Genetic algorithms for clustered VRP

Carmen Quesada Rodríguez  
Degree in Computer Engineering  
*University of Seville*Seville, Spain  
carquerod@alum.us.es

Abstract— Clustering method based on Genetic Algorithms is used in order to solve the Vehicle Routing Problem [1]. The solution combines two approaches: Clustering and Genetic Algorithm (GA) with the purpose of identifying the most optimal delivery routes for a set of vehicles when delivering a set of packages into their destination cities.

Keywords—Clustering, Genetic Algorithm, vehicles, packages, optimization, routes.

# Introduction

The classical Vehicle Routing Problem is composed of a set of packages, with known locations and specific size, and a set of vehicles with a maximum capacity [2]. These vehicles must transport the packages from an initial place to a destination one. The main problem is to deliver all packages without exceeding the capacity of any vehicle and also minimizing the total distance (in kilometres) travelled.

The VRP problem is classified an NP-hard [3] optimization problem in which the required solution time increases with size. By that means, the more packages or cities are involved, more combinations have to be implemented making it harder to find the exact solution.

There are several approaches to solve VRP, in this project, the objective is to develop a hybrid solution, combining two of the techniques learned during Artificial Intelligence course (Clustering and Generic Algorithm). One of the options proposed [4], and the one that I have chosen, is to split the problem into several subproblems according to the distribution obtained in the clustering phase, and then launching several instances of a TSP-solver using genetic algorithms, one for each vehicle.

Clustering [5] goal is to divide an input data set into several subsets (clusters) in such a way that elements within the same subset share some kind of pattern (same origin city). The methodology used is kNN algorithm [6], a nonparametric supervised learning classifier that uses proximity to make predictions about the clustering of a single data point. kNN is one of the most popular classifiers used in machine learning nowadays.

TSP-solver [7] is a software designed to solve Travelling Salesman Problem. The objective is to find the shortest route to travel to a set of cities, visiting each of the cities once.

The optimization technique based on natural selection and evolution is called Genetic Algorithm. It works with an initial population and tries to improve it generation by generation. In this case, each solution represents a possible route that the vehicle can travel to deliver the packages.

Another proposed solution was to create an ad-hoc routine able to generate an initial population such that all individuals are consistent with the previous distribution, and then run a genetic algorithm to solve the VRP problem.

The reason why I have chosen the first option is because I consider myself more familiarized with both methods described in this option (Clustering and Genetic Algorithm) due to the several practices [8] [9] and contents studied in Artificial Intelligence curse. Due to this, I found this approach simpler to implement and test it. In conclusion, I have chosen a solution that follows methods with which I have already practiced, rather than following an ad-hoc routine [10] where the packages assignment and routing are handled simultaneously.

The objective of this project is to find the most optimum route that each vehicle has to follow in order to deliver the packages from the initial city to the destination one, but always taking into account that each package has a size and the vehicles can not exceed their maximum capacity. In addition and in order to make it more realistic, a vehicle can not pick up a package which is not placed in its starting city.

In the following parts of the report the structure of the project is described in detail as well as the decisions that have been taking during its development. Furthermore, the experiments and results obtained are analysed reaching several conclusions.

# Structure

## Structure implementation

For the code structure, a repository available on GitHub was previously created [11]. This facilitates the distribution of the project and is a security measure for saving and tracking every progress made in the code. Additionally, the repository was linked to Visual Studio Code allowing any modification from both environments.

## Files contents

This section analyses the structure in detail of each of the code files.

In this project you can find a folder with the csv files that are used as data. In another folder called graphs, the images generated as results are stored. There is also a README.txt file which summarizes the structure of the source code and indicates how to run and test GA.

The code is divided into three main files: main.py, clustering.py and ga.py.

*main.py*:

This file is the main file of the document, and it connects all the other ones: data folder, clustering.py, ga.py and results. First of all, csv files are read (these files are located in the data/ folder):

packages.csv: contains every package with an id, origin, destination and size.

vehicles.csv: holds every vehicle, their id, origin and capacity.

distances.csv: includes origin and destination city and the kilometers that separates them.

kNN algorithm can not calculate distances between strings. In order to solve this problem, in main.py is declared a dictionary which assigns to each city two coordinates.

*assign\_packages()* function is called from *clustering.py* to distribute packages among vehicles taking into account their origin city and capacity.

