The Vehicle Routing Problem (VRP)

By

Hugo Dufaure de Lajarte
Paul Duranteau
Thomas LE Xuan
Zakaria Midine
Farouk Mohamed
Zakaria Limi

March 26, 2024

Contents

Introduction

The aim of this project is to propose a solution to the problem of routing vehicles between different sites, better known as the VRP (Vehicle Routing Problem). This problem, modelled by a graph with nodes and routes, must take into account the multiplicity of possible routes as well as the constraints linked to the vehicle involved in our problem. It is currently considered to be an NP-hard problem. The objective is to determine the set of routes in order to estimate the cost and then minimise it. To do this, 3 metaheuristic methods will be explained in the rest of the report in order to give an initial approach to the problem. They will then be optimised using multi-agent systems and reinforced learning. The 3 algorithms are:

- the tabu algorithm;
- the simulated annealing;
- the genetic algorithm.

Keywords: Sequencing theory, metaheuristic algorithms, multi-agent systems, reinforcement learning.

1 *The vehicle routing problem with time windows*

1.1 Description

The vehicle routing problem (VRP) involves finding a set of routes, starting and ending at a depot, that together cover a set of customers. Each customer has a given demand, and no vehicle can service more customers than its capacity permits. The objective is to minimise the total distance traveled or the number of vehicles used, or a combination of these. In this project, we consider the vehicle routing problem with time windows (VRPTW), which is a generalisation of the VRP where the service at any customer starts within a given time interval, called a time window.

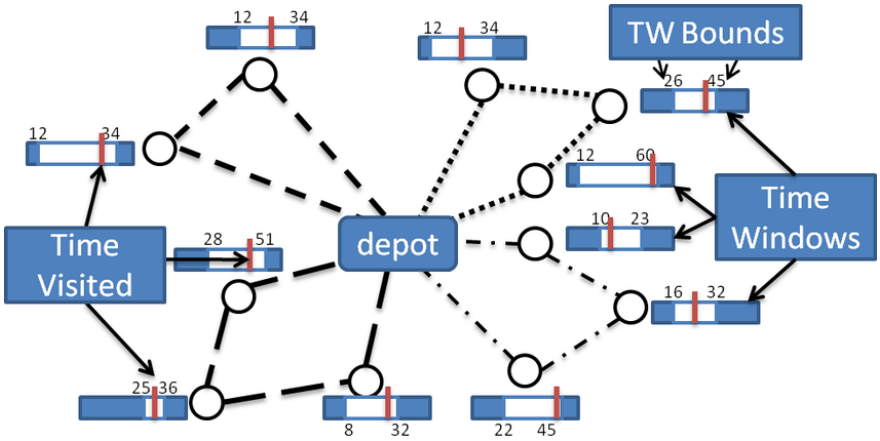


Figure 1: Illustration of the Vehicle Routing Problem with Time Windows (VRPTW)

This problem could be illustrated thanks to a graph including the clients and the depot who will be the knots of the graph and the routes that are connecting all the dots. For each clients, a time window has been given and has to be respected by the delivery vehicle. It is a major constraint that has to be taken in consideration regarding the solution that will be proposed further in this project.

1.2 Complexity

The complexity of the VRP is due to the combinatory nature of the problem which implies that a huge number of routing combinations for multiple vehicles and clients need to be taken into account. Consequently,

multiple possible combinations are to be be evaluated in order for the problem to be solved to optimality. In addition to that, the VRP is an NP-hard problem which means that, for the time being, there is no algorithm with a polynomial time complexity that solves it to optimality. Nevertheless, the use of multiple **optimisation methods** such as Tabu algorithm, Simulated Annealing algorithm and Genetic algorithm, making these collaborate thanks to the use of a **Multi Agent System** with these algorithms as its agents, and the improvement of neighborhood generation thanks to the use of **Reinforcement Learning** enables us to solve this problem efficiently.

1.3 State of art

The VRP has been first introduced in 1959 by G. B. Dantzig and J. H. Ramser. It was previously known as the "Truck Dispatching Problem" modelling the traffic of a fleet of trucks serving several gas stations in oil. Then a generalisation of this problem has been done by G. Clarke and J.R. Wright. This generalisation gave birth to the VRP problem that is commonly known in the field of Operations Research.

With evolution of the solutions for the case of VRP, many variants appeared such as: Capacitated VRP (CVRP), VRP with time window (VRPTW), VRP with heterogeneous fleet (VRPHF) among many.

Nowadays, time window variant is the most tackled among all these variants. Besides, current studies happen to take in count new constraints such as CO2 emission for instance.

Today, exact solutions of the problem are very limited. They are suited for only small scale problems (up to 200 customers approximately). That is why heuristic metaheuristic methods are introduced in order to give a solution at a higher scale (closer to real life). Those solutions are not exact but give solutions with a high yield and can handle any amount of customers. Today, with the evolution of the hardware, the optimisation of VRP has been crucial for several companies. Proper softwares have been developed and are used by various industrial companies, such as Coca-Cola.

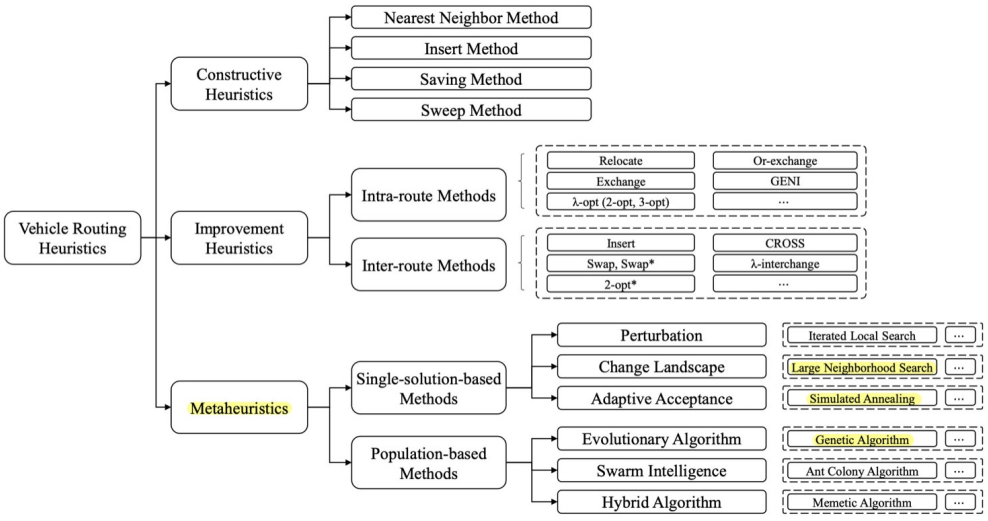


Figure 2: A classification of vehicle routing heuristics

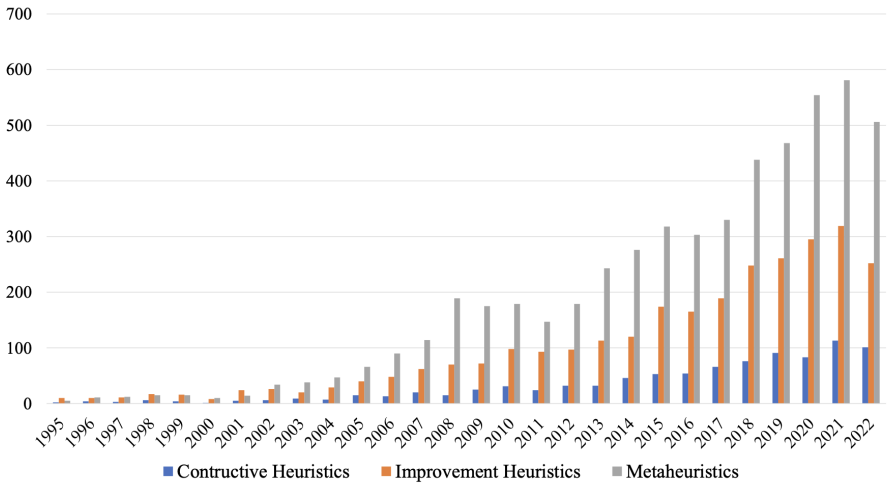


Figure 3: The number of publications on three classes of vehicle routing heuristics

Concerning the metaheuristics, they are considered today very efficients solutions to several hard optimisations problems thanks to their stability. Metaheuristics can be split into two categories : single-solution-based and populations-based methods.

Single-solution-based methods are also known as neighbourhood-based methods. They are designed according several rules such as multiple starts, changing the landscape or the acceptance rule. Tabu algorithm or Simulated Annealing are such algorithms.

Populations-based methods are inspired by natural rules and biological evolution. The solutions that are the most suited to the situation are kept and are put in competition in order to pull out the best solution according to the environment. The genetic algorithm is one the those methods.

These solutions are among the most popular solutions for the metaheuristic approach as illustrated on this graphic below.

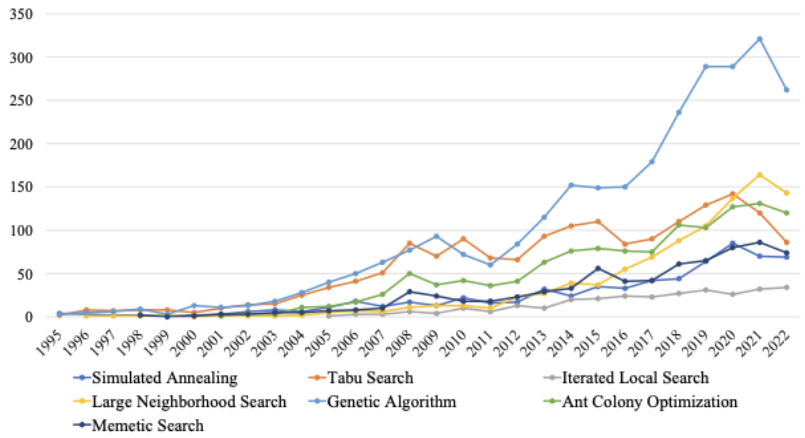


Figure 4: The number of publications of different metaheuristics in vehicle routing research

2 Exploring Excel Data for Vehicle Routing Optimization

This database consists of five Excel files (xls) that provide reference data for solving the Vehicle Routing Problem (VRP) with constraints. The data was obtained experimentally from the production environment of one of the largest distribution companies in Bosnia and Herzegovina.

1. **2_detail_table_customers.xls** - This file contains information about customers who need to be serviced during the delivery of ordered goods. Each customer is uniquely identified by the CUSTOMER_CODE field used for routing.
2. **3_detail_table_vehicles.xls** - This file contains all necessary information about the available fleet of vehicles. Each vehicle is uniquely identified by the VEHICLE_CODE field used for routing.
3. **4_detail_table_depots.xls** - This file contains all necessary information about the available depots on each route. Each depot is uniquely identified by the DEPOT_CODE field used for routing.
4. **5_detail_table_constraints_sdvrp.xls** - This file contains constraints for each of the 11 routes specifying which customers (identified by CUSTOMER_CODE) cannot be served by which vehicles (identified by VEHICLE_CODE). These constraints are specific to the Split Delivery Vehicle Routing Problem (SDVRP).
5. **6_detail_table_cust_depots_distances.xls** - This file contains actual distances (distance and time traveled) between a depot (identified by DEPOT_CODE) and each customer (identified by CUSTOMER_CODE). Thus, the DIRECTION column indicates whether the distance is from DEPOT to CUSTOMER (DEPOT $\longrightarrow$ CUSTOMER) or from CUSTOMER to DEPOT (CUSTOMER $\longrightarrow$ DEPOT).

