 20 routes. This can be explained by the fact that the AMP procedure has less chance of picking "good" solutions at the start of the algorithm. Therefore, less optimal solutions are made and in return inserted into the memory.

Increasing **AMPiter**, **MaxIt** and **p** will result in better overtimes and costs of the solution. However, it increases the running times drastically. Increasing the Tabu iterations **MaxIt** improves the solution the most, but also doubles the running times. This is not surprising considering each "action" within the products neighbourhood needs to be considered twice as often. From case 5 can be seen that the increase in memory capacity **M**, does not improve the solution nor the running times on its own. However, in combination with an increased neighbourhood **p** and initial solution **In**, experiment 7 and 8 respectively, it decreases the running time of the algorithm. Due to the fact that increasing the memory capacity gives a higher chance that the solution can be constructed from existing routes in the memory. Therefore, reducing the time the

SWEEP algorithm has to be run on for the products that are not routed. Surprisingly, increasing the neighbourhood to 9 products, case 9, reduces the quality of the solution. This is unexpected since a larger neighbourhood should give more possible improvements. Furthermore, combining a larger **M**, **p**, **AMPiter** and **MaxIt**, did not increase the solution more than case 4, but drastically increased the running time. Therefore, the experiment 12 was run to see the effect of the increased **M** in combination with **MaxIt**. As can be seen from the table above, the running time decreased a little. However, as explained in the previous chapter, the actual simulation will be run with more than 300 products to get the maximum amount of products scanned within the maximum scheduling time. Therefore, it is expected that the larger **M**, will contribute more to the running time with larger sample sizes. Moreover, it did not reduce the improvement of the solution from increasing **MaxIt**. In addition, increasing **MaxIt** to 20 improved the solution the most. However, to reduce the running times on the larger sample sizes; the main experiment will be run with 15 Tabu Search iteration **MaxIt** and the other parameter conform experiment 12.

5 Results & analysis Arox case study

In this chapter the results, of the simulated Arox case study, are show and analysed. The simulations where performed, as described in the previous chapter. The results of 100 simulations for each cycle counting method is shown in Table 4**.** All values are the mean values over all the simulations. The sample size is the amount of products that are taken into account for the robot routing algorithm. The mean scanned products are the actual scanned products, after the schedule was cut. Furthermore, the error capture percentage is divined as the erroneous products that are scanned by the method divided by the total amount of erroneous products in the simulation.

Table 4: Results Monte Carlo simulations for random sample, ABC, Location-based and logistic regression-based cycle counting methods.

	Random	ABC	Logistic	Location
Mean sample size	600	600	600	914.1
Mean total scanned products	500.1	497.1	503.6	808.3
Mean error capture percentage	29.80%	29.80%	55.20%	45.14%
Mean running time	417.2	419.1	412.0	758.7

For all methods, except the location-based method, the scanning size was set at 600. The scanning size of the location-based method was set at 900. However, the rule for filling the sample dictates, to add aisles of products while 900 products are not in the sample. Therefore, in some cases the sample size was larger. This only resulted in longer running times, as the mean of total scanned products is far lower than the amount of products in the sample. It can be seen that the location-based method was able to scan an extremely larger amount of products, than the rest of the methods. This resulted in a higher error detection percentage than the random and ABC method. However, the logistic regression method far exceeds the error detection percentage of the other methods. Moreover, the Logistic regression method has a more stable performance than the other methods, as 50% of all simulations had an error capture percentage between 50% and 60%, seen from Figure 8.

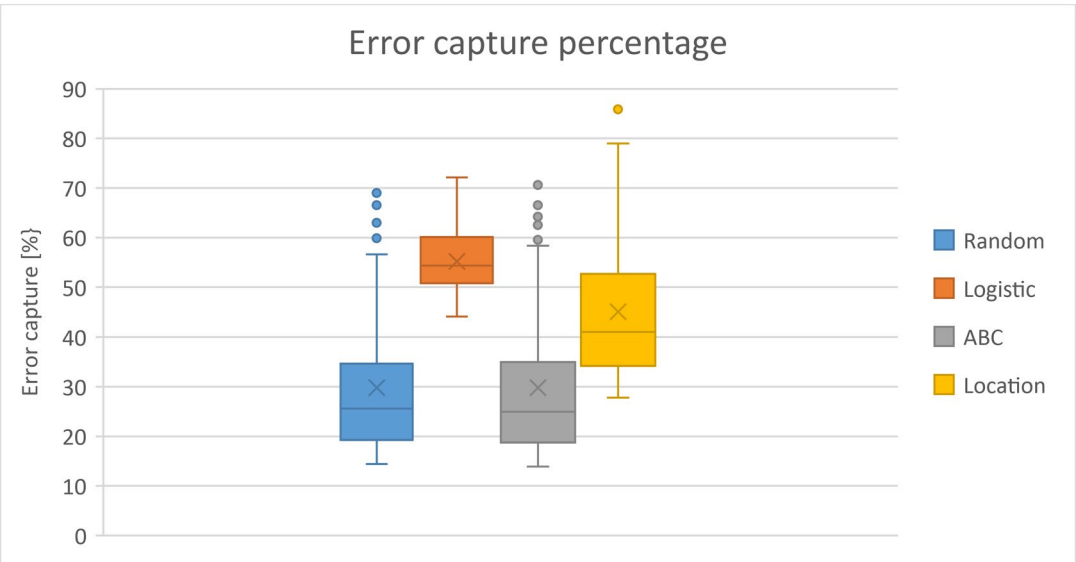


Figure 8: Box plot of the error capture percentage found in the 100 Monte Carlo Simulations performed for random sample, logistic regression-based, ABC and location-based cycle counting. Each plot is divided into four parts: the upper wick, the two halves of the box and the lower wick, which each represent 25% of the data. The x represents the mean of the data and the round points outside of the boxplot represent the outliers.

The location-based method does not have a stable performance, as 50% of the simulations performed had an error capture percentage of 34-52% and 25% of the simulations 52-79%. In contrast, the maximum schedule in the simulations is 4 hours. If the cycle counting system would have 8 hours each day a full warehouse of products can be scanned in $\frac{1}{2}\left(\frac{3285}{808.3}\right) = 2.03$ days. Therefore, the location-based method will be the best method when: robot capacity is high, there are relatively few products to scan, the warehouse is small and the throughput of products is low. However, the logistic regression method will be best suited in systems where few products can be scanned each day and product throughput is high.

Furthermore, from the values of the logistic regression coefficients, seen in Table 2, it can be concluded that the, self-assigned, transaction volume and the shelf level are the best indicators for product misplacement. Both the coefficients are relatively larger than the product weight coefficient. This was to be expected, because the transaction volume and shelf level were created with a strong correlation to product failure.

In addition, the mean price of the erroneous products scanned was only slightly higher for the ABC method. Probably, due to the fact that the product price was randomly drawn from a normal distribution. Furthermore, the prices were assigned without a correlation to erroneous products. Therefore, the mean error capture percentage of the ABC method is similar to the random cycle counting method. Moreover, over the 100 simulations the ABC method captured roughly 13 more products with a price above 1000, compared to the other methods. Therefore, in this case the error price of those highly priced products need to be extremely more expensive than the other products; for the ABC method to be better than the other methods. In contrast, the price of products could be differently distributed in real-life.

5.1 Validation Monte Carlo Simulations

In [40] two method are described to validate the amount of Monte Carlo Simulations needed. The first, is a widely used method, where the performance variable in question is assumed to be normally distributed. Therefore, the amount of required Monte Carlo simulations is calculated for a chosen confidence interval and error percentage. The second method dictates that the proper amount of Monte Carlo simulations is reached when the mean value of the performance variable question converges. Therefore, plotting the mean performance value against the number of Monte Carlo Simulations performed. Both yielded similar results when tested. However, the second method does not assume a distribution function of the performance variable. Therefore, the second method is used in this thesis to validate the amount of Monte Carlo Simulations needed. Shown in Figure 9, is the mean error capture percentage set out against the number of Monte Carlo Simulations performed.

