ALGAV

3DE

1220514 – Liam Saliën

1220528 – Juuso Jalava

1220379 – Benjamin Moreira

1220459 – Patryk Pas

1220448 – Miika Kauppinen

Contents

[1. Introduction 3](#_Toc123803935)

[2. Creation of the initial population of the Genetic Algorithm (GA) 3](#_Toc123803936)

[3. Random Crossover between individuals of the population 5](#_Toc123803937)

[4. Selection of the new generation of the population 5](#_Toc123803938)

[5. Efficacy Analysis comparing the better individual of the created GA compared with the better from the initial version of the GA. 6](#_Toc123803939)

[6. Parametrization of the ending condition of the AG 7](#_Toc123803940)

[7. Use of the GA to handle several trucks, representing in the same chromosome the deliveries of the several trucks. 8](#_Toc123803941)

[8. Study of methods of Machine Learning 8](#_Toc123803942)

[9. From a solution obtained from the GA it is envisaged to be able to allow dynamic changes 9](#_Toc123803943)

[10. Conclusions 11](#_Toc123803944)

# 1. Introduction

The main purpose of the sprint C was firstly expanding our knowledge about genetic algorithms and introducing solutions containing these algorithms, by re-using and improving our existing predicates.

Firstly, our group managed to start with the example solution and adapt it to our own predicates and heuristics.

Then, over time we developed and improved the GA code, as we were getting to know the algorithms. We started with improving the creation of the initial population that uses the previously created heuristics, then we implemented random crossover so that the cut was not always at the same positions, we improved the selection of new generations so that it selects at least the best two individuals of each element, we did an efficiency analysis, improved the ending condition so that it also stops whenever the generations are stabilized, and we handled several trucks.

Finally, we managed to do the bibliographic study on the topic of Machine and allowed dynamic changes. Every task is described briefly below.

# 2. Creation of the initial population of the Genetic Algorithm (GA)

For this task, the example solution was adapted to our solution, and modified to include 2 different heuristics solution, and then randomly generated deliveries routes (from warehouse to warehouse):

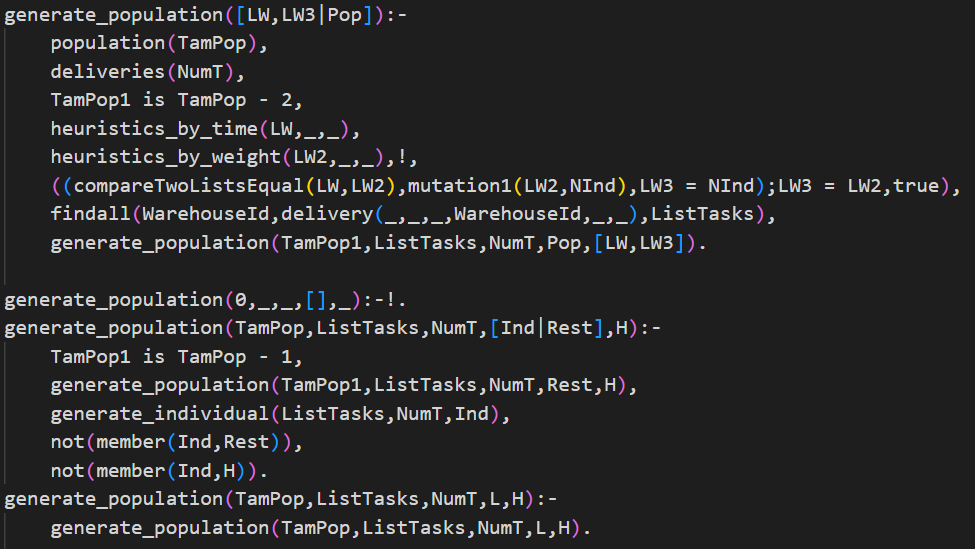


Figure . Updated generation of the initial population predicate.

The first generation population predicate (with one variable) will read the amount of deliveries that are in the system, and after that it will subtract the amount of individuals that are expected by 2, as it was required to include two solutions obtained by two different heuristics. The two heuristics used in our solution are by time, and by weight.

Furthermore it checks if the two solutions doesn’t equal each other (which uses a predicate that compares every single item in two lists).

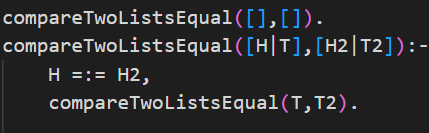


Figure . Predicate to compare if two lists are equal.