3 *Optimisation using Metaheuristics*

As specified earlier, the purpose of this work is to optimise the sequencing of clients, the number of vehicles to send, and the time of delivery for each client(taking into account the specified time window for delivery). This can be modeled by the following integer linear model:

$$\min \quad f = \sum_{r \in Routes} \sum_{c=0}^{n_r - 1} distance[r[c], r[c+1]] \quad + Cost \cdot \sum_{r \in Routes} 1 + Penalty \cdot \sum_{r \in Routes} \sum_{c=0}^{n_r - 1} \mu[r[c]] \tag{1}$$

$$\text{s.t.} \quad \frac{1}{V_{mean}} \sum_{c=0}^{i-1} distance[r[c], r[c+1]] \quad \leq TW[r[i]] \qquad \forall r \in Routes \quad \forall i = 1..n_r, \tag{1a}$$

$$\mu[r[i]] \in \{0, 1\} \qquad \forall r \in Routes \quad \forall i = 1..n_r, \tag{1b}$$

$$\sum_{r \in Routes} 1 \leq 13 \qquad , \tag{1c}$$

$$Route[r][i] \in [\![1..NbClients]\!] \qquad \forall r \in Routes \quad \forall i = 1..n_r \tag{1d}$$

With:

- Decision variables:
  - Routes: list containing the routes and sequences of clients for each route;
  - $\mu[c]$: a binary variable indicating the delivery in the specified time window of client c if its value is 0, and otherwise if its value is 1;
- Data:
  - Distance: a matrix of NbClients+1 lines and NbClients+1 columns containing the distance between a client or depot to the another(it is a symmetric matrix);
  - Cost: a constant designating the mean cost of sending one vehicle to deliver items. This implies supposing that all vehicles have the same transport cost. Even though this helps simply the VRPTW problem, it would be interesting if one could go further with taking into account the variability of the costs for different vehicles depending on the driven distance for delivery.

– Penalty: as its name indicates, this constant serves as a penalty for our objective function f for each late delivery.

3.1 Tabu algorithm

The tabu search is one of the oldest metaheuristics. It was introduced by Glover, 1989, Fisher et al., 1997. At each iteration the neighbourhood of the current solution is explored and the best neighbor is selected as the new current solution. In order to allow the algorithm to escape from a local optimum the current solution is set to the best solution in the neighbourhood even if this solution is worse than the current solution. To prevent cycling visiting recently selected solutions is forbidden. This is implemented using a tabu list. Often, the tabu list does not contain illegal solutions, but forbidden moves. It makes sense to allow the tabu list to be overruled if this leads to an improvement of the current over-all best solution. Criteria such as this for overruling the tabu list are called aspiration criteria. The most used criteria for stopping a tabu search are a constant number of iterations without any improvement of the over-all best solution or a constant number of iteration in all. This can be illustrated by the following flowchart shown in figure 6:

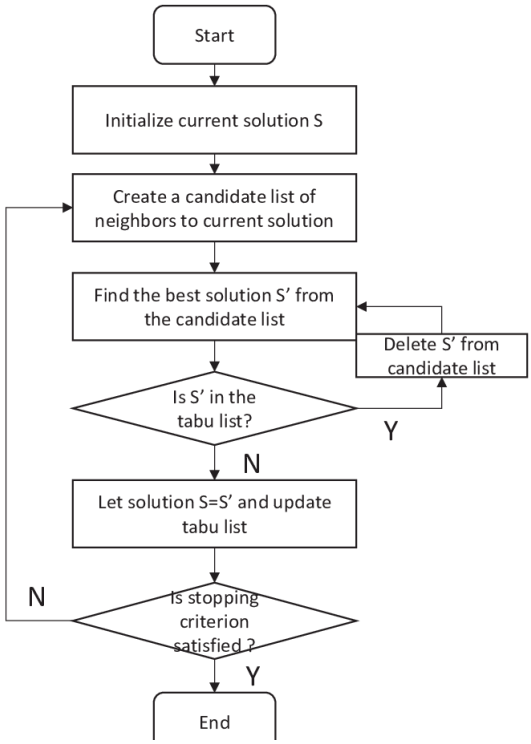
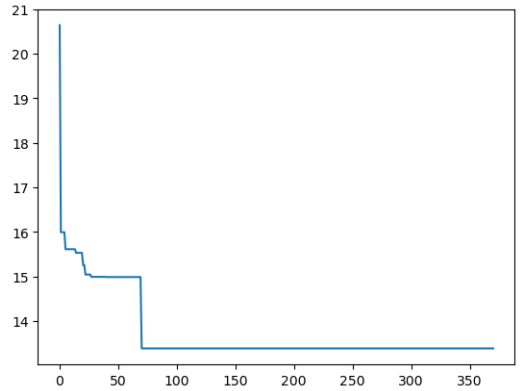


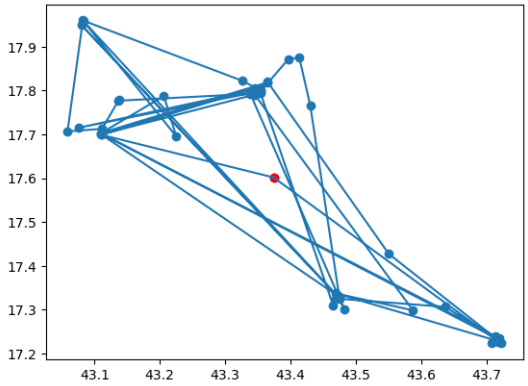
Figure 5: Flowchart Tabu algorithm

3.1.1 Specific VRPTW algorithm

Let's delve more into details of the Tabu algorithm for VRPTW problem. At first, we simplified this problem by considering it as a Traveling Salesman Problem. Specifically, we considered that only one vehicle will deliver the items and that there is no time window to consider. With these hypotheses, the Tabu algorithm gave solutions that aren't optimized, as shown in the curve and graph in figure 6. Stagnation occurred very fast and the algorithm got stuck at a local optimum for most of the iterations(300 iterations out of 350). It was later found that the issue resided in the neighborhood generation that consisted, at first, in randomly shuffling the sequence of clients for nb_neighbors times.



(a) Tabu for simplified VRP convergence curve



(b) Tabu for simplified VRP routes graph

Figure 6: Simplified VRP

The Tabu algorithm optimizing the solutions for a VRPTW has multiple parameters that have great impact on the time for a convergence to occure and the optimality of the final solution. These parameters are as follows:

1. The maximum number of iterations without finding a new best solution nbmax: This parameter serves as a stopping condition.

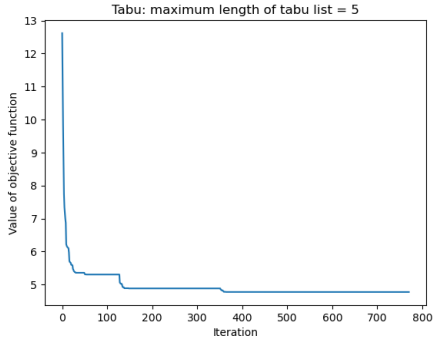
2. The limit function: This designates a value of our objective function that we would be satisfied with if obtained. This parameter also serves as a stopping condition in our algorithm.

3. The maximum size of the tabu list: this parameter indicates the size of the tabu list which, if attained, imposes the removal the oldest move or solution stored in it. Surprisingly, this parameter has an enormous impact on the optimized solution found by the algorithm, as shown in the following images in figure 7;
   From these tests, we conclude that the best solutions for the VRPTW problem are mostly found with big values of the maximum length of tabu list. However, setting a large value for this parameter could result in a decrease in algorithmic efficiency and would demand great computing time and/or power to find the desired optimized solution. That's why a compromise between efficiency and optimality needed to be made. We chose to set this parameter at the value 10 for the rest of this study.

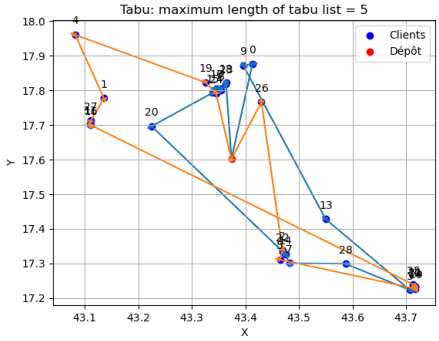
4. The number of neighbors to generate in each iteration: as explained in the flowchart of the algorithm, in each iteration we generate a specific number of neighbors for our solution. This neighborhood generation can be achieved with multiple methods:

   - Generate neighbors randomly: this methods doesn't give good results and will not enable the optimization of the solution.
   - Generate neighbors by swapping clients in a particular route: this method is much better than the previous one and helps the algorithm progress significantly in its optimization of a solution. However, the solutions found when using this method are not yet near optimality as there is no swapping between different routes(a client that has been initialised in a certain route with far away clients in it sequence is stuck with them). That's why we introduce the following method.
   - Generate neighbors by applying intra-route swapping and inter-route swapping: This way the neighborhoods generated for a certain solution help the tabu algorithm find a best solution that reaches the desired objective function.

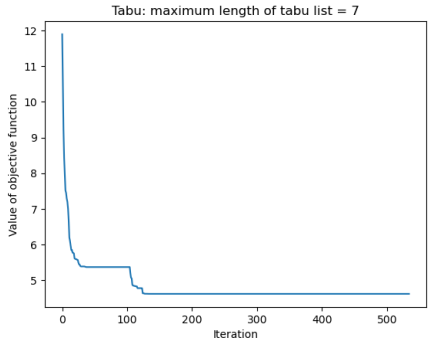
   The graphs and convergence curves shown in *figure 8* illustartes the fact that the higher the number of neighbors we set in our algorithm, the faster the convergence occurs and the better the final solution



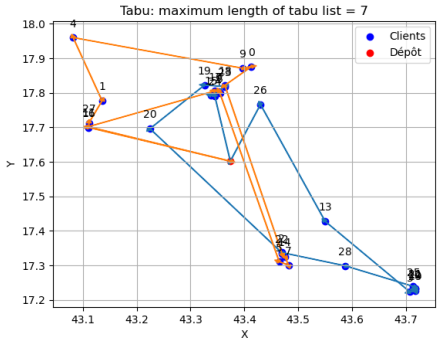
(a) Tabu maximum length of tabu list = 5 convergence curve



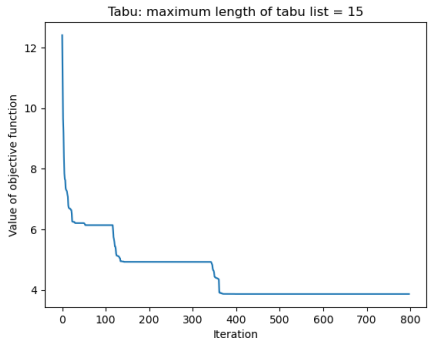
(b) Tabu maximum length of tabu list = 5 routes graph



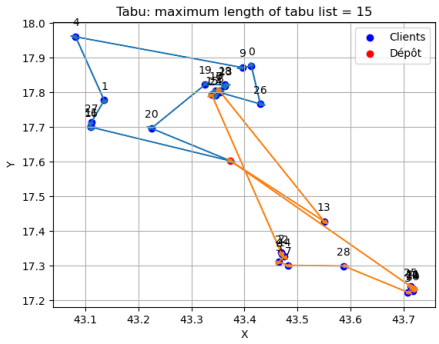
(c) Tabu maximum length of tabu list = 7 convergence curve