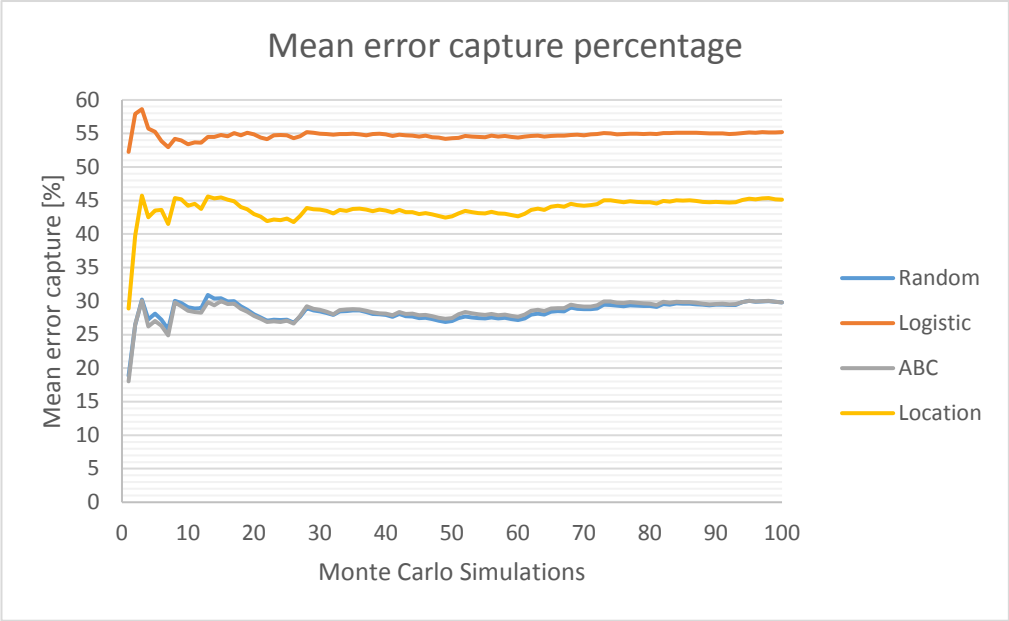


Figure 9: The mean error capture percentage against the number on Monte Carlo Simulations performed, for the four cycle counting methods.

The figure shows that the error capture percentage converges, for all the cycle counting methods, after 75 iterations. Therefore, the 100 Monte Carlo simulations performed are enough to conclude the best cycle counting method based on the error capture percentage.

5.2 Analysis robot routing algorithm

Lastly, it can be seen, from Table 4, that the robot routing algorithm performed really well, on large amounts of products, in terms of running time. It takes on average roughly 7 minutes for instances with a sample size of 600 products and 12.6 minutes for 914 products. Furthermore, the algorithm found a feasible solution in all 400 simulations. The performance of the algorithm in terms of solution quality is hard to measure, as there is no exact solution available and no other algorithm was used in this specific case study. Therefore, the several simulation solutions, of the Arox case study, are analysed graphically. A simulation of the location-based cycle counting method is shown in Figure 10 and the a simulation of the logistic regression-based method in Figure 11, where the recharging depot is denoted as a yellow square, the products as the circles and the routes as lines with each a different colour. It can be seen that the SWEEP method is not optimal in this case, because the aisles on the left are covered in different routes.

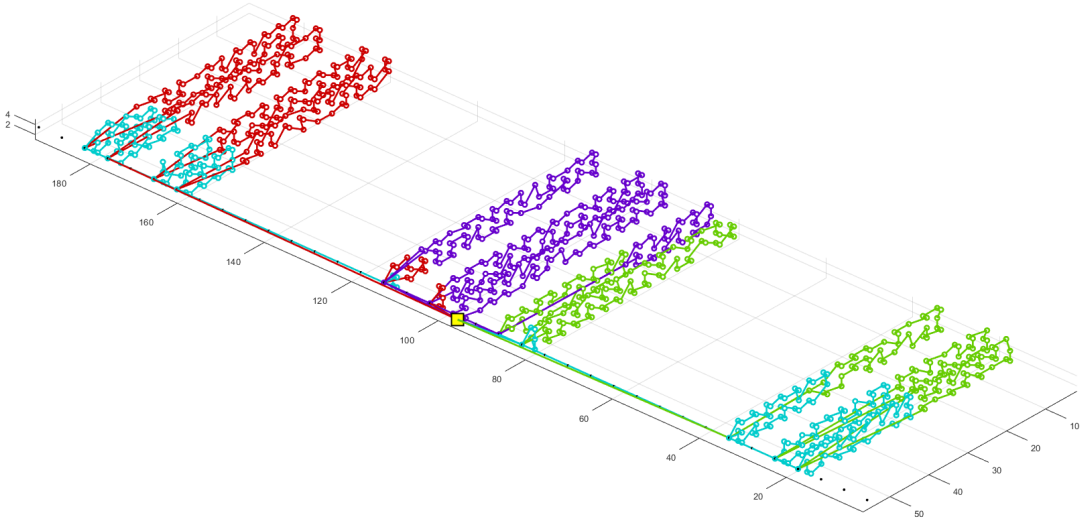


Figure 10: Simulation of the location-based cycle counting method.

The SWEEP algorithm creates these routes, because the products are inserted into the route based on the polar angle on the x and y plane. Therefore, the drones have to fly in and out of the aisles more often, which costs time. This is shown clearly in the last two aisles of the right for the green route in Figure 10 and the green route in Figure 11. Here the drone has to scan few products at the end of the aisle and then fly back to scan products at the end of the next aisle. Therefore, the SWEEP method is less optimal in cases that include large obstacles. However, if the aisles had been accessible from both sides the solutions would be more optimal. In addition, the TABU search procedure was not able to fully correct these not optimal routes, by optimizing between routes, because the neighbourhood of products was set at 5 to balance the running time and solution quality.

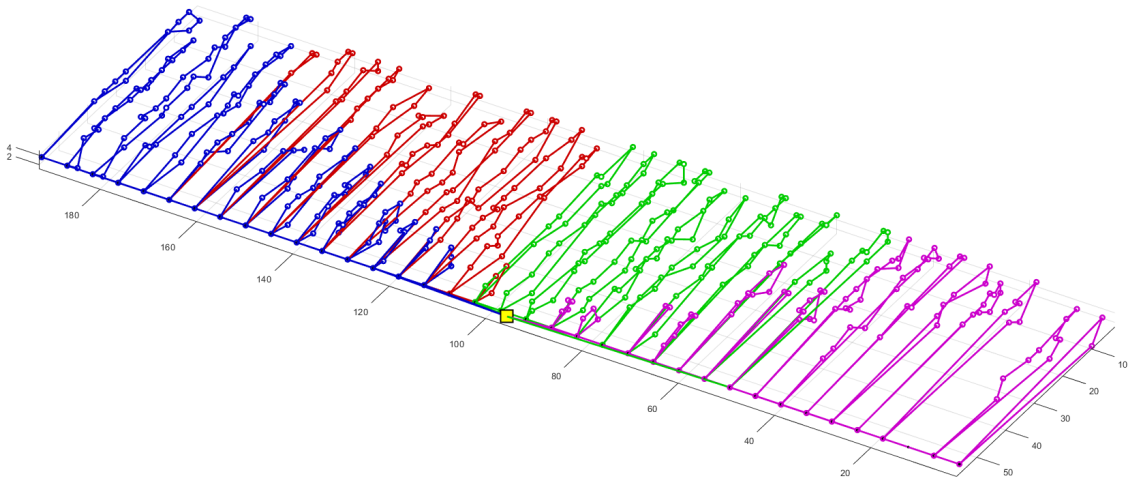


Figure 11: Simulation of the logistic regression-based cycle counting method.

However, the routes within the aisles, shown in Figure 12, seem efficiently made. The 3D figure shows the route of one aisle of products that need to be scanned in one of the simulations. Therefore, not all products in the aisle need to be scanned. It can be seen from the figure that the drone would move through the aisle quit efficiently. Therefore, the 2-optimal heuristic, for optimizing the single traveling salesman tours, and the TABU search procedure, for optimizing between routes, performed well.