If this is the case, a mutation will be done on the second solution (obtained by the weight heuristic).

After this, for the rest of the individuals, another predicate will be called (and will include a list that already contains the two obtained solutions).

This predicate will be recalled until the TamPop is 0, which means we reached the amount of individuals we had to create. It will also check every single time if the generated solution is not already a part of the randomly generated deliveries routes list, and of the list that contains the two solutions obtained by the heuristics.

# 3. Random Crossover between individuals of the population

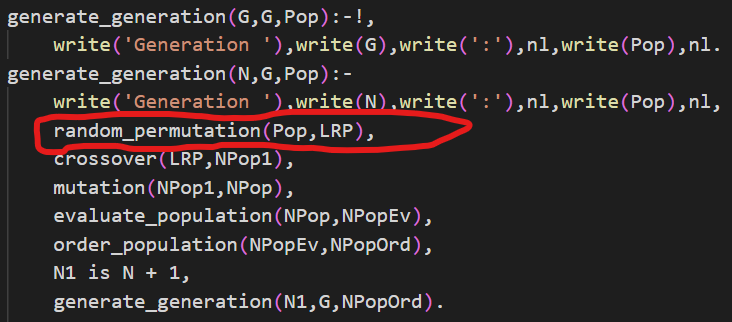


Figure . Updated generating of generation predicate with random crossover.

For this task the generate generation predicate was modified. The build-in predicate random\_permutation/2 is added after the current generation and their individuals are written to the screen.

Right before the written population will go through crossover and mutation etc, it is shifted by this build in predicate, so that the order will be different. Because of this, crossover doesn’t always happen between the first and second warehouseId, etc., etc.

*Example of this predicate in working:*

If we use random\_permutation on the following list:

*[[9,11,1,8,3]\*463.7758474576272,[11,8,1,3,9]\*486.5362288135594,[11,8,3,1,9]\*508.935593220339,[3,1,8,9,11]\*510.73241525423737,[11,3,9,8,1]\*537.482627118644]*

It might lead to this result:

*[[11, 8, 3, 1, 9]\*508.935593220339, [11, 8, 1, 3, 9]\*486.5362288135594, [9, 11, 1, 8, 3]\*463.7758474576272, [11, 3, 9, 8|...]\*537.482627118644, [3, 1, 8|...]\*510.73241525423737]*

This way our solution works more randomly, as well as the crossover takes place more randomly.

# 4. Selection of the new generation of the population

The selection of the new generation of the population considers the elements of the previous population and their descendants after crossover and mutation. It is important to consider that the mutation can change a descendant after crossover, potentially resulting in a better individual.



Figure . New generate of generation predicate with 3 parameters.

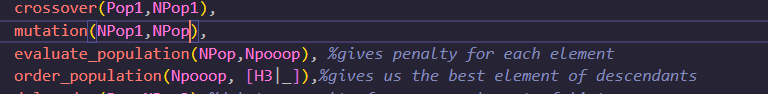


Figure . Basic GA predicates inside the newly geration predicate.

The two best individuals of each element (previous generation (H) and descendants (H3), *Figure 4 and 5*) are automatically included in the new generation. For the remaining elements, a tournament selection process is used to determine which individuals will be included in the new generation. In this process, any element can proceed, but individuals with better evaluations have a higher probability of being selected.

To implement this selection process, we can define a predicate that takes in the previous generation and the descendants and returns the new generation. This predicate can first identify the two best individuals from the previous generation and descendants, and then use the tournament selection process to determine the remaining elements of the new generation.

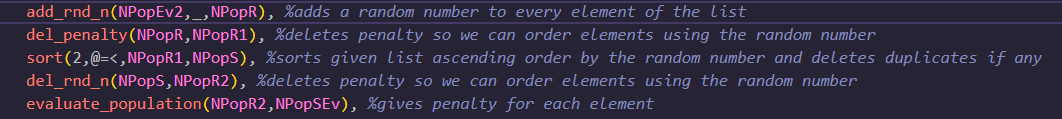


Figure . Mutation predicates.

The tournament selection can be implemented using a random permutation, where we randomly select a group of individuals and compare their evaluations to determine the winner. The winner of the tournament is then added to the new generation, and the process is repeated until all the remaining elements have been selected.