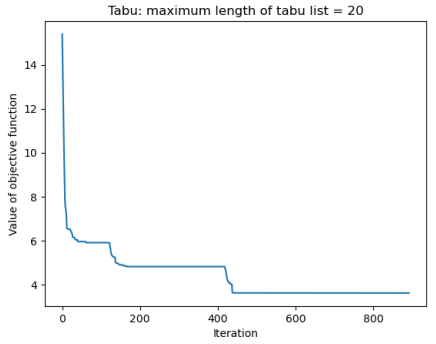
(d) Tabu maximum length of tabu list = 7 routes graph



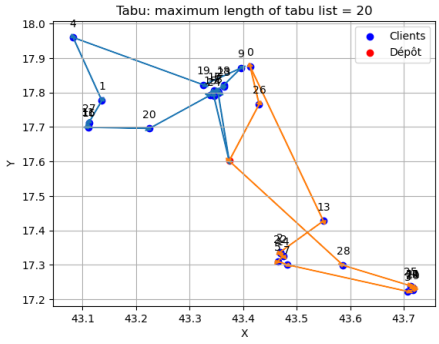
(e) Tabu maximum length of tabu list = 15 convergence curve



(f) Tabu maximum length of tabu list = 15 routes graph



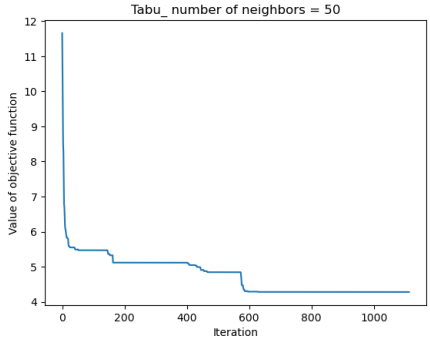
(g) Tabu maximum length of tabu list = 20 convergence curve



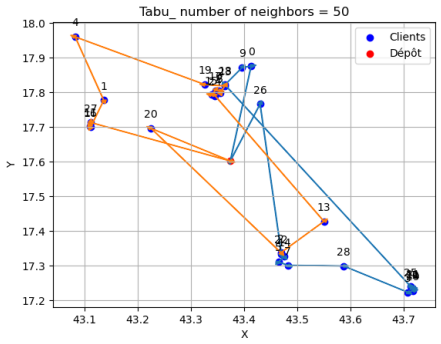
(h) Tabu maximum length of tabu list = 7 routes graph

Figure 7: Impact of the tabu list's maximum size on the algorithm

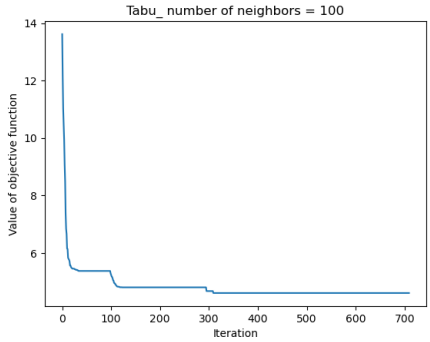
is. However, higher neighbors to generate means higher computational cost of the algorithm resulting in less efficiency. That's why, same as with the maximum length of tabu list parameter, a compromise need to be done. We chose to work with a number of neighbors equal to 100 for the rest of the study.



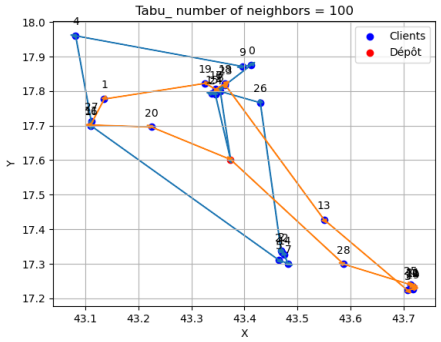
(a) Tabu number of neighbors = 50 convergence curve



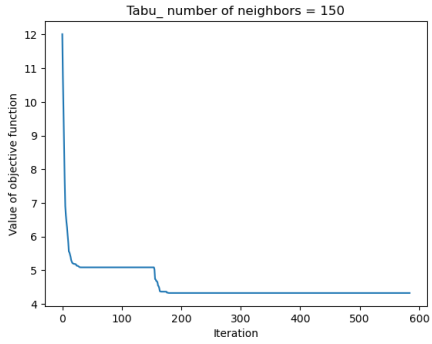
(b) Tabu number of neighbors = 50 routes graph



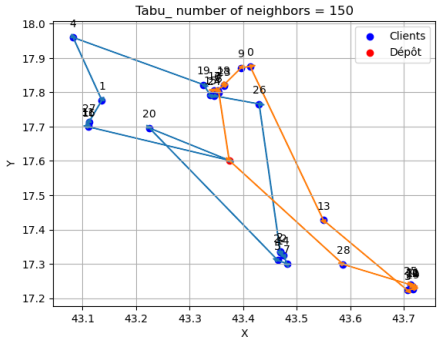
(c) Tabu number of neighbors = 100 convergence curve



(d) Tabu number of neighbors = 100 routes graph



(e) Tabu number of neighbors = 150 convergence curve



(f) Tabu number of neighbors = 150 routes graph

Figure 8: Impact of the number of neighbors on the algorithm

3.1.2 Results analysis

Using the values of the parameters mentioned earlier, we adapted our algorithm to optimize solutions of a number of routes determined by the algorithm itself, not imposed as a parameter, in order for us to take into consideration all 13 possible vehicles. After executing our algorithm multiple times(for 20 iterations), we found the following distribution of the number of best solutions per number of routes/vehicles used:

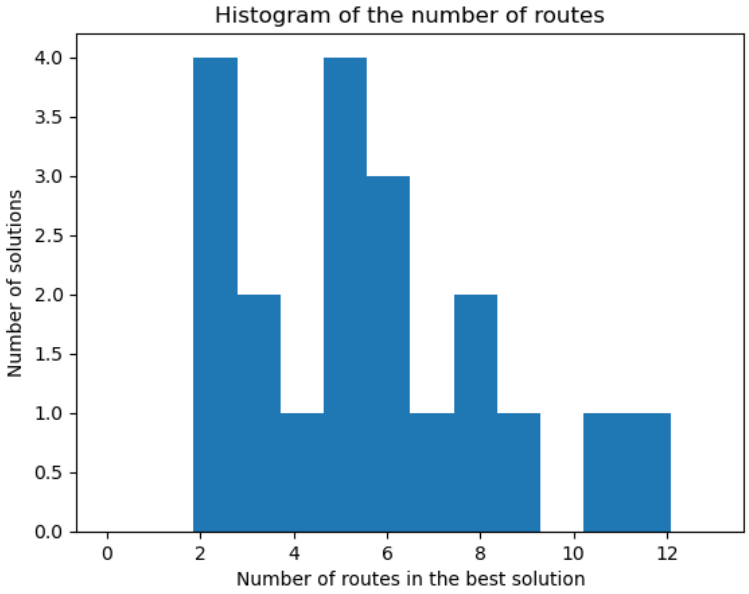
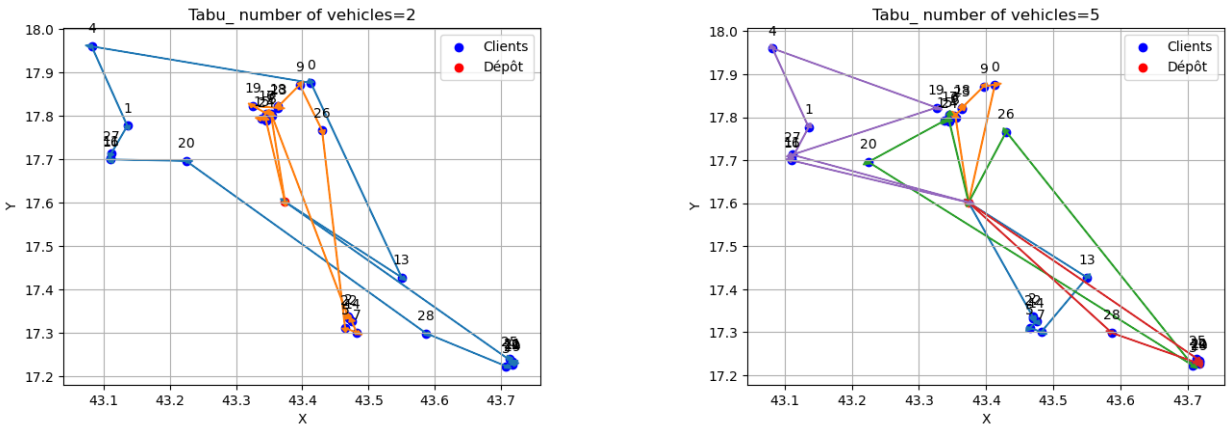


Figure 9: Histogram of the number of best solutions per number of routes

We notice that 2 and 5 are the numbers of vehicles used that are most occurring which means that it's most probable to have an optimized solution satisfying our expectations(good objective function) with 2 or 5 as parameter imposing he number of vehicles to use than with another number of vehicles.

Tuning the different parameters of the Tabu algorithm helped us optimize not only the solution but also the efficiency of the model. We noticed that the greater the number of neighbors to generate and maximum length of the tabu list, the faster stagnation occurs. However, this also results in longer periods of computation for an optimal solution to be found. The graphs given for 2 and 5 vehicles used are as shown in figure 10.



(a) Objective function = 4.8638

(b) Objective function = 6.0742

Figure 10: Graphs for routes when two vehicles are sent vs when 5 vehicles are sent

3.2 Simulated annealing

The simulated annealing algorithm is inspired by a physical analogy in the solids annealing process. It is then applied to solve complex optimization problems, such as Travelling Salesman Problem (TSP) (Kirkpatrick

et al.,1983). The annealing process in the manufacture of crystal is the cooling process of a solid object until the structure is frozen at the minimum energy (Bersimas and Tsitsiklis, 1993). The manufacture of crystal does the heating to a certain point; when the material is hot, the atoms will move freely with a high energy level. And then, the temperature will be slowly dropped until the particles are in the optimum position with minimum energy. Asimulated annealing algorithm can be viewed as a local search algorithm that sometimes moves towards a solution with a higher cost or not optimal solution. This movement can be put out from local minimum (Bersimas and Tsitsiklis, 1993). The simulated annealing algorithm starts with determining the initial solution that considers the current solution and the initial temperature. The initial solutions are built around several viable solution neighbourhoods derived from randomly rearranging existing solutions. According to Tospornsampan et al. (2007), three components must be considered when applying the simulated annealing algorithm: the annealing process or the cooling schedule, making rearrangement or neighbourhood, and termination algorithm.