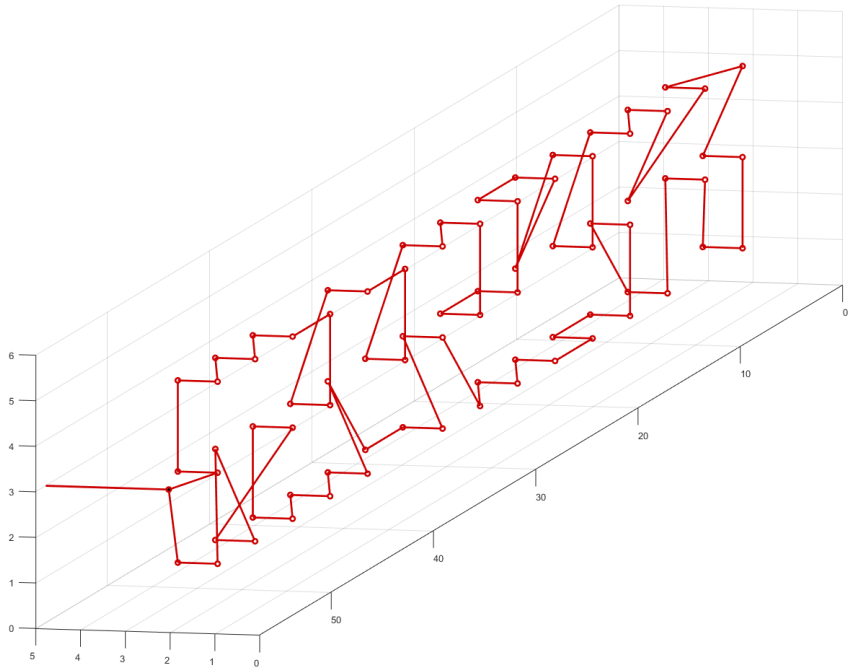


Figure 12: Individual route of one aisle in the simulation

6 Discussion and future research

The location-based cycle counting method would be the best choice in the chosen simulation setup, because scanning all the products is possible within two days. Only an extremely low percentage of products will be missed in the two-day scanning period, due to throughput of products. In addition, the method does not need any prior information about the system. Therefore, the location-based method can be setup easily. However, all methods tested in this system would still need a system for the robots to know its location, because the products are scanned individually. Achieving a real-time location system within a warehouse is still a challenge, but can be done [41]. Furthermore, the logistic regression-based model would be a better choice in large warehouses with a low cycle counting capacity. This result corresponds with the result in [1]. In addition, the logistic regression method only needs historical cycle counting data, but does not need any prior knowledge. Moreover, it can be used to take preventive actions based on the relative size of coefficients assigned to the product features [1]**.** Although, random sample cycle counting performed worse than the other methods; it should not be dismissed as it can be leveraged to get an accurate IRA, based on a random sample instead off a biased sample. Moreover, if the periodic random sample cycle counting reveals poor performance of IRA, the method of [5] can pose a great tool for analysing what the sample size of the cycle counting system should be to maintain a certain level of IRA. Lastly, it is interesting to see that ABC-cycle counting, as the most used cycle counting method based on the Arox interview, performs the worst in this robot cycle counting system. Moreover, it has the least amount of added value to the cycle counting system.

The contribution made to the AMP and Tabu search hybrid metaheuristic algorithm [28] will give the algorithm a wider range of use. The additions made are: 3D case possibility, capability of scheduling recharging, consideration of fixed cost (scanning time), capability of heterogeneous fleet of vehicles. The recharging time is even considered in the improvement of the solution, because the routes are assigned within the Tabu search procedure. Moreover, the robot routing algorithm performs well on the large case study simulations, but it takes relatively long in comparison with [28]. However, the original algorithm is only tested until 400 customers in 2D cases. If longer scheduling times are implemented, more than the 4 hours in the simulations, large amounts of products need to be taken into account, which will take the algorithm extremely longer running times. This does not have to be a problem if the robot cycle counting system is only used outside of operating hours of the warehouse. Therefore, the algorithm will run during operating hours and the schedule will be ready for the robots after operating hours, but this is not ideal.

It is recognized that the robot routing algorithm can be improved. The 2-opt heuristic for optimizing the single traveling salesman tour does not get the most optimal solutions, but is relatively fast. In addition, the 2-opt heuristic is also used for the post-optimization, if the route in the Tabu Search procedure improves the best known solution, whereas the original algorithm uses parts of the GENIUS algorithm for post optimization. However, the Tabu search algorithm in [28] only tries to insert a product before the p closest neighbours. Whereas, the robot routing algorithm considers before and after insertion. The best position towards that neighbour is already chosen and the route is made 2-optimal again afterwards. Therefore, the improvement of implementing another post-optimization algorithm could be minimal. Secondly, the SWEEP could be improved on running time, by adapting the procedure in such a way that the 2-opt heuristic is not needed as much. Now the 2-opt heuristic is needed in every iteration and 5 times in the iterations that the tour violates the maximum distance constraint. Furthermore, the SWEEP algorithm could be improved in cases with long aisles and few aisle entry points, by using another rule for insertion than based on the polar angle. Therefore, future research can be done on improving the SWEEP algorithm.

The constructed robot routing algorithm is generic, such that a heterogeneous fleet of robots can be used. However, if there are driving robots in the system that cannot scan the top halve of products in the warehouse shelfs, then a HVRPFD-MT is not enough. In this case, site dependency should be added to the algorithm, such as the TS-ABB algorithm [32] is capable of. Future research can be done to add site dependency to the robot routing algorithm. Furthermore, the possibility of using multiple recharging depots could be great addition the algorithm as well, as done in [23].

In addition, the distance between products was calculated using "checkpoint" in front of the aisle, where the robots need to pass through, before traveling to the next aisle. In the case study of Arox, this method was feasible, as the warehouse layout was not complex. However, if the warehouses is larger and the layout more complex, this method might not be feasible. Therefore, future research can be done to add an algorithm that can calculate the distance with obstacles for every product in the warehouse to every other product, such as the algorithm in [42].

Lastly, in a small sense it is also achieved, with this routing algorithm, to assign priority. It is not a hard requirement in the algorithm to scan a product before another is scanned, nor would this be necessary in this system as is. However, the "cut-off" method in the simulation, of assigning more products to be scheduled than is possible within the timeframe and stopping the schedules at the maximum time, could be improved by not choosing the smallest route. Rather, the products in the lower classes, i.e. the lower priced in the ABC method or lower probability of failure in the supervised learning method, should be eliminated from the schedule. Furthermore, forcing the robots to recharge after every route is a good rule for the first routes in the schedule. However, in combination with the "cut-off" rule this will give less optimal solutions, because the robots could stop recharging until a certain percentage and scan more products until the end of the scheduling time. An improvement can be made, by taking this possibility into consideration during the Tabu Search procedure. In contrast, maximizing the total amount of products would be more ideal in this situation. However, the algorithm will take extremely long, because all the products have to be taken into account. Therefore, the running time of the 2-opt and Tabu search algorithms will dramatically increase. Therefore, future research can be done to add a procedure that iteratively increase the products.

7 Conclusion

Concluding this thesis the main questions will be answered separately. Firstly. "*How can an algorithm be constructed that schedules robots for the purpose of cycle counting?*". The autonomous robot cycle counting system can be modelled as a Heterogeneous Vehicle Routing Problem with Fixed costs, Dependant costs and Multiple-trips, because different types of robots need to individually scan the products for cycle counting, need to recharge after routes and need to perform multiple cycle counting trips within a day. This type of product can be solved using an Adaptive Memory Programming and Taby Search hybrid metaheuristic algorithm, where routes are created using a SWEEP algorithm, individual routes are improved by a 2-optimal heuristic, the routes and recharging times are assigned to the schedules of the robots using a Best Fit Decreasing bin-packing algorithm, the routes is improved by a Tabu Search procedure and the solution is improved using Adaptive memory programming. Therefore, creating a generic scheduling algorithm for the autonomous robot cycle counting system.

Secondly, *"How can the robot routing algorithm be validated and how does it perform in the case study of Arox?"*. The robot routing algorithm successfully validated in the case study of Arox, as it found a feasible solution in all 400 simulations performed. Furthermore, the algorithm performed really well, on large amounts of products, in terms of running time. It takes on average roughly 7 minutes for instances with a sample size of 600 products and 12.6 minutes for 914 products. However, a graphical analysis showed that the SWEEP algorithm is not an optimal choice for creating routes in cases with long aisles that pose as obstacles for the robots, and few possibilities for entering the aisles.

Lastly, *"What is the best combination of cycle counting method combined with autonomous robot cycle counting in terms of inventory record accuracy?"*. There are four cycle counting methods compatible with the robot cycle counting system: random sample, ABC, location-based and supervised learning-based cycle counting. A good model to use is logistic regression, for the supervised learning-based method, in combination with a Newton-Raphson algorithm for finding the maximum likelihood estimates. The location-based cycle counting method would be the best choice in the case study of Arox, because scanning all the products is possible within three days. Only an extremely low percentage of products will be missed in the two-day scanning period, due to throughput of products. In addition, the method does not need any prior information about the system. Therefore, the location-based method can be setup easily. Furthermore, the logistic regression-based model would be a better choice in large warehouses with a low cycle counting capacity. In addition, the logistic regression method only needs historical cycle counting data, but does not need any prior knowledge. Moreover, it can be used to take preventive actions based on the relative size of coefficients assigned to the product features. To get the most accurate IRA, the random sample cycle counting method should be used, because it is the only method that works with an unbiased cycle counting sample.

8 Bibliography

[1] L. Wijffels, V. Giannikas, P. Woodall, D. McFarlane, and W. Lu, 'An enhanced cycle counting approach utilising historical inventory data', *IFAC-PapersOnLine*, vol. 49, no. 12, pp. 1347–1352, 2016.

[2] M. D. Rossetti, T. Collins, and R. Kurgund, 'Inventory Cycle Counting: A Review', *Proc. 2001 Ind. Eng. Res. Conf.*, pp. 457–463, 2001.

[3] T. Sandle, 'Using drones in large warehouses to prevent inventory mismatches', 2017. [Online]. Available: http://www.digitaljournal.com/tech-and-science/technology/using-drones-in-large-warehouses-to-prevent-inventory-mismatches/article/504812. [Accessed: 23-Dec-2018].