# 5. Efficacy Analysis comparing the better individual of the created GA compared with the better from the initial version of the GA.

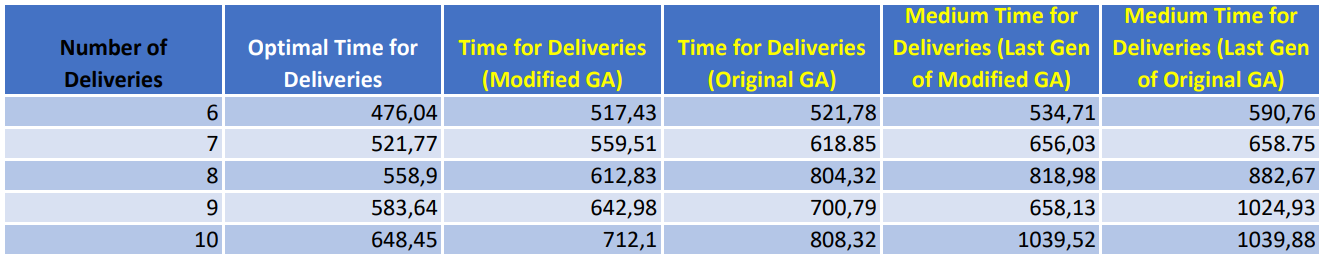


Figure . Table of results of optimal solution, original GA and modified GA.

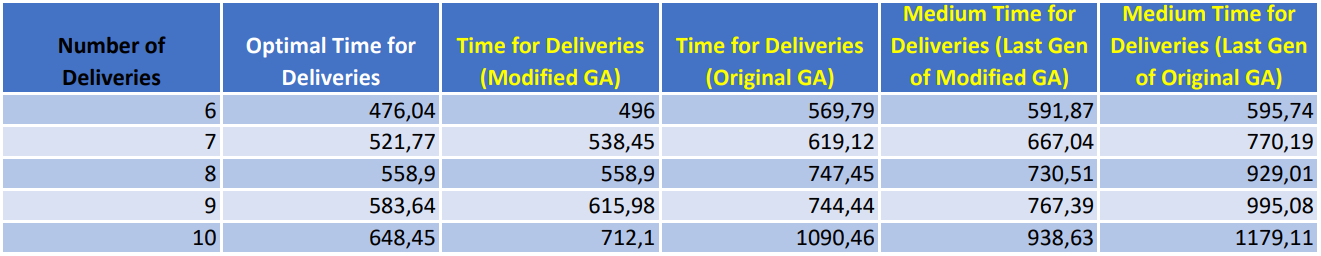


Figure . Table of results of optimal solution, original GA and modified GA. Different input values than Figure 4.

For the analysis we used the original list of deliveries given to us at the start of project. For *Figure 7* the number of generations we used was 6, population size 5, crossover probability 50 % and mutation probability 25 %. In *Figure 5* for modified GA we used 100 generations, population size 30, crossover probability 50 % and mutation 25 %. For original GA in *figure 8* we used 1000 generations, population size 5, crossover probability 10 % and mutation 5 %. In optimal solution the time needed to make higher number of deliveries is always higher. In theory that should also be the case Genetic Algorithm, but given the randomness in GA the better solution is not always found. Increasing the number of generations allows modified GA to find better solutions, but that’s not the case with original GA since it doesn’t save the better solutions for new generations.

# 6. Parametrization of the ending condition of the AG

Text

Description automatically generated

Figure . Updated generating of generation predicate that includes stabilization check.

The regular flow of a Genetic Algorithm flow (including crossover, mutation, etc.) was modified by including a stabilization check. If the populations of the latest generations (assembled by the append predicate) are all the same, we say it is stabilized (will be explained further below) and the predicate can stop. If not, it should do all the modifications and recall itself to generate new generations as long as it is not stabilized OR we didn’t reach the request amount of generations as mentioned/given during the initialization.

Text

Description automatically generated

Figure . Stabilization check predicate.

We agreed that a population is “stabilized” when 10 generations after each other do have the same populations. This is a random number that was chosen, as it cannot be too small, and not too big, and stabilization doesn’t have only one possible solution/meaning (everyone can choose another definition of when something is “stable”). Is there a possibility new solutions will be created after this? Yes, but the obtained solution (when our program says it’s stable), is valid enough and won’t differ that much from further generated possibilities (as 10 times equal is already a lot).