*find\_distance(origin, destination)* function is defined in *main.py* to obtain the optimization routes. For each file in distances.csv, if origen is equal to destination, it returns the kilometers, if not it returns 0. For each vehicle this function obtains the ID and starting city, initializing the empty destinations list. For each package, if package is assigned to that vehicle, the destination city is added to the empty list. If vehicle has no assigned packages, an output message is printed indicating it. Therefore, if there is only one destination, it calculates the distance between origin and that city and prints it. If there are more than one destination, fitness function is defined and distance is initialized to 0. For each city in the route it adds the distance between that city and the previous one, returning the total distance.

Genetic problem is created with genes, individuals length and fitness. To execute it some values are set:

Population size = 30

Generations number = 100

Crossover probability = 0.8

Mutation probability = 0.2

Tournament size = 3

The best route and its total distance is printed.

In the last part of the main.py file, a graph has been implemented. First of all, a list of experiments covered 4 experiments of the problem with variations in population size and mutation probability and, consequently, also on fitness. Each of them is executed three times and average fitness of the three executions is obtained. Two empty lists for x and y axis are created. For each experiment, a tag combining population and mutation is created and added to x list. On the other hand, average value of fitness is added to list y. To configure the bar graph, x axis tag is “population-mutation” and y axis is the average fitness. Then graph is saved as an image and showed on screen.

*clustering.py:*

The main use of this file is to assign packages using kNN classification. It contains the *assign\_packages()* function which distributes packages among vehicles using clustering based on kNN algorithm.

*kneighbors* from sklearn is a classifier in which each vehicle is represented as a point using their origin city coordinates. With n\_neighbors = 2, kNN algorithm is applied to predict the closest vehicle to a package. If the vehicle is not full, by that means, it has enough capacity to carry another package, the package is assigned to that vehicle. If not, the function looks for the nearest vehicle available.

The original VRP problem is now divided into several subproblem as it was indicated in option A. Now, it is easier to apply the genetic algorithm to obtain the solution.

*ga.py*:

The genetic algorithm is codified in this file to solve the TSP problem for each vehicle. In this problem, each individual is a possible solution, that is, a route that a vehicle can follow to deliver all its packages.

*class Problem\_Genetic(object)* is the base of the genetic algorithm [9] defining:

*genes*: cities to visit.

*individual\_length*: number of stops.

*decode*: list of cities to visit.

*fitness*: evaluate each individual and penalize.

*mutation* and *crossover*: generate new individuals

*genetic\_algorithm()* is the function which execute the algorithm by creating an initial population and evaluating it fitness. Using tournament selection to choose the parents, it generates a new population (mutation and crossover). This process is repeated and the solution is the best individual, that is, the shortest route for each vehicle.

## Design decisions

During the whole development of the project, several decisions were taken to ensure an efficient and realistic solution.

1. Solving methodology. The first decision was, as it is explained in the introduction, to choose between splitting the problem into several subproblems or creating an ad-hoc routine.
2. Package characteristics. Once the first option was chosen, the specific objectives of the work needed to be declared [4]. The questions were: What information is provided for each package? Packages could only have coordinates to deliver it or they could be added any additional property in order to make it more complicated. The decision was to add “size” as a new feature so that every package had a different one.
3. Vehicle characteristics. Are all vehicles identical? In this problem each vehicle has a specific maximum capacity. At first, the thought was that the easiest part was to give every vehicle the same capacity, but, in order to give more realism to the problem and to add complexity to its resolution, the decision was to give a specific one to each vehicle.
4. Clustering with k-Nearest Neighbors. The kNN classifier for clustering was another decision based on the need to control the assignment of each package taking into account this two factors: vehicle is near the origin city and the vehicle has enough capacity to pick-up the package. As kNN allows taking decisions based on distances and proximity, it was considered the best option.
5. Choosing k. Cross-validation was initially considered in order to choose the best k value, as it was explained in Practice 1 [8]. However, in this problem this option is not possible as each vehicle has an only id identifier, that is, only appears once. There are no repetitions so cross-validation can not be applied as the necessary divisions are impossible to reach. Then, n\_neignors = 2 was also selected. This gives the problem more flexibility because if the first neighbor is full (no more capacity in the vehicle), the second nearest neighbor is chosen, but it does not increase complexity.
6. Cities coordinates. Another decision was to assign coordinates to the city. This is related to the previous decision as city are strings and kNN requires numbers to calculate distances. This coordinates have been generated randomly and are not geographically correct, but solve the conflict of calculating Euclidean distances.
7. Genetic Algorithm structure. The decision was that there was a GA for each vehicle instead of one for all. This made the resolution of the problem less complex as chromosomes were smaller.
8. Chromosomes. This is related to the representation of the individuals as a permutation of cities. Due to this, crossover and mutation were simplified.
9. Results as graphs. Implement two graphs with matplotlib library. One of the graphs shows the optimized routes for each vehicle. The other graph helps to compare the results of the different experiments in an easier and direct way as it represents their average fitness values.