[4] M. D. Rossetti, T. Collins, and R. Kurgund, 'Inventory Cycle Counting', *Proc. 2001 Ind. Eng. Res. Conf.*, pp. 457–463, 2001.

[5] J. M. Wilson, 'Quality control methods in cycle counting for record accuracy management', *Int. J. Oper. Prod. Manag.*, vol. 15, no. 7, pp. 27–39, 1995.

[6] Pinc, 'Warehouse Drone for Inventory Management | PINC', 2018. [Online]. Available: https://www.pinc.com/warehouse-drone-inventory-management. [Accessed: 22-Dec-2018].

[7] Swedberg Claire, 'PINC Brings RFID, Optic Sensor Drones Indoors With Fuel Cells - 2017-05-12 - Page 1 - RFID Journal', 2017. [Online]. Available: https://www.rfidjournal.com/articles/view?16008. [Accessed: 23-Dec-2018].

[8] Dronescan, 'Inventory — DroneScan', 2018. [Online]. Available: http://www.dronescan.co/inventory/. [Accessed: 22-Dec-2018].

[9] HARDISGROUP, 'Eyesee: the drone allowing to automate inventory in warehouses | Hardis', 2018. [Online]. Available: https://www.hardis-group.com/en/our-activities/logistics-solutions/eyesee-drone-allowing-automate-inventory-warehouses. [Accessed: 14-Jan-2019].

[10] SCDigest, 'New Systems Uses Robots and RFID Readers to Perform Continuous Cycle Counting', 2017. [Online]. Available: http://www.scdigest.com/ontarget/17-08-23-1.php?cid=12908. [Accessed: 14-Jan-2019].

[11] I. Maw, 'Automate Inventory Management with Mobile Robots > ENGINEERING.com', 2017. [Online]. Available: https://www.engineering.com/AdvancedManufacturing/ArticleID/15529/Automate-Inventory-Management-with-Mobile-Robots.aspx. [Accessed: 12-Dec-2018].

[12] J. Gu, M. Goetschalckx, and L. F. McGinnis, 'Research on warehouse design and performance evaluation: A comprehensive review', *Eur. J. Oper. Res.*, vol. 203, no. 3, pp. 539–549, 2010.

[13] A. G. Kök and K. H. Shang, 'Evaluation of cycle-count policies for supply chains with inventory inaccuracy and implications on RFID investments', *Eur. J. Oper. Res.*, vol. 237, no. 1, pp. 91–105, 2014.

[14] Q. Lee, *Cycle counting & Inventory Accuracy*. 2016.

[15] R. B. Brooks and L. W. Wilson, *Inventory Record Accuracy, Unleashing the power of cycle counting*, 2nd ed. John Wiley & Sons, Inc., 2007.

[16] A. RAMAN, N. DEHORATIUS, and Z. TON, 'D3. The Achilles' Heel of Supply Chain Management', *Harv. Bus. Rev.*, vol. 79, no. 0, p. 25–+, 2001.

[17] R. Tersine, *Principles of Inventory and Materials Management*, Fourth. Prentice-hall International, Inc., 1994.

[18] A. Sarac, N. Absi, and S. Dauzere-Pérès, 'A literature review on the impact of RFID technologies on supply chain management', *Int. J. Prod. Econ.*, vol. 128, no. 1, pp. 77–95, 2010.

[19] P. Gy, 'Sampling of discrete materials - A new introduction to the theory of sampling: I. Qualitative approach', *Chemom. Intell. Lab. Syst.*, vol. 74, no. 1, pp. 7–24, 2004.

[20] D. C. Montgomery and G. Runger, *Applied Statistics and Probability for Engineers*, 6th ed. John Wiley Singapore Pte. Ltd., 2014.

[21] B. Golden, S. Raghavan, and E. Wasil, 'The vehicle routing problem: Latest advances and new challenges', *Oper. Res. Comput. Sci. Interfaces Ser.*, 2008.

[22] J. K. Lenstra, 'Complexity of Vehicle Routing and Scheduling Problems', vol. 11, pp. 221–227, 1981.

[23] B. Crevier, J. F. Cordeau, and G. Laporte, 'The multi-depot vehicle routing problem with inter-depot routes', *Eur. J. Oper. Res.*, vol. 176, no. 2, pp. 756–773, 2007.

[24] M. Gendreau, A. Hertz, and G. Laporte, 'New Insertion and Postoptimization Procedures for the Traveling Salesman Problem', *Oper. Res.*, vol. 40, no. 6, pp. 1086–1094, 1992.

[25] C. Blum and A. Roli, 'Metaheuristics in combinatorial optimization: overview and conceptual comparison', *ACM Comput. Surv.*, 2003.

[26] E. D. Taillard and L. M. Gambardella, 'Adaptive Memory Programming□: A Uni ed View of Metaheuristics 1 Introduction', vol. 135, pp. 1–22, 2001.

[27] a. Sen and K. Bülbül, 'A Survey On Multi Trip Vehicle Routing Problem', *VI. Int. Logist. Supply Chain Congr. 2008*, no. July, 2008.

[28] A. Olivera and O. Viera, '2007-Adaptive memory programming for the vehicle routing problem with multiple trips.pdf', no. October 2004.

[29] B. E. Gillett and L. R. Miller, 'A Heuristic Algorithm for the Vehicle-Dispatch Problem', *Oper. Res.*, vol. 22, no. 2, pp. 340–349, 1974.

[30] M. Gendreau, A. Hertz, and G. Laporte, 'Tabu Search Heuristic for the Vehicle Routing Problem', *Management*, vol. 40, no. 10, pp. 1276–1290, 1994.

[31] P. Toth and S. Martello, 'knapsack problems algorithms and computer implementations', *Discret. Math. Optim. Wiley*, 1990.

[32] F. Alonso, M. J. Alvarez, and J. E. Beasley, 'A tabu search algorithm for the periodic vehicle routing problem with multiple vehicle trips and accessibility restrictions', *J. Oper. Res. Soc.*, vol. 59, no. 7, pp. 963–976, 2008.

[33] J. F. Cordeau, M. Gendreau, and G. Laporte, 'A tabu search heuristic for periodic and multi-depot vehicle routing problems', *Networks*, 1997.

[34] M. Englert and H. Röglin, 'Worst case and probabilistic analysis of the 2-opt algorithm for the TSP: extended abstract', *Proc. eighteenth*, pp. 1295–1304, 2007.

[35] 'batteryuniversity.com'. [Online]. Available: https://batteryuniversity.com/learn/article/charging_lithium_ion_batteries.

[36] R. Tibshirani, 'Logistic regression and Newton's method', 2012.

[37] E. Erhardt, 'Statistical Computing chapter 11 Logistic regression and newton raphson's method', 2015.

[38] A. Schworer and P. Hovey, 'Newton-Raphson Versus Fisher Scoring Algorithms in Calculating Maximum Likelihood Estimatates', *Electron. Proc. Undergrad. Math. Day 2004*, no. 1, pp. 1–11, 2004.

[39] M. Amelin, *Monte Carlo Simulation in Engineering*. 2013.

[40] E. Bukaçi, T. Korini, E. Periku, S. Allkja, and P. Sheperi, 'Number of iterations needed in Monte Carlo Simulation using reliability analysis for tunnel supports', *J. Eng. Res. Appl. www.ijera.com*, vol. 6, no. 6, pp. 60–64, 2016.

[41] C. Kirsch and C. Röhrig, 'Global localization and position tracking of an Automated Guided Vehicle', *IFAC Proc. Vol.*, vol. 44, no. 1 PART 1, pp. 14036–14041, 2011.

[42] S.-N. Chow, J. Lu, and H.-M. Zhou, 'THE SHORTEST PATH AMID 3-D POLYHEDRAL OBSTACLES', *SIAM Multiscale Model. Simul.*, 2013.

Appendix A: Flight and recharging times commercial drones

Make and Model	Max Flight Time	Charging Time
DJI Mavic 2 Zoom	32 minutes	90+ minutes
DJI Mavic Pro	27 minutes	60+ minutes
Blade Chroma 4K	30 minutes	60+ minutes
DJI Phantom 4 Pro	30 minutes	210 minutes
Traxxas Aton Plus	25 minutes	90 minutes
Yuneec Typhoon H	25 minutes	50+ minutes
DJI Phantom 3 Standard	25 minutes	60+ minutes
Yuneec Q500+	25 minutes	120 minutes
Parrot Bebop 2	24 minutes	55+ minutes
UPair One	18 minutes	90 minutes
Hubsan H501S	20 minutes	150 minutes

Appendix B: Multi-trip VRP algorithms against benchmark instances

From [28] the best known VRP Multi-trip algorithms are tested against benchmark instances. If the cell is empty. The algorithm found a feasible solution. When no feasible solution is found, the total length of the solution and the LTR is presented. The LTR is the ratio of the longest route and the maximum working time of the first vehicle.