These predicate firstly checks if there are already more than 10 generations, and if this is the case, it will reverse the generations as we only need the 10 latest ones. It will then check the latest population with the 9 populations that were generated before that one, and if all are equal, we can say it is stabilized.

# 7. Use of the GA to handle several trucks, representing in the same chromosome the deliveries of the several trucks.

For GA to run with several trucks we need to separate individuals. Every truck holds 5-6 deliveries. First, we must check how many trucks we need (figure 11). Predicate takes account if the number of trucks we need is close to bigger number (e.g.,1,95) it chooses 3 truck to handle the deliveries.

Text

Description automatically generated

Figure . predicate to check how many deliveries is needed

After generating the initial population, we need to separate populations for each truck (figure 12).

[4,15,7,8,17,3,2,10,12,6,14,1,11,13,16,9] = [[15,4,1,8],[11,17,10,2,6,7],[9,16,12,13,3,14]].

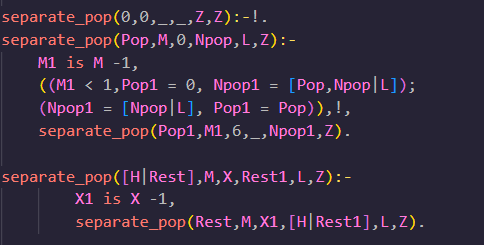


Figure . Predicate to separate populations

Then we evaluate populations and put them in descending order because the time is biggest time of all trucks.

[[1,14,6,12,10,2]\*598.6396186440678,[3,17,8,7,15,4]\*414.91864406779666,[11,13,16,9]\*296.25]].

# 8. Study of methods of Machine Learning

*Machine learning (ML)* is a subset of AI that is focused on teaching computers to learn from data and to improve it with experience instead of being explicitly programmed to do so. The algorithms are trained to find patterns in large data sets to make the best decisions and predictions based on that analysis.

Those applications improve with use and become more accurate the more data they have access to.[[1]](#footnote-1)

As Machine Learning is a very big topic to cover, there are a lot of methods present to actually apply it to certain problems.

Below we study multiple of those methods, in order for us to both understand the different Machine Learning methods, and how to apply them to our problem of performing deliveries from warehouse to warehouse (starting from one central warehouse located in Matosinhos), and taking into account the battery levels (that shouldn’t go below a previously stated minimum of 20%) and the most optimal route that has the lowest cost (which is also based on the total weight of the electric truck and its payload):

1. A Machine learning method that can be used to estimate the energy consumption of the electric trucks during the deliveries as accurate as possible is the **Bayesian** Machine learning method, or more precisely, the **Bayesian regression** method. It’s a type of Supervised Learning.[[2]](#footnote-2)

This will obtain/estimate both the optimal as the variance (of) energy consumption based on an initial prior, and afterwards improve this precision by combining it with (new) empirical data[[3]](#footnote-3) (= data based on what is experienced/retrieved earlier[[4]](#footnote-4)).

Thanks to the obtained variance, this method makes it also possible to plan routes more efficient and predict the probability of higher energy consumptions.[[5]](#footnote-5)

The method works as follows:

It needs to find 2 coefficients alpha and beta for each road link (randomly), to calculate the energy probability distribution for that road link. For this it first computes a *prior* based on **map data and vehicle model**, secondly finds the *likelihood* based on **measured data** (from vehicles driving on the road) that will be combined with this prior to become the *posterior*, and finally this will lead to the *posterior predictive[[6]](#footnote-6)*.

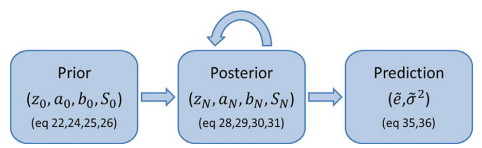


Figure . Bayesian Machine Learning method.[[7]](#footnote-7)

If we look into more detail, the likelihood function contains information regarding the total energy consumed, the total mass and the average speed. The function that is obtained using among other things matrix notations is combined with the prior, that consists of both a mean (obtained by Gaussian distribution) and a variance (obtained by Inverted Gamma distribution). To retrieve the initial data for the prior, a probabilistic speed profile for each road link in the road network should be used.[[8]](#footnote-8)

The posterior method will then be a result of these two combined, and thanks to ML the system can learn recursively by using the newest measurements of energy, mass, and speed every time a vehicle drives a certain road link.