## Methodology

Methodology summarizes the function of the whole code in programming order.

The first step was defining the data of the problem as it was not given. To simulate a real delivery problem, three csv files were created: packages.csv, vehicles.csv and distances.csv. The content of each file is explained in II Structure. This files were randomly created using an IA tool.

The project needed a main module (main.py) so in order to make the code clear and simple, another two files were created in which the clustering and genetic algorithm code are implemented.

Once this was created, the first thing to do was to read the data in the csv files using pandas library. Then, city coordinates were created as a dictionary.

Package assignment is done in clustering.py and consists in assigning each package to a vehicle using k-nearest neighbors algorithm.

TSP optimization is based on launching a GA for each vehicle with assigned packages. After a fixed number of generations, the best route is the solution.

The last step was to output the results of the whole VRP problem: origin city, assigned packages, optimized route and total distance for each vehicle.

It is important to mention that using this methodology the following libraries had been used: pandas, numpy, sklearn and random.

# Experiments and Results

During this project, several experiments had been made in order to test the different methodologies used and the efficiency of the routes for each vehicle.

## Data

Before carrying out the experiments, in main.py were defined and load three data sets in order to simulate an environment for the resolution of the VRP problem.

The three csv, previously explained and defined in data folder, are loading using pandas library and printed to verify that they are correctly read.

TABLE I. TABLE PACKAGES

|  |  |  |  |
| --- | --- | --- | --- |
| **ID** | **Initial City** | **Destination City** | **Size** |
| 1 | Madrid | Sevilla | 5 |
| 2 | Valencia | Bilbao | 7 |
| 3 | Bilbao | Madrid | 3 |
| 4 | Sevilla | Barcelona | 4 |
| 5 | Barcelona | Valencia | 6 |
| 6 | Madrid | Bilbao | 2 |
| 7 | Valencia | Madrid | 8 |
| 8 | Bilbao | Barcelona | 1 |
| 9 | Barcelona | Sevilla | 9 |
| 10 | Sevilla | Valencia | 5 |

TABLE II. TABLE VEHICLES

|  |  |  |
| --- | --- | --- |
| **ID** | **Origin City** | **Capacity** |
| 1 | Madrid | 8 |
| 2 | Sevilla | 10 |
| 3 | Valencia | 20 |
| 4 | Bilbao | 5 |
| 5 | Barcelona | 10 |

TABLE III. TABLE CITIES

|  |  |  |
| --- | --- | --- |
| **Origin** | **Destination** | **Kilometres** |
| Barcelona | Bilbao | 610 |
| Barcelona | Madrid | 620 |
| Barcelona | Sevilla | 1000 |
| Barcelona | Valencia | 350 |
| Bilbao | Barcelona | 610 |
| Bilbao | Madrid | 400 |
| Bilbao | Sevilla | 850 |
| Bilbao | Valencia | 620 |
| Madrid | Barcelona | 620 |
| Madrid | Bilbao | 400 |
| Madrid | Sevilla | 530 |
| Madrid | Valencia | 360 |
| Sevilla | Barcelona | 1000 |
| Sevilla | Bilbao | 850 |
| Sevilla | Madrid | 530 |
| Sevilla | Valencia | 660 |
| Valencia | Barcelona | 350 |
| Valencia | Bilbao | 620 |
| Valencia | Madrid | 360 |
| Valencia | Sevilla | 660 |

## Package assigment

*clustering.py* uses kNN algorithm with k = 2 to assign each package to the closest vehicle (or the second closest if the first was not available), taking into account distance and capacity constraints.

First of all, assign\_packages() function (from clustering.py) is imported to main.py file. A message is displayed when executing: “Package assignment between vehicles considering origin city, package size and vehicle capacity:”. For each vehicle in the list, it obtains the id, origin city and shows a message with this two features.