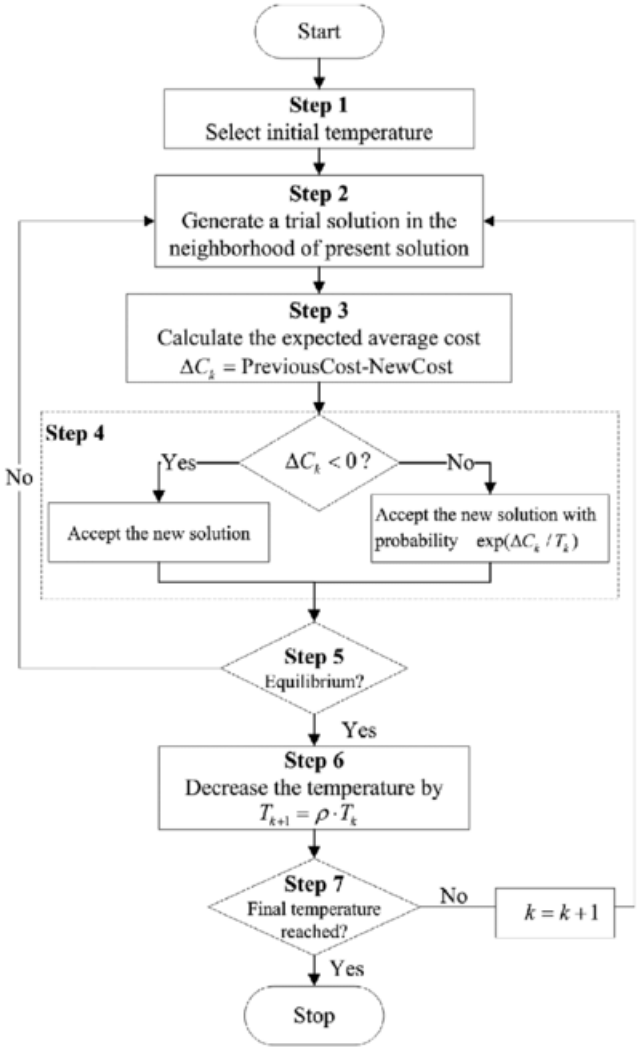


Figure 11: Flowchart of simulated annealing

3.2.1 Specific VRPTW algorithm

The VRPTW can be defined as follows: there are a number of customers C, several vehicles V, and a depot d. Each customer has units of demand, and the vehicles have a maximum capacity to be carried. The aim of the problem is finding a set of routes starting and ending at d. Each customer in C must be visited by exactly one vehicle in V. An amount of time is required by the vehicle on the road and during servicing the customer.

An additional constraint called time window forces the vehicle to service the customers within their available time windows. The vehicle must wait if it arrives earlier than the opening time of the customer. However, late arrival of the vehicle is not permitted. Various objective functions have been addressed in the existing studies. However, most of the studies consider reducing the number of vehicles and the total travel distance of the vehicles simultaneously.

3.2.2 Development of Simulated Annealing for the VRPTW

As a generic probilistic method, simulated annealing has the capability to obtain near-optimum solutions for complex problems with large search spaces. The method has been successfully implemented for various combinatorial problems.

Simulated annealing is inspired by the physical process of annealing, where a material such as steel or glass is heated and then cooled. By simulating this physical process, each step of the simulated annealing algorithm replaces the current solution with a neighbor solution. The neighbor solution is randomly generated using functions or operators that are developed specifically according to the problem.

If the new (neighbor) solution improves the objective function, then it is accepted, and the new solution is preserved. Otherwise, the new solution is accepted with a probability that depends both on the difference between the objective function values and on a global parameter T (called the temperature), which is gradually decreased during the process. This mechanism causes simulated annealing to almost randomly change the current solution when T is large in earlier iterations. However, as T is decreased, the current solution is gradually driven towards the optimum area.

The allowance for accepting a worse solution enables simulated annealing to avoid local optimum areas. This feature distinguishes simulated annealing from simple hill-climbing search methods that accept only better solutions in each iteration.

Simulated annealing requires three main processes to solve the VRPTW:

- Generating a random initial solution.
- Producing a new solution using special neighbor operators.
- Applying a function to reduce the probability of accepting
- a new worse solution, which is useful for escaping local optima which is useful in escaping local optimums during the course of the search for the global optimum.

```
BEGIN Simulated Annealing
Let t = temp0
Generate random initial solution S as
current solution
WHILE t≥temp1
   REPEAT inner_itrtimes
     Let d1=value of the objective
        function (distance) of S
     Produce Sn (neighbor of S) as a
        new solution
     Let d2=value of the objective
        function (distance) of Sn
     IF d2<d1 THEN
accept change (S← Sn)
       Let Sb← Snew
     ELSE IF d2>d1 THEN
accept change with probability
                prob = 1 / e^((d2−d1)k / d1t)
     END IF
   END REPEAT
   Let t←t * cooling_factor
END WHILE
Return Sb as the best solution
END Simulated Annealing
```

Figure 12: Pseudo-code of Simulated Annealing

The parameters for the proposed simulated annealing heuristic are set as follows:

- The initial temperature temp_0 is set to 0.8,
- The cooling factor is set to 0.995,
- The final temperature temp_1 is set to 0.01,
- The number of iterations per fixed temperature (*inner_iteration*) is set to 6000.

3.2.3 Charts

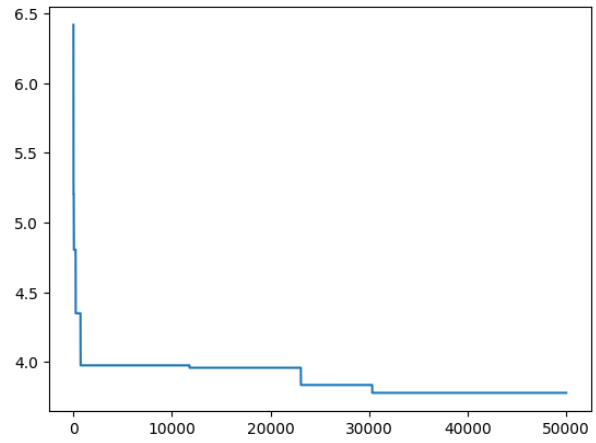


Figure 13: Cost evaluation in the Simulated Annealing method for a small-sized database

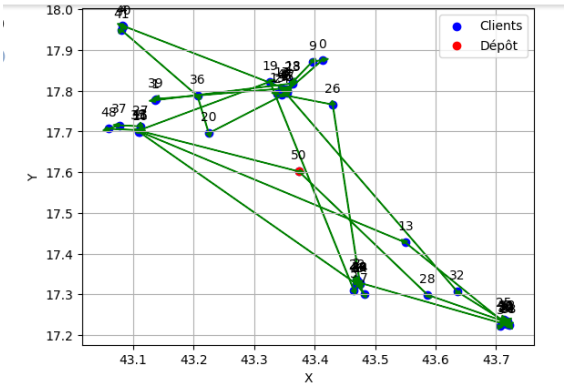


Figure 14: Flowchart of the Simulated Annealing

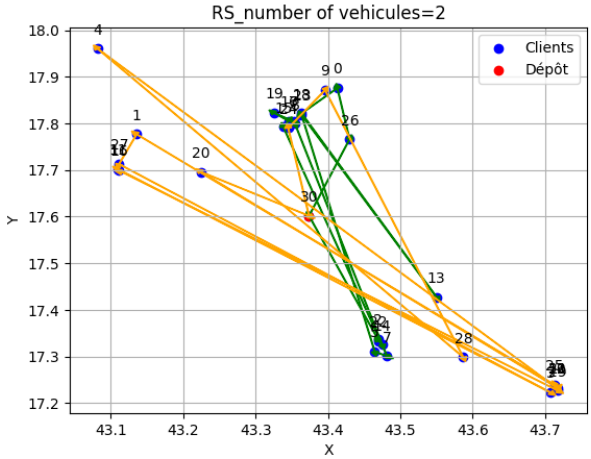


Figure 15: Flowchart of the Simulated Annealing with 2 Vehicle

3.2.4 Results analysis

Simulated annealing can perform well on small data sets where it can efficiently explore the solution space and converge to a near-optimal solution. However, when dealing with large data sets, simulated annealing may face challenges such as slower convergence and difficulty in exploring the vast solution space efficiently.

Using the values of the parameters mentioned earlier, we adapted our algorithm to optimize solutions of a number of routes determined by the algorithm itself, not imposed as a parameter, in order for us to take into consideration all 13 possible vehicles. After executing our algorithm multiple times(for 27 iterations), we found the following distribution of the number of best solutions per number of routes/vehicles used:

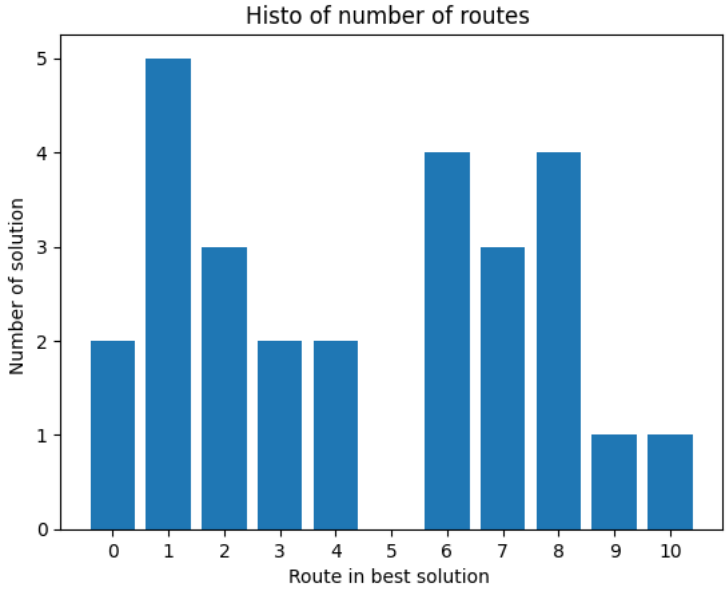


Figure 16: Histogram of the number of best solutions per number of routes

Overall , we had to consider the following parameters :

- Initial Solution: The initial solution or starting point from which the optimization algorithm begins its search. A good initial solution can potentially lead to quicker convergence.

- Temperature Schedule: In simulated annealing, the temperature schedule determines how the temperature decreases over time. It controls the probability of accepting worse solutions as the algorithm progresses. Choosing an appropriate cooling schedule is essential for balancing exploration and exploitation.

3.3 Genetic algorithm

3.3.1 Specific VRPTW algorithm

Genetic algorithms have been inspired by the natural selection mechanism introduced by Darwin. They apply certain operators to a population os solutions of the problem at hand, in such a way that the new population is improved compared with the previous one according to a pre-specified criterion function. This procedure is applied for a pre-selected number of iterations and the output of the algorithm is the best solution found in the last population or, in some cases, the best solution found during the evolution of the algorithm. In general, the solutions of the problem at hand are coded and the operators are applied to the coded versions of the solutions. The way the solutions are coded plays an important role in the performance of a genetic algorithm. Inappropriate coding may lead to poor performance. The operators used by genetic algorithms simulate the way natural selection is carried out. The most well-known operators used are the reproduction, crossover, and mutation operators applied in that order to the current population. The reproduction operator ensure that, in probability, the better a solution in the current population is, the more (less) replicates it has in the next population. The crossover operator, which is applied to the temporary population produced after the application of the reproduction operator, selects pairs of solutions randomly, splits them at a random position, and exchanges their second parts. Finally, the mutation operator, which is applied after the application of the reproduction and crossover operators, selects randomly an element of a solution and alters it with some probability. Hence genetic algorithms provide a search technique used in computing to find true or approximate solutions to optimisation and search problems.