The result in the end will then be a posterior distribution of coefficients alpha and beta to predict energy consumption when given an input of mass and speed for a certain road link.[[9]](#footnote-9)

1. Machine Learning can also help us to predict the demand for every warehouse for the next day, so that we can (for example) predict the amount of trucks (and therefore truck drivers) in advance.

To predict the amount of deliveries/the deliveries for the next day, the *fuzzy deep contractive autoencoder* Machine Learning model can be used.[[10]](#footnote-10) This is a type of a contractive autoencoder, which is an unsupervised deep learning technique that helps a neural network encode unlabeled training data.[[11]](#footnote-11)

This ML model needs at least the following input features:

1. traffic flow (Average hourly numbers of passengers carried by buses, metros, private cars and pedestrians during 8:00–20:00 and the remaining period)[[12]](#footnote-12)
2. public places (Numbers, average opening hours, and average hourly number of visitors on rail stations, metro stations, bus stations, universities, colleges, middle schools, primary schools, kindergartens, big hotels, medium hotels, small hotels, shopping malls, medium shops, small shops, large farmers markets, small farmers markets, and other unclosed public places)[[13]](#footnote-13)
3. historical data (Amounts of demands of the region and the whole city in the previous seven days). It could also help to know the current economic situation, as this might affect the amount of orders placed, which directly affects the amount of deliveries to be done.[[14]](#footnote-14)

We need all those input features, as these together affect the demand for deliveries.

It then works as follows:

1. The model is a stacked layer of autoencoders, and each encoder consists of an encoder for mapping an input vector x (see input features) into a hidden representation *y* and a decoder to map it back to a reconstructed vector x’.[[15]](#footnote-15)
2. It then will learn in two stages (unsupervised and supervised).[[16]](#footnote-16)
3. Unsupervised: layer by layer training to minimize reconstruction error. Function looks as follows:

min *J* (𝑊 , 𝐛, 𝐛 ′ ) = ∑ 𝐱∈*X* ( ‖𝐱, 𝑓′ (𝑓(𝐱))‖ ). *X* is the training set and ‖𝐱, 𝐱 ′‖ is distance between vector x and x’.[[17]](#footnote-17)

1. Then we perform supervised training of the previous obtained function to minimize the rooted mean square error, and this will result in a function that uses a set of labeled training samples (*Xl*), and has the actual and expected output on each input of x (= each input feature as listed before):

Min *l* = ∑𝐱∈ *Xl* (𝑓̃(𝐱) − 𝑓̂(𝐱))2 )

Thanks to the fuzzy parameters, the uncertain relationship between the input and output can be handled, and demands can be predicted. The same method was used earlier to predict mouth mask demands during covid times.[[18]](#footnote-18)

1. Another machine learning method also allows to determine more precisely the remaining range of electric vehicles. It is a mixed model mixing two machine learning models: *Extreme Gradient Boosting Regression Tree (XGBoost) and Light Gradient Boosting Regression Tree (LightGBM)*.[[19]](#footnote-19)

The prediction work is done in two steps. First, the machine learning model will be fed with different characteristic values of the vehicle and trained in a cloud environment to improve its accuracy. Then, the machine learning model will be used to predict the remaining battery life. As said before, the final model will be a mix of XGBoost and LightGBM. The real data of the vehicles like speed or battery voltage will be retrieved from the CAN bus, then transmitted to a T-Box (which the vehicles are usually equipped with), then sent to the cloud environment.  
The more samples, the more accurate the model will be. The data is then sorted, sampled and processed to determine the relationship between the energy consumed and the distance traveled in km

Diagram

Description automatically generated

Figure - source: Machine Learning-Based Method for Remaining Range Prediction of Electric Vehicles, Yu Wang, 2020

1. Although the autonomy of the batteries is important in the framework of the delivery by electric truck, the health of these batteries is just as important. Indeed, the more a battery is degraded, the more it will lose its maximum autonomy. This is why a model based on machine learning to predict the health of lithium-ion batteries has been created. It uses Autoregressive Integrated Average Modelling (ARIMA) and supervised learning with decision trees as a basic estimator to predict the state of health of the batteries.