Problem	*m*	TLG96 Length	TLG96 LTR	BM98 Length	BM98 LTR	PS04 LTR	AAB07 Length	AAB07 LTR	OV07 Length	OV07 LTR	SP07 LTR
CMT-1	3	533.00	1.115	556.34	1.041	1.026	554.37	1.041	558.82	1.024	1.003
	4	546.29	1.027	547.10	1.027	1.085	547.10	1.027	547.10	1.027	1.056
CMT-2	6	841.60	1.032	858.04	1.031	1.019	851.59	1.052			1.068
	7	843.60	1.073	870.07	1.088	1.064	855.59	1.050	873.40	1.009	1.102
CMT-3	5	829.50	1.062			1.052					1.056
	6	842.85	1.032	837.82	1.003	1.061	839.58	1.002			1.050
CMT-4	7	1042.39	1.033	1063.95	1.071	1.072	1053.39	1.041	1072.82	1.002	1.09
	8	1049.02	1.075	1069.54	1.031	1.058	1062.53	1.036			1.1
CMT-5	9			1329.28	1.056	1.024	1327.63	1.041			1.047
	10	1316	1.024	1336.92	1.051	1.064	1350.52	1.045			1.076
CMT-11	4	1042.11	1.020	1069.24	1.011	1.052	1097.05	1.038			1.052
CMT-12	4			819.56	1.012						
	5	819.56	1.050	828.94	1.036		825.65	1.035			1.015
	6	819.56	1.064	819.56	1.072	1.029	819.99	1.113	852.19	1.014	1.024
F-11	2	241.97	1.031	255.12	1.011	1.020					1.020
	3	244.60	1.075	254.40	1.011	1.020			256.85	1.025	1.020