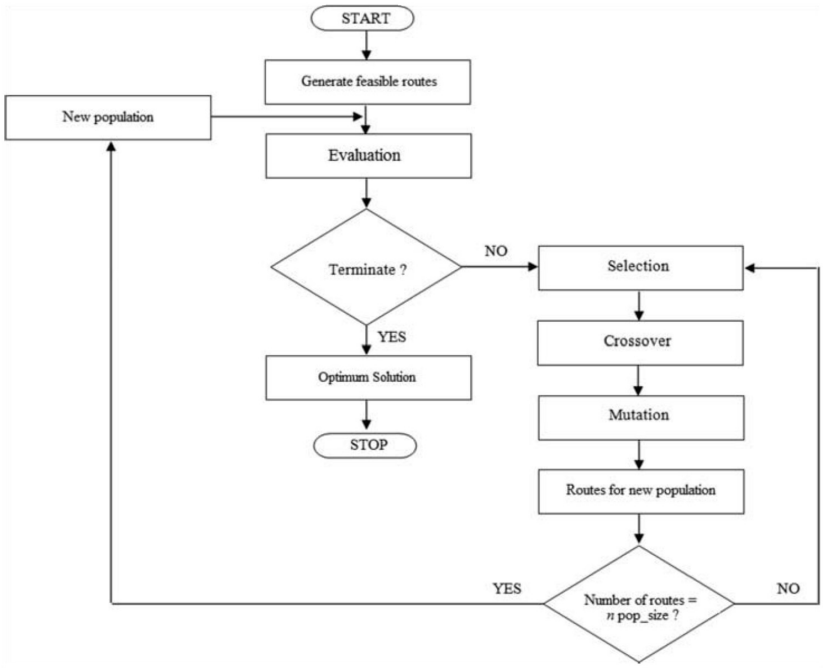


Figure 17: Flowchart of the genetic algorithm

3.3.2 Charts

Let's delve more into details of the Genetic algorithm for VRPTW problem. At first, like we did with the Tabu algorithm we simplified this problem by considering it as a Traveling Salesman Problem (TSP). Specifically, we considered that only one vehicle will deliver the items and that there is no time window to consider. This will be the case for this section. We will consider multiple vehicles in the MAS section. With these hypotheses, the Genetic algorithm gave solutions that are almost optimized for the TSP, as shown in the curve and graph in figure 18.

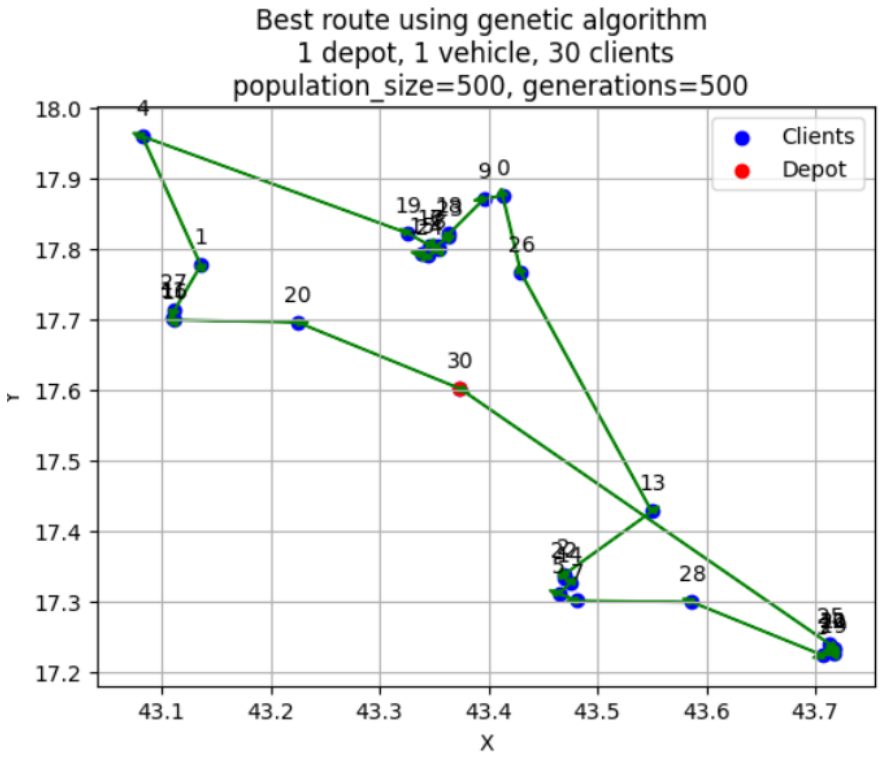


Figure 18: Flowchart of the genetic algorithm

3.3.3 Results analysis

The genetic algorithm performs quite well on a small database. There are still some locations that could be swapped for the route.

Overall, we had to consider the following parameters :

1. The population size: this parameter has a big impact on the convergence of the algorithm towards an optimal solution. We tested different values for this parameter and found the curves in figure 19. This shows that the greater the value of population size parameter, the less stagnation occurs and the faster optimality is reached.

2. The number of generations: this parameter consists of the number of iterations the algorithms will execute. At each iteration it creates a new generation thanks to crossbreeding and mutation. This parameter has no great impact on the algorithm once we pinpoint the number of iterations that enable the algorithm to reach stagnation. The tests showed that for 350 iterations(when tested with 30 clients) is sufficient and the algorithm will remain stuck for iterations that follow.

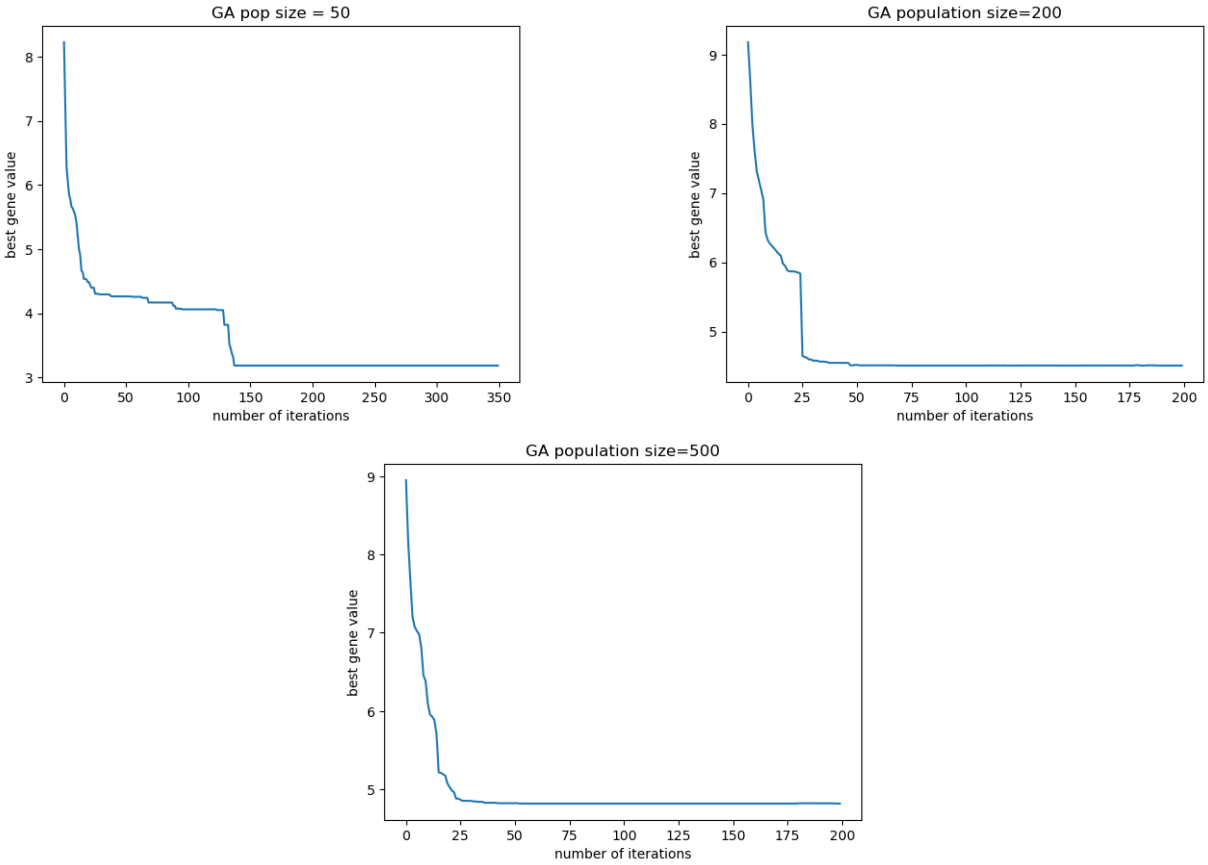


Figure 19: Impact of population size parameter on the convergence curve

3. The mutation rate: this parameter is of great importance in the genetic algorithm as it tells the algorithm when to generate a mutated gene(a solution). Testing with different values of this parameter give the convergence curves in figure 20.

4 *Collaborative Optimisation: Metaheuristics with MAS*

In this section, we study the concept of Multi-Agent Systems and how we applied it to the VRPTW problem.

Definition: A MAS(Multi-Agent System) is a system that defines an environment containing multiple agents specifying the rules of interaction between them.

These types of interactions are defined in the MAS thanks to a class method called `contact`. This class method tells the collaborative agent to exchange data with other agents in case their optimized solutions at time step t is of better objective function. In our case, we modeled a MAS with agents as metaheuristic algorithms. In order for us to do this, we use the package `mesa` for Python programming. An MAS is defined with a model class and one or multiple agent classes in Python using this package.

- **Agent class:** in this class we define the `step` method that contains the metaheuristic algorithm and the method `contact` that we mentioned above.
- **MAS model:** In this class we initialize the desired agents in our MAS environment, we choose the type of `scheduler` (the method of activity scheduling for different agents, in our case we worked with `SimultaneousActivation` as it is more beneficial to execute the algorithms simultaneously in each

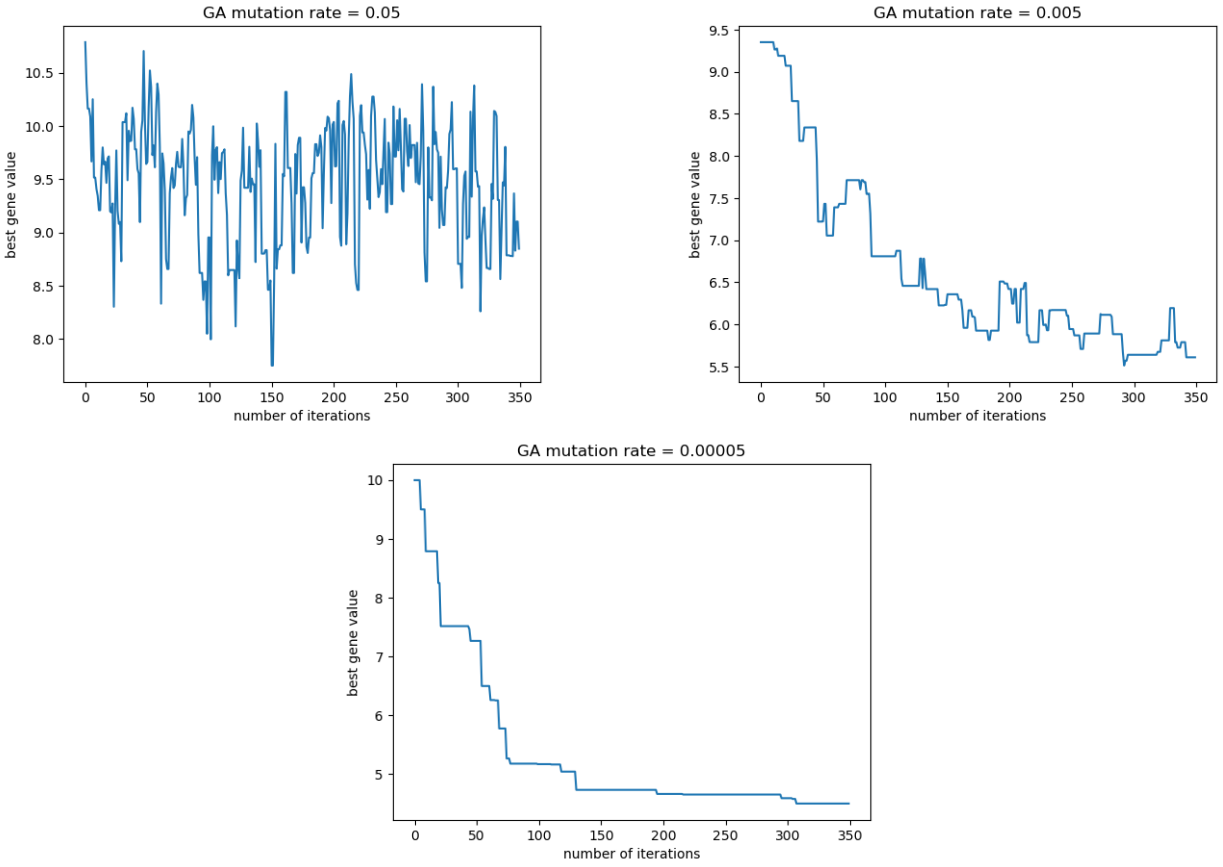


Figure 20: impact of mutation rate

time step) and we collect interesting data that would help us visualise the convergence curve and/or the optimized routes' graphs.