In their study[[20]](#footnote-20), The authors considered 31 samples of electric truck battery data and calculated the energy, cumulative energy and capacity of the batteries using the following formulas :

Une image contenant texte

Description générée automatiquement

Figure - source: A DYNAMIC BATTERY STATE-OF-HEALTH FORECASTING MODEL FOR ELECTRIC TRUCKS: LI-ION BATTERIES CASE-STUDY, Matti Huotari , Shashank Arora, Avleen Malhi, Kary Främling, 2020

Using these data, an ARIMA (AutoRegressive Integrated Modeling Average) model was used to create a prediction model. Finally, a regression model was used to determine the SOH (State Of Health) in percentage.

1. In order to minimize unforeseen events and to be more reactive in the context of goods deliveries by electric vehicle, machine learning can predict or signal a breakdown by analyzing the symptoms[[21]](#footnote-21). Using data retrieved from the electrical system of an electric car under different conditions, it is possible to determine a model representative of the electric vehicle. The determination of this model is done by training the model offline with the data.   
   Once the model is trained, it is important to choose a window of analysis, i.e. the time during which an incident can be considered as correlated with an event. With a window of 10 seconds, it is then possible to detect the fault and its potential causes. Indeed, when an error is detected, the model will analyze the different parameters before and after the error in order to be able to draw conclusions about the real cause of the fault.

A picture containing graphical user interface

Description automatically generated

Figure - MACHINE LEARNING BASED ELECTRIC VEHICLE POWER SYSTEM FAULT DIAGNOSIS, Kumar Gaurav, Sengupta Somnath, 2017

In order to determine the real cause of the fault, the Euclidean distance is calculated for each possible cause using the following formula:

*Where p and q are row vectors of length n*

A ranking of the probabilities of correlation between the fault and each possible cause is then made.

1. Finally, machine learning can also be used to determine the Best Possible Route from a list of waypoints by taking into consideration different parameters such as the maximum length of the route, the penalties for being late and the quantity of waypoints. Although the study we are using concerns home food deliveries[[22]](#footnote-22), the principle remains the same with waypoints, and charging time (order ready time in the study) taken into account. Indeed, the authors of the study managed to build a *three-layer, feed-forward artificial neural network (ANN)* to determine the time the order will be ready. We could adapt this method to determine the time needed to charge the truck at each warehouse and then, have a better choice in the path selection.

# 9. From a solution obtained from the GA it is envisaged to be able to allow dynamic changes

First, we must make deliveries dynamic so we can do changes to it (figure 14).



Figure . Predicate that makes delivery dynamic.

We have predicates for creating (figure 15), updating (figure 16), and deleting (figure 17) a delivery. Each predicate asks needed details from the user and carries out given task.

*Add\_delivery* asks user *Id, date, weight, destination warehouse, time to load and time to unload*. After that it uses *asserta* asserts a clause into the beginning of database. After that it writes “New delivery added” and ends the predicate.

Text

Description automatically generated

Figure . Predicate to add delivery.

*Change\_delivery* asks user Id and load. Then gets all other details of the delivery with given id, *retractall* deletes that delivery from the database we want to update and *asserta* creates an updated one. After that it writes “Delivery modified” and ends the predicate.

Text

Description automatically generated

Figure . Predicate to change delivery.

*Delete\_delivery* asks user Id of the delivery we want to delete and *retractall* deletes that delivery from the database. After that it writes “Delivery deleted” and end the predicate.

A picture containing timeline

Description automatically generated

Figure . Predicate to delete delivery.

To get the program to work with dynamic changes we need to make predicate which gives us the number of deliveries when we need (figure18).

Text

Description automatically generated

Figure . Predicate to get number of deliveries

We created predicate *generate\_dynamic* (figure 19), Which runs until *generate\_dynamic1* stops. At the end of *generate\_dynamic1, changes* predicate (figure 20) asks user to give 1, 2, 3 or 0. 0 or any else number or letter stops the predicate.

Text

Description automatically generated

Figure . Generate predicate for dynamic changes

Text

Description automatically generated

Figure . Predicate which asks to type a number and do stuff according to it.

# 10. Conclusions

A successfully working Genetic Algorithm was written for our problem of performing deliveries from warehouse to warehouse (starting from one central warehouse located in Matosinhos), and taking into account the battery levels (that shouldn’t go below a previously stated minimum of 20%) and the most optimal route that has the lowest cost (which is also based on the total weight of the electric truck and its payload).

An example GA was used and adapted to our problem, and lots of improvements has been made to optimize it.