Appendix C: Robot routing algorithm code

```
function [Memory,BestSolAMP]=AMPmainT2(model)
%% Problem Definition

CostFunction=@(L) TourLengthT(L,model);     % Cost Function

T=model.T;                    %maximum time in a day the robots can cycle count
nrobots=model.nrobots;        %amount of robots available for cycle counting
M=model.M;                    %maximum routes stored in the memory
AMPiter=model.AMPiter;        %max iterations of the AMp procedure
In=model.In;                  %amount of initial solutions
maxD=model.maxD;

Memory=cell(1,4);             %memory for storing routes
ActionList=CreatePermActionListT(model.n,model.p,model.N_p);    % Action List

%% Initialization

% Create Empty Individual Structure
empty_individual.L=[];
empty_individual.LD=[];
empty_individual.robots=[];
empty_individual.Cost=[];
empty_individual.Overtime=[];

% Initialize Memory
PIsorted2=model.PIsorted;
sol=empty_individual;

% Initialize Best Solution Ever Found
BestSolAMP=sol;
BestSolAMP.Cost=inf;
BestSolAMP.Overtime=inf;

while numel(Memory(:,1))<In+1
    PIsorted2=model.PIsorted;
    sol=empty_individual;

    for di=1:numel(maxD)
        [sol.L,r]=SWEEPT2(model,PIsorted2,maxD(di));
        sol.LD=CostFunction(sol.L);
        [sol.robots,sol.Cost,sol.Overtime,sol.routeonrobot]=BestfitdecreasingT2(r,T,↙
nrobots,sol.LD,model);
        sol.Cost=sol.Cost+sol.Overtime;
        [sol,r]=TabuSearchT2(sol,r,model);
    end

    % Update Best Solution Ever Found
    if or(sol.Overtime<BestSolAMP.Overtime,sol.Overtime==BestSolAMP.Overtime && sol.↙
Cost<BestSolAMP.Cost)
       BestSolAMP=sol;
    end

    for i=1:r
    Memory(end+1,:)={sol.L{i},sol.Overtime,sol.Cost,sol.robots};
    end
```

```
    disp(['memory count is: ' num2str(numel(Memory(:,1)))]);
end

Memory=Memory(2:end,:);

Memory=sortrows(Memory,[2 3],{'ascend' 'ascend'})

% Array to Hold Best Costs
BestCostAMP=zeros(AMPiter,1);

%% Adaptive memory programming

for iter=1:AMPiter

    PIsorted2=model.PIsorted;
    sol=empty_individual;
    if numel(Memory(:,1))>M
        Memory=Memory(1:M,:);
    end

    Memory2=Memory;
    Memory3=cell(1,4);

    while ~isempty(Memory2)

        %Choose a route from the memory, best routes have highest chance
        chance=0;
        for i=1:numel(Memory2(:,1))
             chance(i)=2*(numel(Memory2(:,1))+1-i)/(numel(Memory2(:,1))*(numel(Memory2
(:,1))+1));
        end

        chosenroute=randsample(numel(Memory2(:,1)),1,true,chance);

        %store the chosen route
        if isempty(Memory3{1,1})
            Memory3(1,:)=Memory2(chosenroute,:);
        else
            Memory3(end+1,:)=Memory2(chosenroute,:);
        end

        %Remove all routes that have customers in common with the chosen route
        counter=numel(Memory2(:,1));
        i=0;
        while i<counter
            i=i+1;
             if sum(ismember(Memory3{end,1},Memory2{i,1}))>0
                 Memory2(i,:)=[];
                 i=i-1;
                 counter=counter-1;
             end
        end
    end
```

```
        %Check if there are customers not routed, if so make new route and add to
existing solution
        for i=1:numel(Memory3(:,1))
            sol.L{i}=Memory3{i,1};
            sol.LD(i)=Memory3{i,3};
            for j=1:numel(Memory3{i,1})
                 PIsorted2(PIsorted2(:,1)==Memory3{i,1}(j),:)=[];
            end
        end

    if ~isempty(PIsorted2)
        rbot=randi(1:numel(maxD),1);
        maxD2=maxD(rbot);
        [Lplus,rplus,LDplus]=SWEEPT2(model,PIsorted2,maxD2);
        for i=1:rplus
            sol.L{end+1}=Lplus{i};
        end
    end
    r=numel(sol.L);

    %Perform Tabu Search procedure on new constructed solution
    sol.LD=CostFunction(sol.L);
    [sol.robots,sol.Cost,sol.Overtime,sol.routeonrobot]=BestfitdecreasingT2(r,T,
nrobots,sol.LD,model);
    sol.Cost=sol.Cost+sol.Overtime;
    [sol,r]=TabuSearchT2(sol,r,model);

    % Update Best Solution Ever Found
    if or(sol.Overtime<BestSolAMP.Overtime-0.0001,sol.Overtime==BestSolAMP.Overtime &&
sol.Cost<BestSolAMP.Cost)
        BestSolAMP=sol;
    end

    % Save Best Cost Ever Found
    BestCostAMP(iter)=BestSolAMP.Cost;

    %Update Memory
    for i=1:r
        Memory(end+1,:)={sol.L{i},sol.Overtime,sol.Cost,sol.robots};
    end

    Memory=sortrows(Memory,[2 3],{'ascend' 'ascend'});

    BestSolAMP.robots
    BestSolAMP.Overtime

    % Show Iteration Information
    disp(['Iteration ' num2str(iter) ': Best Cost = ' num2str(BestCostAMP(iter))]);

end
%%
Memory

if BestSolAMP.Overtime>0
    BestSolAMP.Cost=sum(BestSolAMP.robots);
```

```
end


%% Results

figure;
plot(BestCostAMP,'LineWidth',2);
xlabel('Iteration');
ylabel('Best Cost');
grid on;


end
```

```
function [BestSol,r]=TabuSearchT2(sol,r,model)
%% Problem Definition

T=model.T;
nrobots=model.nrobots;
ScanTime=model.ScanTime;
maxD=model.maxD;
n=model.n;
p=model.p;
N_p=model.N_p;
CostFunction=@(L) TourLengthT(L,model);     % Cost Function

ActionList=CreatePermActionListT(n,p,N_p);    % Action List
nAction=numel(ActionList);                % Number of Actions

%% Tabu Search Parameters

MaxIt=model.MaxIt;                        % Maximum Number of Iterations
thetaMin=model.thetaMin;                      % Minimum tabu rounds
thetaMax=model.thetaMax;                     % Maximum tabu rounds

TL=[thetaMin:thetaMax];         % Tabu Length
Alpha=1;                        % fluctuating penalty for infeasible solutions
AlphaMin=model.AlphaMin;        % minimum penalty
AlphaMax=model.AlphaMax;        % maximum penalty

%% Initialization
%update which customer is on which route and in which position
nodeonroute=zeros(n,3);
al=1;
for i=1:r
    for j=1:numel(sol.L{i})
         nodeonroute(al,:)=[sol.L{i}(j) i j];
         al=al+1;
    end
end
nodeonroute=sortrows(nodeonroute,1);

% Initialize Best Solution Ever Found
BestSol=sol;

% Array to Hold Best Costs
BestCost=zeros(MaxIt,1);

% Initialize Action Tabu Counters
TC=zeros(nAction,1);

%% Tabu Search Main Loop

for it=1:MaxIt

    bestnewsol.Cost=inf;
    bestnewsol.Overtime=inf;

    % Apply Actions
```

```
    for i=1:nAction
        if TC(i)==0
            newsol.L=DoActionT(sol.L,nodeonroute,ActionList{i});
            newsol.LD=CostFunction(newsol.L);
            [newsol.robots,newsol.Cost,newsol.Overtime,newsol.routeonrobot] ↙
=BestfitdecreasingT2(r,T,nrobots,newsol.LD,model);
            newsol.Cost=newsol.Cost+Alpha*newsol.Overtime;
            newsol.ActionIndex=i;
            if or(newsol.Overtime<bestnewsol.Overtime-0.0001,newsol.↙
Overtime==bestnewsol.Overtime && newsol.Cost<bestnewsol.Cost) && sum(newsol.↙
routeonrobot(:,3)<=newsol.routeonrobot(:,5))==r
                bestnewsol=newsol;
            end
        else                                                        %for↙
aspiration (if the tabu move kan improve the best solution so far, its tabu status↙
gets overridden
            newsol2.L=DoActionT(sol.L,nodeonroute,ActionList{i});
            newsol2.LD=CostFunction(newsol.L);
            [newsol2.robots,newsol2.Cost,newsol2.Overtime,newsol2.routeonrobot] ↙
=BestfitdecreasingT2(r,T,nrobots,newsol.LD,model);
            newsol2.Cost=newsol2.Cost+Alpha*newsol2.Overtime;
            newsol2.ActionIndex=i;

            if or(newsol2.Overtime<BestSol.Overtime-0.0001,newsol2.Overtime==BestSol.↙
Overtime && newsol2.Cost<BestSol.Cost) && sum(newsol2.routeonrobot(:,3)<=newsol2.↙
routeonrobot(:,5))==r
                bestnewsol=newsol2;
            end

        end
    end

    % Update Current Solution
    sol=bestnewsol;
    emptyrouteindex=zeros(1,r);

    %if due to customer insertion a route is emtpy, remove that route
    for i=1:r
        if isempty(sol.L{i})
            emptyrouteindex(i)=1;
        end
    end

    for i=1:numel(emptyrouteindex)
        if emptyrouteindex(i)==1
            sol.L(i)=[];
            sol.LD(i)=[];
            r=r-1;
        end
    end

    %update which customer is on which route and in which position
    al=1;
    nodeonroute=zeros(n,3);
    for i=1:r
```

```
        for j=1:numel(sol.L{i})
             nodeonroute(al,:)=[sol.L{i}(j) i j];
             al=al+1;
        end
    end
    nodeonroute=sortrows(nodeonroute,1);

    % Update Tabu List
    for i=1:nAction
        if i==bestnewsol.ActionIndex
            TC(i)=randsample(TL,1);    % Add To Tabu List
        else
            TC(i)=max(TC(i)-1,0);    % Reduce Tabu Counter
        end
    end

    % Update Best Solution Ever Found
    if or(sol.Overtime<BestSol.Overtime-0.0001,sol.Overtime==BestSol.Overtime && sol.↙
Cost<BestSol.Cost) %sol.Cost<=BestSol.Cost
       for i=1:r
           pnew=sol.L{i};
           [sol.L{i},sol.LD(i)]=tsp2optT(pnew,model.DD,model.d0,ScanTime);
       end

       [sol.robots,sol.Cost,sol.Overtime,sol.routeonrobot]=BestfitdecreasingT2(r,T,↙
nrobots,sol.LD,model);
       sol.Cost=sol.Cost+Alpha*sol.Overtime;
       BestSol=sol;
       %update which customer is on which route and in which position
       al=1;
       nodeonroute=zeros(n,3);
       for i=1:r
           for j=1:numel(sol.L{i})
                nodeonroute(al,:)=[sol.L{i}(j) i j];
                al=al+1;
           end
       end
       nodeonroute=sortrows(nodeonroute,1);
    end

    %Update Alpha
    if sol.Overtime==0
        Alpha=Alpha/2;
        if Alpha<AlphaMin
            Alpha=AlphaMin;
        end
    else
        Alpha=Alpha*2;
        if Alpha>AlphaMax
            Alpha=AlphaMax;
        end
    end

    % Save Best Cost Ever Found
    BestCost(it)=BestSol.Cost;
```

```
    % If Global Minimum is Reached
    if BestCost(it)==0
        break;
    end
end
```

```
function [LD]=TourLengthT(L,model)

% model = CreateModelforroute2();       % Create TSP Model
    n=model.n;
    maxD=model.maxD;
    d0=model.d0;
    d=model.d;
    ScanTime=model.ScanTime;
    r=numel(L);

    for i=1:r
        if numel(L{i})>1
            LD(i)=d0(L{i}(1))+d0(L{i}(end));
            for j=1:numel(L{i})-1
                 LD(i)=LD(i)+d(L{i}(j),L{i}(j+1))+ScanTime;
            end
        elseif numel(L{i})==1
           LD(i)=d0(L{i}(1))+d0(L{i}(end))+ScanTime;
        else
            LD(i)=0;
        end
    end
end
```

```
function [L,r,LD,check] = SWEEPT2(model,PIsorted2,maxD)

    n=model.n;
    PIsorted=model.PIsorted;
    d0=model.d0;
    DD=model.DD;
    d=model.d;
    Pangle=model.Pangle;
    AVR=model.AVR;
    ScanTime=model.ScanTime;
    n=numel(PIsorted2(:,1));
    Randpol=randi([1 n],1);          %moet straks uit de functie want nu is het altijd↙
het zelfde random getal!!!
    w=Randpol;                       %the index number of the randomly choses customer
    q=zeros(1,n);                    %the array sorted by polar angle, from the↙
randomly chosen begin customer

    if w==1
       q=[PIsorted2(1:n,1)]';
    else
       q=[PIsorted2(w:n,1);PIsorted2(1:(w-1),1)]';
    end

    r=1;                 %route r in cell array L
    v=1;                 %customer v in route r
    Ncount=n;            %while there are customrs unrouted
    L=cell(n,1);            %list of routes
    LD=zeros(numel(L),1);   %total distance of route r in L
    TOBEROUTED=q;
    KIIreturn=[];


    while ~Ncount==0                                                  %assigning all the↙
customers to a route via polar angle coordinate

           if 2*max(d0)>maxD                                          %if de max distance↙
capacity is more than going from the depot to the node and back then the algorithm↙
will stay in a loop
               disp("maxD to low!!!!!!!!!!!!!!!!!!!!!!");
               break;
           end

           if numel(TOBEROUTED)==0 && numel(KIIreturn)>0

               r=r+1;
               v=1;
               L{r}=KIIreturn;
               [L{r},LD(r)]=tsp2optT(L{r},DD,d0,ScanTime);
               Ncount=1;
               KIIreturn=[];

           else

               L{r}(v)=TOBEROUTED(1);
               [L{r},LD(r)]=tsp2optT(L{r},DD,d0,ScanTime);
```

```
            end

            if LD(r)>maxD

               L{r}=L1;
               LD(r)=LD1;
              %for initializing step 8
               if numel(TOBEROUTED)<4
                  f4TOBEROUTED=TOBEROUTED(1:numel(TOBEROUTED));
              else
                  f4TOBEROUTED=TOBEROUTED(1:4);
               end
               [Y,I]=min(d(TOBEROUTED2(1),TOBEROUTED));     %JJX is the nearest customer↙
to the customer last added
               JJX=TOBEROUTED(I);
               TOBEROUTEDTEMP=TOBEROUTED;
               TOBEROUTEDTEMP(TOBEROUTEDTEMP==JJX)=[];
               [Y,I]=min(d(JJX,TOBEROUTEDTEMP));    %JII is the custoemr that is nearest↙
to JJX
               JII=TOBEROUTEDTEMP(I);
               [Y,I]=min(d0(L{r})+(Pangle(L{r})'*AVR)); % KII is hier de index van de↙