The interactions between agents are of two types:

- **Collaborative:** this means that an agent is able to access the best solutions found by the other agents at time t and update its own if it finds a solution better than what it currently has.
- **Non-collaborative:** this means that an agent doesn't consider the solutions of the other agents and only relies on itself.

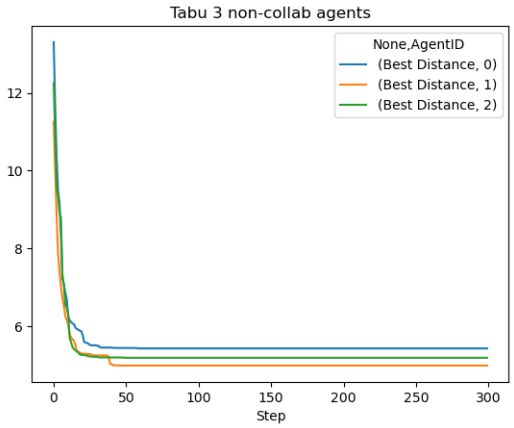
4.1 Tabu algorithm with MAS

4.1.1 Specific VRPTW algorithm

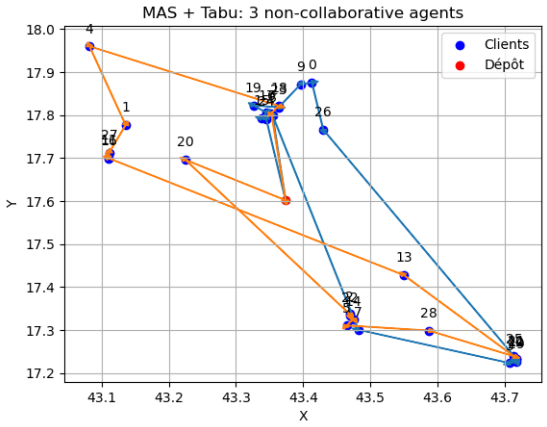
Integrating the Tabu algorithm in a Multi-Agent System consists of creating an agent class with its `step` method composed of the same code as the body of the while loop in our tabu algorithm. This could be interpreted by the fact that executing our MAS algorithm for N steps is the same as a tabu algorithm being run with N as its maximum_number_of_iterations_without_improvement parameter.

4.1.2 Charts

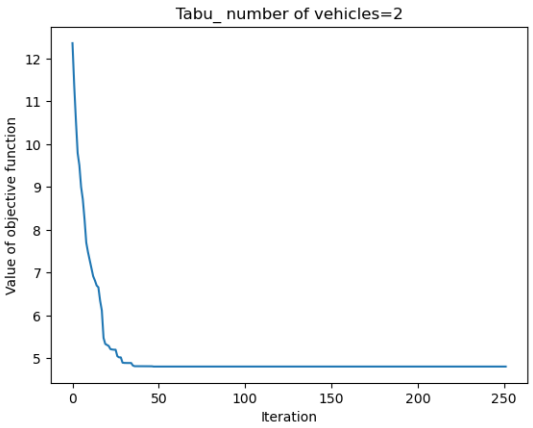
Thanks to the `DataCollector` provided by mesa, we were able to use important data to compare between tabu algorithm with and without MAS as shown in the figure below:



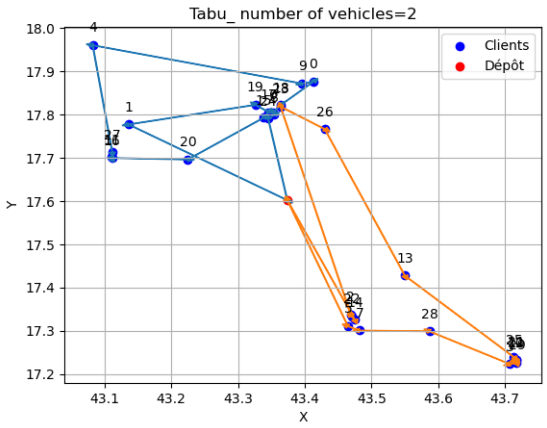
(a) Tabu+MAS convergence curve



(b) Tabu+MAS graph



(c) Tabu convergence curve

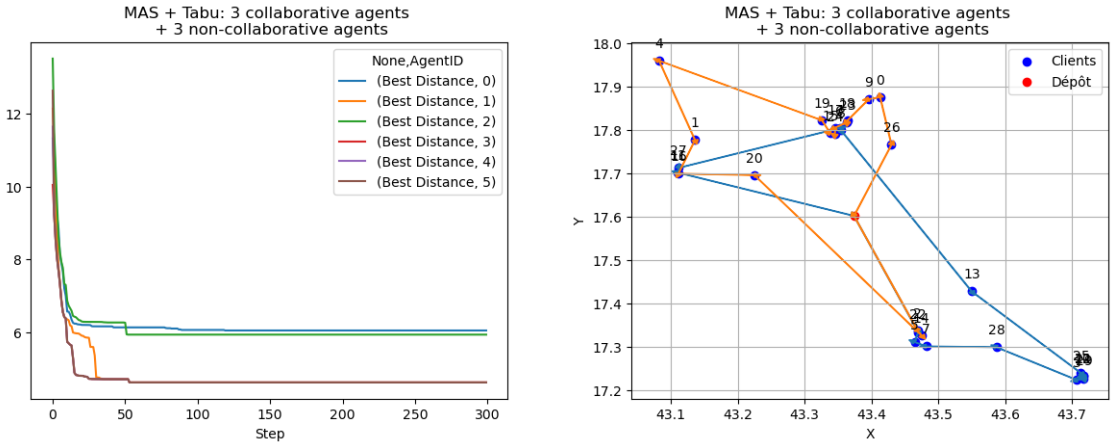


(d) Tabu graph

Figure 21: Comparison between tabu with MAS of 3 non-collaborative agents and tabu algorithm

From figure 21 we can see that the tabu agents in our MAS have the same behaviour as a tabu algorithm without MAS when they don't collaborate. They reach stagnation very fast(premature convergence), their optimized objective functions are of the same scale as that of tabu algorithm without MAS and there are multiple instances where a the optimized sequence suggests leaving a 'neighborhood' of clients and going back to it later on which isn't reasonable.

Now, let us compare between collaborative and non-collaborative agents. For that, we create an agent class attribute called `collaborative` enabling the MAS model to distinguish between agents that need to take data from other agents that found better solutions and optimize them, and others that need to rely only on their algorithms. Next, we define the rules of interaction between agents thanks to the class method `contact`. This class method takes the best agent(with most optimal objective function at time t) and tests if the current tabu agent has a less optimal solution. In this case, the tabu agent stores its neighbor agent's best solution as his own and applies its algorithm to optimize it. Launching the MAS with 3-collaborative tabu agents and 3-non collaborative ones gives the following convergence curves.



(a) Convergence curve

(b) Route graph

Figure 22: MAS+Tabu 3 collaborative agents vs 3 non collaborative agents

4.1.3 Results analysis

Thanks to the convergence curve in figure 22(a), we notice that the objective function's optimal value found by collaborative agents is much better than that found by non-collaborative agents/ tabu algorithm without MAS. As for the route suggested by MAS +Tabu, it's clear that it's much closer to optimality than tabu alone or MAS with non-collaborative agents: there's less crossings between client neighborhoods even though we can see that two vehicles visit the same client neighborhood which increases the delivery costs.

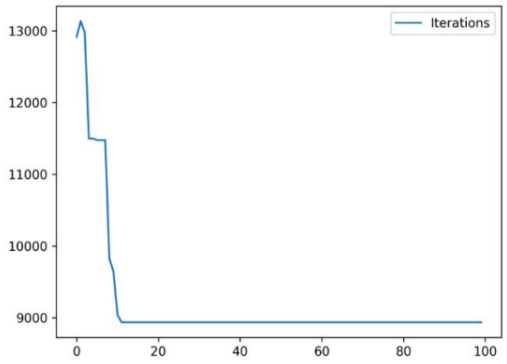
4.2 Simulated annealing with MAS

4.2.1 Specific VRPTW algorithm

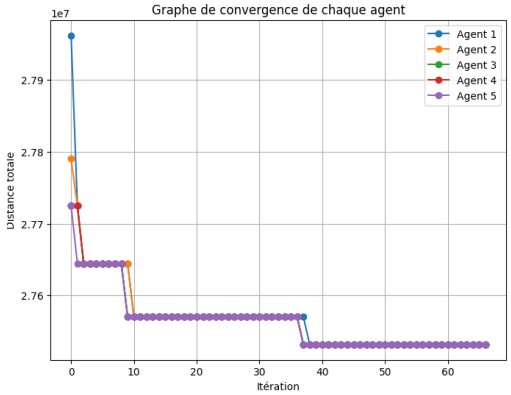
Incorporating the Simulated Annealing algorithm into a Multi-Agent System entails crafting agent modules equipped with a step function mirroring the iterative process of Simulated Annealing. Executing the MAS algorithm for a designated number of steps parallels the operation of Simulated Annealing with an equivalent maximum iteration count. This convergence highlights the flexibility of MAS frameworks in accommodating various optimization methodologies, showcasing the versatility of Simulated Annealing within such systems.

4.2.2 Charts

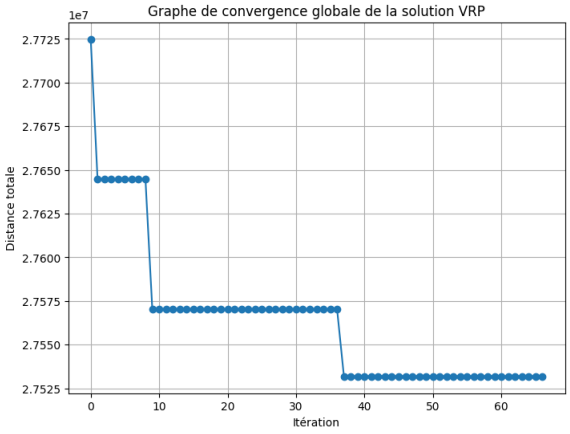
The incorporation of Simulated Annealing with Multi-Agent Systems (MAS) introduces an intriguing fusion of optimization techniques within decentralized environments. Utilizing charts to depict the performance of this amalgamation offers a visual narrative of the convergence patterns of individual agents over time. These charts serve as powerful tools for tracking the optimization progress and illustrating the collaborative dynamics within the MAS framework. However, it's essential to recognize that while charting enhances our understanding of the convergence process, the computational costs associated with this combined approach must be carefully considered. Nonetheless, the visual representation provided by these charts offers valuable insights into the efficacy and dynamics of Simulated Annealing integrated with MAS, paving the way for further exploration and refinement in decentralized optimization paradigms.