At first, total load of the vehicle is set to 0. For each package, if it is assigned to that vehicle, it obtains data from package and shows a message with them: id, init city, destination city and size. Then it sums package size to the vehicle total load. At the end, it prints the actual load of the vehicle and its maximum capacity, as shown in the following table:

TABLE IV. TABLE PACKAGE ASSIGMENT

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **ID** | **City** | **Pkg** | **From** | **To** | **Size** | **L/C** |
| 1 | Madrid | 1 | Madrid | Sevilla | 5 | 7/8 |
| 1 | Madrid | 6 | Madrid | Bilbao | 2 | 7/8 |
| 2 | Sevilla | 4 | Sevilla | Barcelona | 4 | 9/10 |
| 2 | Sevilla | 10 | Sevilla | Valencia | 5 | 9/10 |
| 3 | Valencia | 2 | Valencia | Bilbao | 7 | 15/20 |
| 3 | Valencia | 7 | Valencia | Madrid | 8 | 15/20 |
| 4 | Bilbao | 3 | Bilbao | Madrid | 3 | 4/5 |
| 4 | Bilbao | 8 | Bilbao | Barcelona | 1 | 4/5 |
| 5 | Barcelona | 5 | Barcelona | Valencia | 6 | 6/10 |

## Route optimization

In *ga.py* it isdefined the *genetic\_algorithm()* function which allows to execute the GA to find the most efficient route. This function and the *class Problem\_Genetic* are imported to *main\_py.*

For each vehicle, the destination cities of the packages assigned are extracted. If there are no packages assigned, the vehicle has no route (a message is printed). If there is only one destination, the distance between that city and the origin one is calculated.

If there are more than one destination, fitness function is used to evaluate the sum of distances and validate the solution.

To execute the algorithm, an instance of the problem is created where genes are cities to visit, each individual is a permutation of cities and fitness function evaluate the total distance of the path.

Some parameters are used: population size, number of generations, crossover probability, mutation probability and tournament selection.

The result is printed an the output is the best route found for each vehicle and the total distance each one travels.

TABLE V. TABLE ROUTES

|  |  |  |
| --- | --- | --- |
| **Vehicle** | **Route** | **Distance (km)** |
| 1 | Madrid → Bilbao → Sevilla | 1250 |
| 2 | Sevilla → Valencia → Barcelona | 1010 |
| 3 | Valencia → Madrid → Bilbao | 760 |
| 4 | Bilbao → Madrid → Barcelona | 1020 |
| 5 | Barcelona → Valencia | 350 |

The following graph represents the result of applying GA to all of the vehicles, that is, the best route for each of them in terms of less kilometers travelled and all packages sent:

Gráfico, Gráfico de barras

El contenido generado por IA puede ser incorrecto.

Figure 1. Routes optimization

## Experiments

Several experiments have been done with different population size and mutation probability. The algorithm has been executed three times each time and their fitness functions are also shown as an average. The number of generations is always 100, the crossover probability is 0.8 and tournament size is equal to 3.

In the following table those results are recollected to analyze the difference results obtained:

TABLE VI. TABLE EXPERIMENTS

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Population | Mutation | Run1 | Run2 | Run3 | Promedio |
| 20 | 0.1 | 123.4 | 125.6 | 124.1 | 124.37 |
| 30 | 0.1 | 120.0 | 118.5 | 119.2 | 119.23 |
| 30 | 0.25 | 108.0 | 109.5 | 107.3 | 108.27 |
| 50 | 0.3 | 105.2 | 106.8 | 104.5 | 105.50 |

Analyzing this data, the following conclusions have been observed:

Increasing population size reduced fitness average, that is, results are better as there is more variety of individuals.

Increasing the mutation rate also helps to obtain more accurate results, which is translated to better routes. This improvement is due to the fact that mutation increases diversity in population so the probability of obtaining the best route also increases.

The best route obtained is the one with population equals to 50 and mutation probability 0.3 where the distance travelled is the smaller one.

A graph that shows those results has been plot so that it is easier to observe those conclusions previously mentioned:

Gráfico, Gráfico de barras

El contenido generado por IA puede ser incorrecto.

Figure 2. Comparison on average fitness for different experiments

Analyzing this bar graph it is clearly observed that the average fitness for the last experiment is much smaller than the one of the first experiment. Reading X axis and comparing it with Y axis, it is concluded that the higher the population and mutation rate is, the less the obtained fitness is.