customer dichtst bij de depot en dichtst bij de volgende route
               KII=L{r}(I);          %om van een index naar een customer te gaan
               %D2
               D2=L{r};
               D2(D2==KII)=[];
               D2(end+1)=JJX;
               [D2,LD2]=tsp2optT(D2,DD,d0,ScanTime);
                   if LD2<=maxD && numel(JJX)>0 && ismember(JJX,f4TOBEROUTED)

                       %D3
                       if numel(TOBEROUTED)<5
                          [D3,LD3]=tsp2optT(TOBEROUTED(1:numel(TOBEROUTED)),DD,d0,↙
ScanTime);
                       else
                          [D3,LD3]=tsp2optT(TOBEROUTED(1:5),DD,d0,ScanTime);
                       end
                       %D4

                       D4=D3;
                       D4(D4==JJX)=[];
                       D4(end+1)=KII;
                       [D4,LD4]=tsp2optT(D4,DD,d0,ScanTime);
                       if numel(JII)>0 && numel(D4)>1 && ismember(JII,f4TOBEROUTED) &&↙
LD(r)+LD3<LD2+LD4
                           %D5

                           D5=D2;
                           D5(end+1)=JII;
                           [D5,LD5]=tsp2optT(D5,DD,d0,ScanTime);
                           %D6

                           D6=D4;
                           D6(D6==JII)=[];
                           [D6,LD6]=tsp2optT(D6,DD,d0,ScanTime);
```

```
                    end
                end

            if LD2<=maxD && numel(JJX)>0 && ismember(JJX,f4TOBEROUTED) && LD(r)
+LD3>=LD2+LD4
                L{r}=D2;
                LD(r)=LD2;
                L1=L{r};
                LD1=LD(r);
                TOBEROUTED=TOBEROUTEDTEMP;
                TOBEROUTED2=[JJX TOBEROUTED];
                KIIreturn(end+1)=KII;
            elseif LD2<=maxD && ismember(JJX,f4TOBEROUTED) && LD(r)+LD3<LD2+LD4 &&
numel(JII)>0 &&ismember(JII,f4TOBEROUTED) && LD5<maxD && LD5+LD6<LD(r)+LD3
                L{r}=D5;
                LD(r)=LD5;
                L1=L{r};
                LD1=LD(r);
                TOBEROUTEDTEMP(TOBEROUTEDTEMP==JII)=[];
                TOBEROUTED=TOBEROUTEDTEMP;
                KIIreturn(end+1)=KII;
                TOBEROUTED2=[JII TOBEROUTED];
                v=v+1;
                Ncount=Ncount-1;
            else
                r=r+1;
                v=1;
                TOBEROUTED=[KIIreturn TOBEROUTED];
                KIIreturn=[];
                L1=[];
                LD1=[];
            end
        else
          if numel(TOBEROUTED)>0
                LD1=LD(r);
                L1=L{r};
                TOBEROUTED2=TOBEROUTED;
                TOBEROUTED(1)=[];
                v=v+1;
                Ncount=Ncount-1;
            else
                Ncount=Ncount-1;
            end
        end
    end

    L=L(1:r);
    LD=LD(1:r);
    Check=0;

    for i=1:r
    L{i};
    Check=Check+numel(L{i});
    end
```

```
    Check;


end
```

```
function [pnew,LD] = tsp2optT(pnew,DD,d0,ScanTime)

pnew=pnew+1;                        %adding 1 to all products for adding the depot as
number 1
pnew=[1 pnew];                     %adding the depot as the start

lolol=1;

D=DD;                               %distance matrix including the depot
dd=D(2:end,2:end);                  %distance matrix without the depot
n = numel(pnew);                   %amount of products in the route including the depot

zmin = -1;

pnew(1:n)=pnew(n:-1:1);             %making the depot the end of the route (for the
purpose of the algorithm)

while zmin < -0.1*10^-3

    zmin = 0;
    i = 0;
    b = pnew(n);

    while i < n-1
        a = b;
        i = i+1;
        b = pnew(i);
        Dab = D(a,b);
        j = i+1;
        d = pnew(j);
        while j < n
            c = d;
            j = j+1;
            d = pnew(j);
            z = (D(a,c) - D(c,d)) + D(b,d) - Dab;
            if z < zmin
                zmin = z;
                imin = i;
                jmin = j;
            end
        end
    end

    if zmin < 0
        pnew(imin:jmin-1) = pnew(jmin-1:-1:imin);
    end
    lolol=lolol+1;
    if lolol>80000
        disp("error in 2-opt")
    end
end

pnew(end)=[];
pnew=pnew-1;
```

```
n=numel(pnew);
pnew(1:n)=pnew(n:-1:1);
LD=d0(pnew(1));
if n>1
    for i=1:n-1
        LD=LD+dd(pnew(i),pnew(i+1))+ScanTime;
    end
LD=LD+d0(pnew(end));
else
    LD=LD+d0(pnew(end));
end

end
```

```
function [robots,LDR,Overtime,routeonrobot]=BestfitdecreasingT2(r,T,nrobots,LD,model)

    maxD=model.maxD;
    chargeto15=model.chargeto15;
    chargeto85=model.chargeto85;
    chargeto15max=model.chargeto15max;
    robots=zeros(1,nrobots);
    routeonrobot=zeros(r,5);

    %Cost assosiated with each route
    for i=1:r
        LDE(i,:)=[LD(i) i];
    end

    %Assign longest route to the most "empty" vehicle
    LDE2=LDE;
    at=1;
    for i=1:r
        recharging=0;
        [Y,I]=max(LDE(:,1));
        rou=I;
        robotX=zeros(2,numel(robots));
        for i=1:numel(robots)
            robotX(:,i)=[i;robots(i)];
        end
        for i=1:numel(robots)
            if LDE(rou,1)>maxD(i)*1.05
                robotX(:,robotX(1,:)==i)=[];
            end
        end
        [Y,I]=min(robotX(2,:));
        bot=robotX(1,I);
        LDE(rou,1);
        max(LDE(:,1));
        %after every route the robot recharges
        if LDE(rou,1)<0.15*maxD(bot)
           recharging=(LDE(rou,1)*chargeto15);                        %recharging time↙
from 85-100% is 1 hour
        else
           recharging=((LDE(rou,1)-0.15*maxD(bot))*chargeto85)+chargeto15max;    %↙
recharging time from 0-85% is 1 hour + 1 hour 85-100%
        end
        robots(bot)=robots(bot)+LDE(rou,1)+recharging;
        routeonrobot(at,:)=[bot LDE(rou,2) LDE(rou,1) recharging maxD(bot)];↙
%keep track of which route is assigned to which vehicle
        at=at+1;
        LDE(rou,:)=[];
    end
    routeonrobot=sortrows(routeonrobot,1);
    LDE=LDE2;
    routeonrobot2=routeonrobot;

    %If there is overtime, try swapping routes with the most overtime vehicle
    Overtime=0;
```

```
    for i=1:nrobots
        if robots(i)-T(i)>0             %total overtime of all the robots
            Overtime=Overtime+(robots(i)-T(i));
        end
    end

    if sum(robots<=T)<nrobots
        robotcounter=[(1:numel(robots))' robots'];
        robotcounter=sortrows(robotcounter,2,'descend');

        for i=1:numel(robots)
            if robotcounter(i,2)>T(i)
                change=0;
                Overtime2=0;
                bot=robotcounter(i,1);
%                 botcount=0;
                routeonrobotMAX=[];
                while ismember(bot,routeonrobot(:,1))       %Delete the routes that↙
are on the chosen (max overtime)vehicle
%                         botcount=botcount+1
                        routeonrobotMAX=[routeonrobotMAX;routeonrobot(routeonrobot(:,↙
1)==bot,:)];
                        routeonrobot(routeonrobot(:,1)==bot,:)=[];
                end
                robots2=robots;
                bestrobots=robots2;
                for i=1:numel(routeonrobotMAX(:,1))                             %For↙
all routes assigned to the chosen vehicle try swapping a route with another vehicle
                    for j=1:numel(routeonrobot(:,1))
                        if routeonrobotMAX(i,3)<maxD(routeonrobot(j,1))
                        robots2(bot)=robots2(bot)-routeonrobotMAX(i,3)-routeonrobotMAX↙
(i,4);
                        robots2(bot)=robots2(bot)+routeonrobot(j,3)+routeonrobot(j,4);
                        robots2(routeonrobot(j,1))=robots2(routeonrobot(j,1))-↙
routeonrobot(j,3)-routeonrobot(j,4);
                        robots2(routeonrobot(j,1))=robots2(routeonrobot(j,1))↙
+routeonrobotMAX(i,3)+routeonrobotMAX(i,4);
                        end
                        Overtime2=0;
                        for go=1:nrobots
                            if robots2(go)-T(go)>0              %total overtime of all↙
the robots
                                Overtime2=Overtime2+(robots2(go)-T(go));
                            end
                        end
                        if  Overtime2<Overtime    %keep track of the best route swap↙
%sum(robots2<=T)==nrobots && sum(robots2)<sum(bestrobots)
                            change=1;
                            bestrobots=robots2;
                            newroute1=[bot routeonrobot(j,2) routeonrobot(j,3)↙
routeonrobot(j,4) maxD(bot)];
                            newroute2=[routeonrobot(j,1) routeonrobotMAX(i,2)↙
routeonrobotMAX(i,3) routeonrobotMAX(i,4) routeonrobot(j,5)];
                        end
                        robots2=robots;
```

```
                    end
                end
                if change==1                            %if an improvement is found
make the route swap between vehicles
                    routeonrobot3=routeonrobot2;
                    routeonrobot2(routeonrobot3(:,2)==newroute2(1,2),:)=newroute1;
                    routeonrobot2(routeonrobot3(:,2)==newroute1(1,2),:)=newroute2;
                end

                robots=bestrobots;
                routeonrobot=routeonrobot2;
            end
        end
    end

    LDR=sum(robots);        %total time of all the robots
    Overtime=0;

    for i=1:nrobots
        if robots(i)-T(i)>0             %total overtime of all the robots
            Overtime=Overtime+(robots(i)-T(i));
        end
    end

end
```

```
function ActionList=CreatePermActionListT(n,p,N_p)

    nSwap=n*(p-1);
    nInsertion=n*(p-1);
    nInsertionbefore=n*(p-1);
    nAction=nSwap+nInsertion+nInsertionbefore;

    ActionList=cell(nAction,1);

    c=0;

    % Add SWAP
    for i=1:n
        for j=N_p(i,2:p)
            if ~ismember(i,N_p(j,2:p))
            c=c+1;
            ActionList{c}=[1 i j];
            end
        end
    end

    % Add Insertion after
    for i=1:n
        for j=N_p(i,2:p)
                c=c+1;
                ActionList{c}=[2 i j];
        end
    end

    % Add Insertion before
    for i=1:n
        for j=N_p(i,2:p)
            if abs(i-j)>1
                c=c+1;
                ActionList{c}=[3 i j];
            end
        end
    end

    ActionList=ActionList(1:c);

end
```

```
function LE=DoActionT(L,nodeonroute,a)

    switch a(1)
        case 1
            % Swap
            LE=DoSwapT(L,nodeonroute,a(2),a(3));
        case 2
             % Insertion after
            LE=DoInsertionT(L,nodeonroute,a(2),a(3));
        otherwise
            % Insertion before
            LE=DoInsertionbeforeT(L,nodeonroute,a(2),a(3));
    end

end
```

```
function LE=DoInsertionbeforeT(L,nodeonroute,i1,i2)

    LE=L;
    p=L{nodeonroute(i1,2)};
    if nodeonroute(i1,2)==nodeonroute(i2,2)
         if abs(nodeonroute(i1,3)-nodeonroute(i2,3))<2
            LE{nodeonroute(i1,2)}([nodeonroute(i1,3) nodeonroute(i2,3)])=p↙
([nodeonroute(i2,3) nodeonroute(i1,3)]);
         elseif nodeonroute(i1,3)<nodeonroute(i2,3)
            LE{nodeonroute(i1,2)}=p([1:nodeonroute(i1,3)-1 nodeonroute(i1,3)+1:↙
nodeonroute(i2,3)-1 nodeonroute(i1,3) nodeonroute(i2,3):end]);
        else
            LE{nodeonroute(i1,2)}=p([1:nodeonroute(i2,3)-1 nodeonroute(i1,3)↙
nodeonroute(i2,3):nodeonroute(i1,3)-1 nodeonroute(i1,3)+1:end]);
        end
    else
            LE{nodeonroute(i1,2)}=[L{nodeonroute(i1,2)}(1:nodeonroute(i1,3)-1) i2 L↙
{nodeonroute(i1,2)}(nodeonroute(i1,3):end)];
            LE{nodeonroute(i2,2)}=[L{nodeonroute(i2,2)}(1:nodeonroute(i2,3)-1) L↙
{nodeonroute(i2,2)}(nodeonroute(i2,3)+1:end)];
    end
end
```

```
function LE=DoInsertionT(L,nodeonroute,i1,i2)

    LE=L;
    p=L{nodeonroute(i1,2)};
    if nodeonroute(i1,2)==nodeonroute(i2,2)
        if abs(nodeonroute(i1,3)-nodeonroute(i2,3))<2
            LE{nodeonroute(i1,2)}([nodeonroute(i1,3) nodeonroute(i2,3)])=p↙
([nodeonroute(i2,3) nodeonroute(i1,3)]);
        elseif nodeonroute(i1,3)<nodeonroute(i2,3)
            LE{nodeonroute(i1,2)}=p([1:nodeonroute(i1,3)-1 nodeonroute(i1,3)+1:↙
nodeonroute(i2,3) nodeonroute(i1,3) nodeonroute(i2,3)+1:end]);
        else
            LE{nodeonroute(i1,2)}=p([1:nodeonroute(i2,3) nodeonroute(i1,3) nodeonroute↙
(i2,3)+1:nodeonroute(i1,3)-1 nodeonroute(i1,3)+1:end]);
        end
    else
            LE{nodeonroute(i1,2)}=[L{nodeonroute(i1,2)}(1:nodeonroute(i1,3)) i2 L↙
{nodeonroute(i1,2)}(nodeonroute(i1,3)+1:end)];
            LE{nodeonroute(i2,2)}=[L{nodeonroute(i2,2)}(1:nodeonroute(i2,3)-1) L↙
{nodeonroute(i2,2)}(nodeonroute(i2,3)+1:end)];
    end
end
```

```
function LE=DoSwapT(L,nodeonroute,i1,i2)

    LE=L;
    if nodeonroute(i1,2)==nodeonroute(i2,2)
        p=L{nodeonroute(i1,2)};
        LE{nodeonroute(i1,2)}([nodeonroute(i1,3) nodeonroute(i2,3)])=p([nodeonroute
(i2,3) nodeonroute(i1,3)]);
    else
        LE{nodeonroute(i1,2)}(nodeonroute(i1,3))= i2;
        LE{nodeonroute(i2,2)}(nodeonroute(i2,3))= i1;
    end
end
```