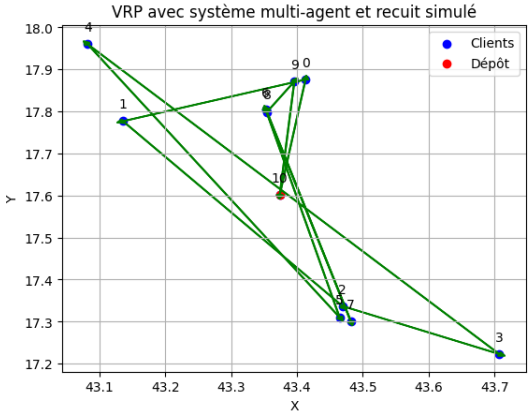
(a) Cost evaluation using the Simulated Annealing algorithm



(b) Convergence of Agent



(c) convergence curve



(d) 1 Vehicle Routing Problem (VRP) with Multi-Agent System and Simulated Annealing

4.2.3 Results analysis

Our investigation into solving the Vehicle Routing Problem (VRP) compared two approaches: simulated annealing and a multi-agent system with simulated annealing. The results suggest that the multi-agent system with simulated annealing outperforms the standalone simulated annealing algorithm in finding lower cost solutions.

This finding indicates that incorporating multiple agents collaborating through simulated annealing leads to a more efficient search process for optimal VRP solutions, potentially reducing transportation costs.

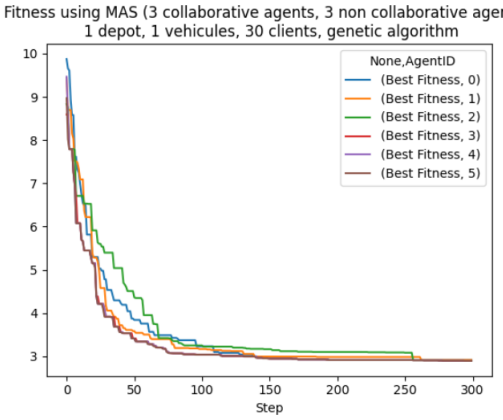
4.3 Genetic algorithm with MAS

4.3.1 Specific VRPTW algorithm

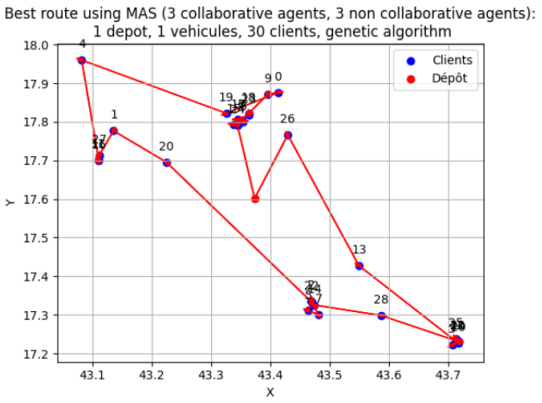
As for the genetic algorithm, we create MAS genetic algorithm agents with a step method that executes the body of the for loop in the genetic algorithm. This means that for each time step done executed by the Multi Agent System, our agents create a new population thanks to the crossbreeding and mutation phenomenons applied to members of the initial population at time t.

4.3.2 Charts

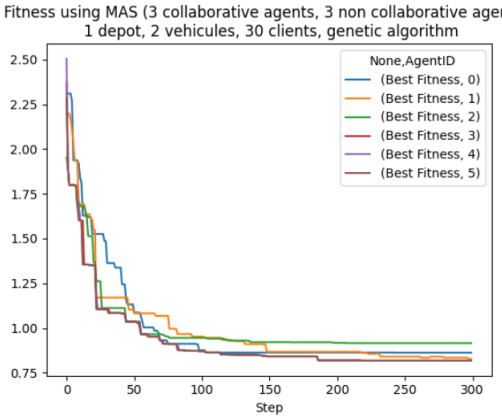
We launched the MAS for different numbers of vehicles for 3 collaborative agents and 3 non-collaborative ones. The results found are as follows:



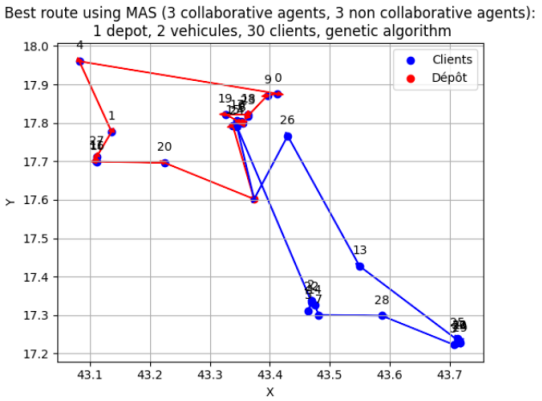
(a) Fitness evolution



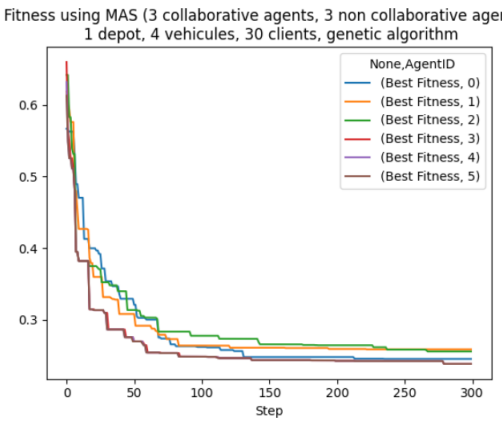
(b) Best route



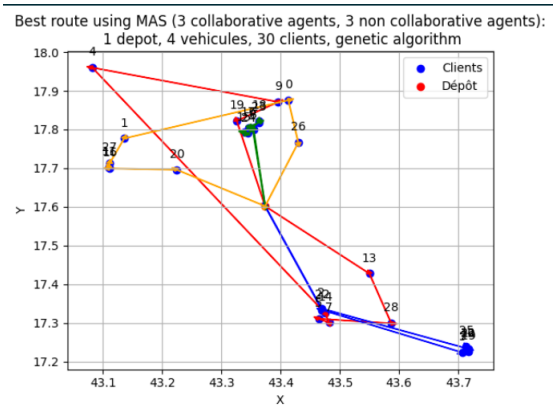
(c) Fitness evolution



(d) Best route



(e) Fitness evolution



(f) Best route

Figure 24: Genetic algorithm with MAS

4.3.3 Results analysis

We noticed that for one vehicle the collaborative agents converge (reach optimality) faster than non-collaborative agents. However, all of the agents obtain final solutions that are of the same magnitude as shown on figure 24 (a). However, when we increase the number of vehicles wanted(we impose the number of vehicles to use), we notice that the collaborative agents converge to a solution that is noticeably of a better objective function value than those found by non-collaborative agents. The difference in convergence between the two types of agents gets more and more visible as we increased the number of vehicles.

5 *Collaborative Optimisation: Metaheuristics with ADMM and Reinforcement Learning*

In the three metaheuristic algorithms studied above, there are functions that generate neighbors of a current solution in the algorithm(for tabu and simulated annealing) and functions that execute mutations(for genetic algorithm). The purpose of this section is to improve these functions using a technique called Reinforcement Learning.
Reinforcement Learning is a technique based on the notion of reward and punishment mechanisms. It consists of executing multiple actions and learning from it mistakes and successed thanks to the concept of reward. In the case of VRP, the reward is either $-Objective\_function$ or $\frac{1}{Objective\_function}$. This reward takes place following a set of actions/transitions from one state to another.

The states that we considered are $S \in$ {InterRouteShif ; IntraRouteShift ; InterRouteSwap ; IntraRouteSwap ; TwoIntraRouteSwap ; TwoIntraRouteShift}:

1. Intra-Route Swap: swapping a customer with another client in the same route
2. Inter-Route Swap: Neighborhood Feature that performs the exchange move of a customer of a route with a customer from another road.
3. Intra-Route Shift: Neighborhood function that performs moving a customer to another position on the same road.
4. Inter-Route Shift: Neighborhood function that performs moving a customer from one route to another.
5. Two Intra-Route Swap: neighborhood function that consists of the exchange of customers on the same route, as well as the intra-route exchange neighborhood function. However, in the Two Intra-Route Swap function, two consecutive customers are exchanged with two others consecutive customers on the same route
6. Two Intra -Route Shift: Neighborhood function which consists of the relocation of customers on the same road, as well as the neighborhood function intra shift -road. However, in the function Two Intra -Route Shift, two consecutive customers are removed from their position and reinserted into another position on the same road;

As for the actions, they consist of the transition from one neighbor generation method to another. For example, if the current state is intra-route-swapping, there are 6 actions that can be taken: either remain in the same state, or go to one of the other five methods.
There are multiple algorithms for reinforcement learning. In this study, we used the **Q-learning** algorithm.

5.1 Tabu algorithm with MAS and Reinforcement Learning

5.1.1 Specific VRPTW algorithm

The purpose of integrating the RL in our algorithms being the improvement of neighbor generation, we developed the following Q-learning algorithm for tabu algorithm's neighborhood generation:

Procedure AdaptiveSearch_Qlearning(solution):
initialiser Q;
improved ← 0;
no_improvement ← 0;
x_best ← x0;
x ← x0;
reward ← 0;
neighbors ← [ ];
iter ← 0;
next_state ←choose_action(0,2); # 2: initial state, prioritize exploration
repeat
iter++;

```
            if no_improvement =0 then
                state ← next_state;
                next_state ← choose_action(state,1); # 1: epsilon greedy function
            else
                next_state ←choose_action(0,2);
            endif
            x,re ← choose_action(x0,next_state);
            x,re ← choose_action(x0,next_state);
            if Objective_function(x)≤Objective_function(x_best) then
                xbest ← x;
                no_improvement ← 0;
            else
                no_improvement ← 1;
            endif
            reward← reward+re;
            a_prime ← np.argmax(Q[next_state]);
            Q[state,next_state]←Q[state,next_state]+alpha*(reward+gamma*Q[next_state,a_prime]-Q[state,next_state]);
            ep←ep*decay_rate;
            add x to neighbors;
        until iter ≥ nb_max
        return voisins;
end procedure
```

With **choose_action** is a function that, depending on the current state, decides what next action needs to be done. Its algorithms is as follows:

```
Procedure choose_action(state,type_function):
        next_state ← 0;
        if type_functio = 1 then
            next_state ← ϵ-greedy(state);
        else
            if type_function = 2 then
                next_state ← randomAction();
            endif;
        endif;
end procedure
```

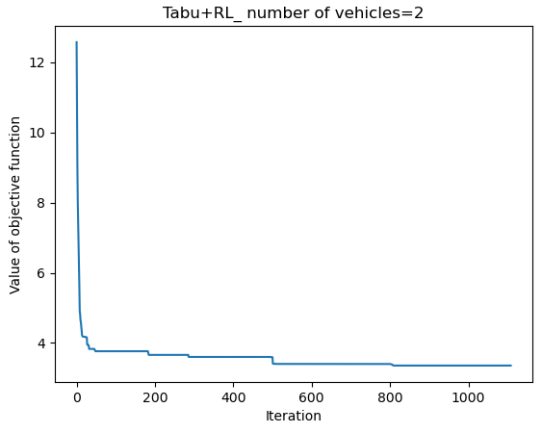
The ϵ-greedy function used in the previous algorithm helps balance between exploration(taking new actions and going to new states that weren't visited before) and exploitation(taking actions that are most prone to giving good rewards from what have been learned/experienced before).
The Q-matrix used in the AdaptiveSearchAlgorithm is a matrix of six-by-six representing 6 states as rows and 6 actions as columns and each coefficient of the matrix(initialized as zero) represents the quality of an action from what have been previously experienced. In other words, exploitation when in a current state S means taking the action that has the best quality, i.e highest Q value in the Q-matrix for the row corresponding to the state S.