Also, a study on the topic of Machine Learning in the field of electric trucks and route planning was performed, which used the following references:

1. SAP. (n.d.). *What is machine learning? | Definition, types, and examples | SAP Insights*. <https://www.sap.com/insights/what-is-machine-learning.html>
2. Basso et al.-2021,’Electric vehicle routing problem with machine learning for energy prediction’, *Transportation Research Part B*,vol.145,no.1,pp.28-31.
3. Cambridge Dictionary. (2023). empirical definition: 1. based on what is experienced or seen rather than on theory: 2. based on what is experienced or. . .. Learn more.
4. Chen et al.-2023,’ Integration of machine learning prediction and heuristic optimization for mask delivery in COVID-19’,*Swarm and Evolutionary Computation*,vol.76,no.1,pp.4-5.
5. DeepAI. (2019, May 17). *Contractive Autoencoder*. <https://deepai.org/machine-learning-glossary-and-terms/contractive-autoencoder>
6. TBD
7. TBD
8. TBD

1. SAP. (n.d.). *What is machine learning? | Definition, types, and examples | SAP Insights*. https://www.sap.com/insights/what-is-machine-learning.html [↑](#footnote-ref-1)
2. Basso et al.-2021,’Electric vehicle routing problem with machine learning for energy prediction’, *Transportation Research Part B*,vol.145,no.1,pp.28. [↑](#footnote-ref-2)
3. Ibid. [↑](#footnote-ref-3)
4. Cambridge Dictionary. (2023). empirical definition: 1. based on what is experienced or seen rather than on theory: 2. based on what is experienced or. . .. Learn more. https://dictionary.cambridge.org/dictionary/english/empirical [↑](#footnote-ref-4)
5. Basso et al.-2021,’Electric vehicle routing problem with machine learning for energy prediction’, *Transportation Research Part B*,vol.145,no.1,pp.28-29. [↑](#footnote-ref-5)
6. Ibid. [↑](#footnote-ref-6)
7. Basso et al.-2021,’Electric vehicle routing problem with machine learning for energy prediction’, *Transportation Research Part B*,vol.145,no.1,pp.31. [↑](#footnote-ref-7)
8. Ibid., 29. [↑](#footnote-ref-8)
9. Ibid., 30. [↑](#footnote-ref-9)
10. Chen et al.-2023,’ Integration of machine learning prediction and heuristic optimization for mask delivery in COVID-19’,*Swarm and Evolutionary Computation*,vol.76,no.1,pp.4. [↑](#footnote-ref-10)
11. DeepAI. (2019, May 17). *Contractive Autoencoder*. https://deepai.org/machine-learning-glossary-and-terms/contractive-autoencoder [↑](#footnote-ref-11)
12. Chen et al.-2023,’ Integration of machine learning prediction and heuristic optimization for mask delivery in COVID-19’,*Swarm and Evolutionary Computation*,vol.76,no.1,pp.5. [↑](#footnote-ref-12)
13. Chen et al.-2023,’ Integration of machine learning prediction and heuristic optimization for mask delivery in COVID-19’,*Swarm and Evolutionary Computation*,vol.76,no.1,pp.5. [↑](#footnote-ref-13)
14. Ibid. [↑](#footnote-ref-14)
15. Ibid., 4. [↑](#footnote-ref-15)
16. Ibid., 5. [↑](#footnote-ref-16)
17. Ibid. [↑](#footnote-ref-17)
18. Ibid. [↑](#footnote-ref-18)
19. Machine Learning-Based Method for Remaining Range Prediction of Electric Vehicles, Yu Wang, 2020 [↑](#footnote-ref-19)
20. A DYNAMIC BATTERY STATE-OF-HEALTH FORECASTING MODEL FOR ELECTRIC TRUCKS: LI-ION BATTERIES CASE-STUDY, Matti Huotari , Shashank Arora, Avleen Malhi, Kary Främling, 2020 [↑](#footnote-ref-20)
21. MACHINE LEARNING BASED ELECTRIC VEHICLE POWER SYSTEM FAULT DIAGNOSIS, Kumar Gaurav, Sengupta Somnath, 2017 [↑](#footnote-ref-21)
22. Hybrid evolutionary optimization for takeaway order selection and delivery path planning utilizing habit data, Min-Xia Zhang, Jia-Yu Wu, Xue Wu, Yu-Jun Zheng, 2021 [↑](#footnote-ref-22)