# Conclusions

During the development of this work about the Vehicle Routing Problem (VRP), many of the contents learned throughout the Artificial Intelligence course have been deepened and reinforced. Not only using the libraries learned (pandas, NumPy, random, matplot), but also being able to experiment with them and learn more about their functions. The implementation of this whole project facilitated knowledge and skills which help when solving any other real-world problem.

This learning has not been the only thing that has been acquired from this practical assignment. Due to the amount of code programmed in order to obtain the solution of this problem, as well as the tests and experiments performed to verify its efficiency, the conclusions drawn in this report are numerous and diverse.

This report has presented a hybrid approach to solve the Vehicle Routing Problem by combining clustering with Genetic Algorithms (GA). The solution has been based on splitting the problem into several subproblems according to clustering and the launching several instances of TSP-solver, one for each vehicle. Using this approach has made it possible to obtain the final result, that is, the shortest route for each vehicle. The problem might seem long and complicated as it involves running an algorithm for a large number of packages, vehicles and cities. However, by dividing it this way, arriving at the solution has consisted on solving small samples. This shows that applying clustering before optimization helps to solve the problem in an easier, faster and more understandable way.

The use of kNN algorithm for clustering has also been an effective method for assigning packages to vehicles depending on their proximity and capacity constraints. Using n\_neighbors = 2 helped to obtain a more realistic solution making all packages assigned but without increasing the complexity of the algorithm.

The Travelling Salesman Problem has been solved by the implementation of the Genetic Algorithm for each vehicle. This ends on an efficient route planning. The search space was reduced and optimizing the problem was easier. On the other hand, using permutations of cities guaranteed obtaining efficient solutions and simplified the application of crossover and mutation.

Although the approach chosen was not the most complex one, it provided enough flexibility and realism for experimenting with the problem solution. By combining simple but powerful methods learned in course practices the final solution has been reached.

As it has been shown in III. Experiments an Results, after executing the algorithms, they had been proved under different methods.

Each of the vehicles was assigned to a route and then optimized, the total distance travelled by each vehicle changed depending on the different cities each of them visited. The results showed that GA algorithm has perfectly generated efficient routes for each vehicle.

Several experiments were run by changes in the population size and mutation rate. The conclusions obtained from the analysis of these experiments reflect the impact of population size and mutation on the execution of Genetic Algorithms. An increase in the diversity allows the algorithm to explore more possibilities, making it more difficult to stagnate in the search for the best. In addition, if the mutation rate is too low, the algorithm could stop improving, and if it is too high it could become less reliable. Using a value as 0.3 helps to find better routes and adds some variation.

As a conclusion, this assignment shows that combining different AI techniques can considerably help solve complex problems.

# Bibliography

The datasets used in this study, as well as the implementation of the tables in IEEE Conference format, have been generated using an IA tool (ChatGPT, OpenAI, 2025) to simulate realistic CSV for the experiments.

##### References

1. Gocken, T. & Yaktubay, M, “Comparison of different clustering algorithms via genetic algorithm for VRPTW”, 2019
2. Sam R. Thangiah, “Genetic Clustering: an adaptative heuristic for the multidepot vehicle routing problem”, 2001
3. A. Seshadri, “Vehicle routing problems 101”, 2019
4. A. Riscos Núñez, “Genetic algorithms for clustered VRP”, Artificial Intelligence , University of Seville, 2024/25
5. M. A. Gutiérrez Naranjo, F. J. Martín Mateos, A. Riscos Núñez, J. L. Ruiz Reina, J. M. Moyano, “Articial Intelligence – Unit 1: Inductive Machine Learning”, University of Seville, 2024/25
6. IBM. (s.f.). “¿Qué es el algoritmo de k vecinos más cercanos?”, Available: <https://www.ibm.com/mx-es/think/topics/knn>
7. F. Sancho Caparrini, “Algoritmos de Hormigas y el Problema del Viajante (TSP)”, Departamento de Ciencias de la Computación e Inteligencia Artificial, Universidad de Sevilla. [Online] Available: <https://www.cs.us.es/~fsancho/Blog/posts/ACO_TSP.md>
8. “Practice 1”, Artificial Intelligence, University of Seville, 2024/25
9. “Practice 3”, Artificial Intelligence, University of Seville, 2024/25
10. J. Ochelska-Mierzejewska, A. Poniszewska-Maranda, W. Maranda, “Selected genetic algorithms for vehicle routing problem solving”, 2021
11. C. Quesada, “VRPproblemAI”, GitHub repository, <https://github.com/carmenquesada/VRPproblemAI.git> , Accesed: May 27, 2025