Appendix D: Logistic regression algorithm code

```
function newtonmethod5 ()
clc; close;

Var1=struct2cell(load('Var1Train.mat'));
Var2=struct2cell(load('Var2Train.mat'));
Montez=struct2cell(load('MontezTrain.mat'));
Montey=struct2cell(load('MonteyTrain'));

xx1=[];
xx2=[];
xx3=[];
yy=[];
for i=1:numel(Var1{1})
        xx1=[xx1 Var1{1}{i}];
        xx2=[xx2 Var2{1}{i}];
        xx3=[xx3 Montez{1}{i}];
        yy=[yy Montey{1}{i}];
end

xx=[xx1' xx2' xx3'];
yy=yy';

rng('default');
rng(1);
jo=randperm(numel(xx(:,1)));
jotrain=jo(1:round(0.33*numel(xx(:,1))))';
jotest=jo(round(0.33*numel(xx(:,1)))+1:end)';

xtest=xx(jotest,:);
ytest=yy(jotest,:);

x=xx(jotrain,:);
y=yy(jotrain,:);

[sizes, r] = size(x)
[sizesTest, rtest] = size(xtest)

m=ones(sizes,1);

X = [ones(sizes, 1), x];
Xtest=[ones(sizesTest, 1), xtest];

beta=zeros(r+1,1) ;             %coeficients of the factors +1 for the stability↙
coeficient
beta(1,1)=log(sum(y)/sum(m-y));%coeficients of the factors +1 for the stability↙
coeficient
p=prob(X,beta) ;                %probability
betaOld=beta  ;                 %oldcoefficients for update

MaxIt=20 ;                      %max itertions
tolerance=1*10^-9 ;             %tollerance for convergens
iterCount=0 ;                   %itertion counter
madeChange=1;                   %If no change is made stop iterating
Cost=zeros(MaxIt,1);            %the cost over all data points for every iteration
```

```
while madeChange==1 && iterCount<MaxIt
     iterCount=iterCount+1;
     p=prob(X,beta);
     beta=betaOld-costFu(X,y,p,m);
     relativeChange=abs(beta-betaOld); %want to see if change is made, can also be on↙
beta values
     madeChange=sum(relativeChange)>tolerance;
     betaOld=beta;
     Cost(iterCount)=-1/sum(m)*likelihood2(y,X,beta,m); %the cost over all data points
 end

Cost=Cost(1:iterCount);

figure
plot(0:iterCount-1, Cost, 'o--', 'MarkerFaceColor', 'r', 'MarkerSize', 8)
xlabel('Iteration'); ylabel('Cost')
savefig('Costgraph.fig')

prob2= round(prob(X,beta));
acc=mean(double(prob2==y)*100);

probTest= round(prob(Xtest,beta));
accTest=mean(double(probTest==ytest)*100)

filename = 'beta.mat';
save(filename, 'beta');
filename = 'Accuracy100Training.mat';
save(filename, 'accTest');

end
```

```
function cost = costFu(X,y,p,m)

gradient=X'*(y-m.*p); %gradient

v=diag(m.*p.*(1-p));  %variance diagonal matrix
hessian=-X'*v*X;      %hessian

cost=hessian\gradient;

end
```

```
function prob= prob(X,beta)

 prob=exp(X*beta)./(1+exp(X*beta));

end
```