5.1.2 Charts

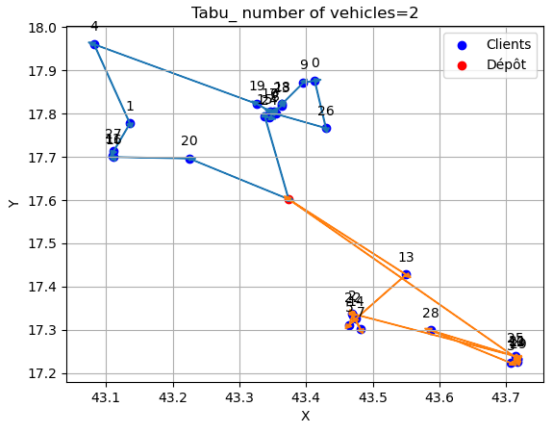
We launched the Tabu algorithm with RL at first, without MAS. The results found were so much better than classic Tabu algorithm as shown below:

We notice that there's no more overlapping between vehicles and routes intersect so much less than in the case of solutions optimized with classic Tabu algorithm. Two client neighborhoods were designated for two separate vehicles.
Now, let's test Tabu algorithms agents in MAS with Reinforcement learning. The results for non-collaborative agents is shown in the figure 26:

This test is the same as executing Tabu algorithm with RL 6 times and taking the best result out of

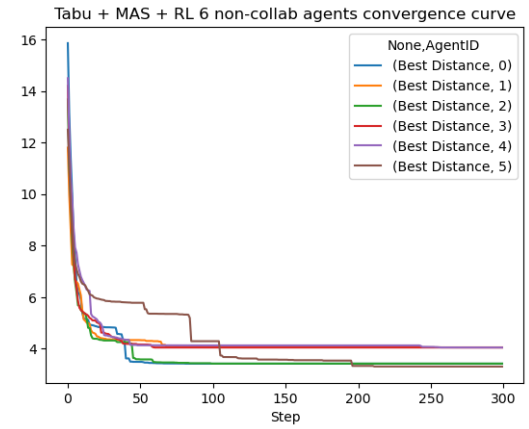


(a) Convergence curve

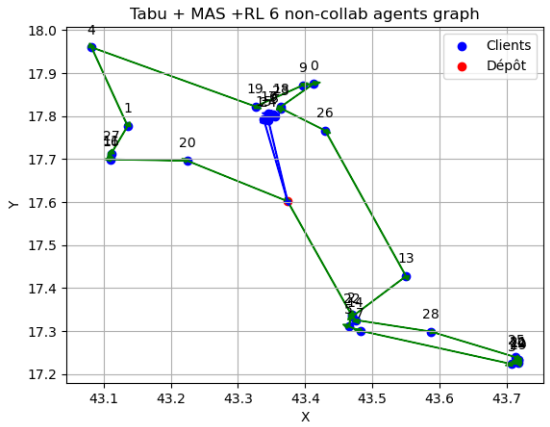


(b) Route graph

Figure 25: Tabu + Reinforcement learning



(a) Convergence curve



(b) Route graph

Figure 26: Tabu + Reinforcement learning

the six, since the MAS agents don't collab. A minor difference betwen the two is the number of iteratons. As shown on the convergence curves, Tabu alone iterates for over 1000 iterations while MAS execute 300 steps, which explains why the graph given by Tabu + RL is better than that given by non collab Tabu + RL agents.
As for collaborative agents, we found the following result in figure 27:

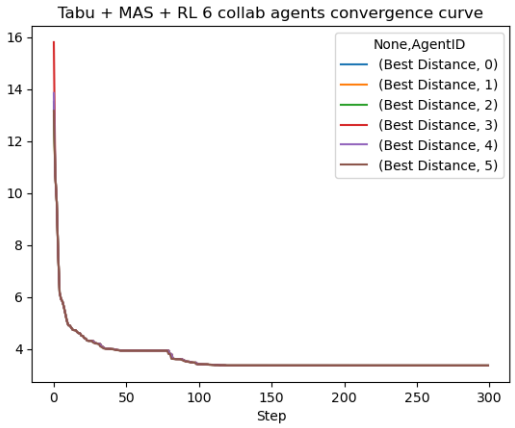
5.1.3 Results analysis

Replacing the neighbor generation function Tabu algorithm with the Adaptive Q_learning search algorithm helped go further in the optimization of solutions by decreasing the number of times routes intersect and separating vehicles into different client neighbors ensuring then a better solution.

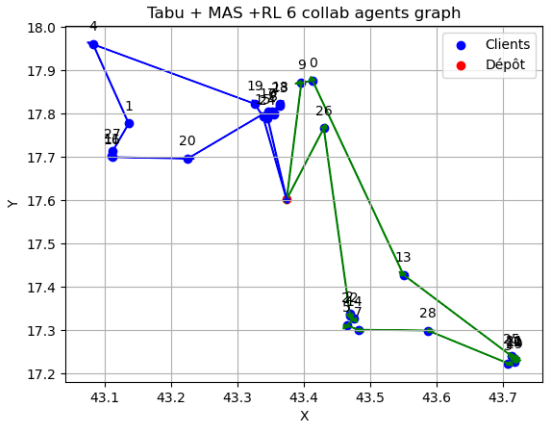
5.2 Simulated annealing with MAS and Reinforcement Learning

5.2.1 Specific VRPTW algorithm

The specific algorithm for solving the Vehicle Routing Problem with Time Windows (VRPTW) using a combination of metaheuristics, multi-agent systems (MAS), and reinforcement learning (RL). This algorithm aims to optimize the routes for a fleet of vehicles to serve a set of customers within specified time windows while minimizing total travel distance.



(a) Convergence curve



(b) Route graph

Figure 27: Tabu + MAS + Reinforcement learning

1. **Initialize**:
   - Set parameters (e.g., temperature for simulated annealing, exploration rate for RL).
   - Generate an initial solution (set of routes) for each vehicle, respecting time windows and capacity constraints.
2. **Repeat until convergence**:
   (a) **Simulated Annealing (SA) Phase**:
      i. Apply local search operators (e.g., swap, insertion) to explore the neighborhood of the current solution.
      ii. Evaluate the objective function (total travel distance) for the new solution.
      iii. Accept the new solution with a probability based on the SA acceptance criterion and temperature.

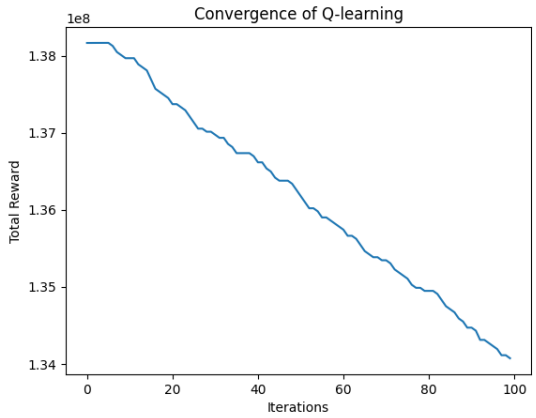
   (b) **Multi-Agent Systems (MAS) Phase**:
      i. Divide the problem into sub-problems and assign them to individual agents.
      ii. Agents optimize their assigned sub-problems using a combination of local search and cooperation with other agents.
      iii. Exchange information periodically to share promising solutions and coordinate efforts.

   (c) **Reinforcement Learning (RL) Phase**:
      i. Define states representing different configurations of vehicle routes and their associated rewards (e.g., minimizing total travel distance while meeting time windows).
      ii. Apply Q-learning algorithm to learn the Q-values for state-action pairs, where actions represent modifications to vehicle routes.
      iii. Update Q-values based on rewards obtained from exploring the solution space.

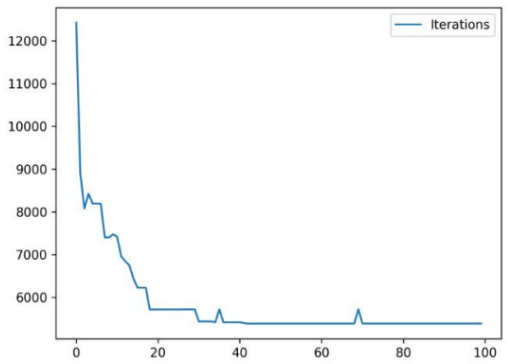
   (d) **Integration and Decision-making**:
      i. Combine the solutions obtained from SA, MAS, and RL phases.
      ii. Select the best solution among the candidates generated by different components.
3. **Evaluate the final solution**:
   - Assess the quality of the final solution in terms of total travel distance, adherence to time windows, and other relevant metrics.
   - If the convergence criteria are met, terminate the algorithm; otherwise, return to step 2.
4. **Output**:

- Return the optimized routes for each vehicle, along with associated costs and performance metrics.

Charts



(a) Convergence of Q-learning



(b) Curve of convergence

Results analysis

By leveraging Q-learning algorithms, vehicles learn optimal collision avoidance strategies by analyzing various factors such as vehicle speed, distance to surrounding vehicles, and potential obstacles. Through continuous interaction with the environment, vehicles refine their decision-making processes to identify safe trajectories and execute appropriate actions in real-time. This dynamic adaptation enables vehicles to proactively steer away from potential hazards, ensuring the safety of passengers, pedestrians, and other road users.

In dealing with large datasets, the omnipresence of fluctuations is indeed a key consideration. Fluctuations, or variations in data, are inevitable due to factors such as measurement errors, environmental changes, or inherent randomness in the system being studied. When applying Q-learning to collision avoidance strategies in dense traffic scenarios, the presence of fluctuations becomes even more pronounced and decreasing the time required for each route.

5.3 Genetic algorithm with MAS and Reinforcement Learning

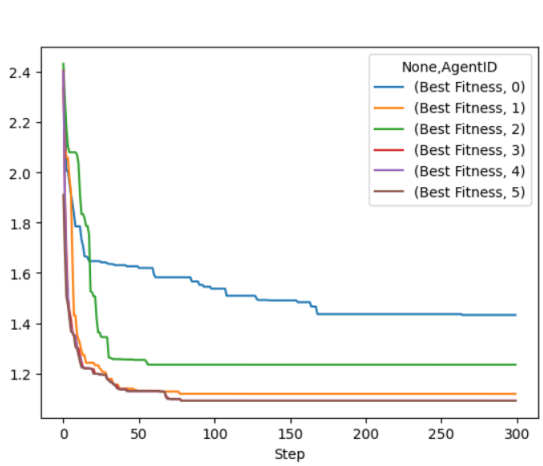
5.3.1 Specific VRPTW algorithm

In this section we chose to consider only the states 1, 2, 3, 5 as we wanted the number of clients in each route to remain unchanged.

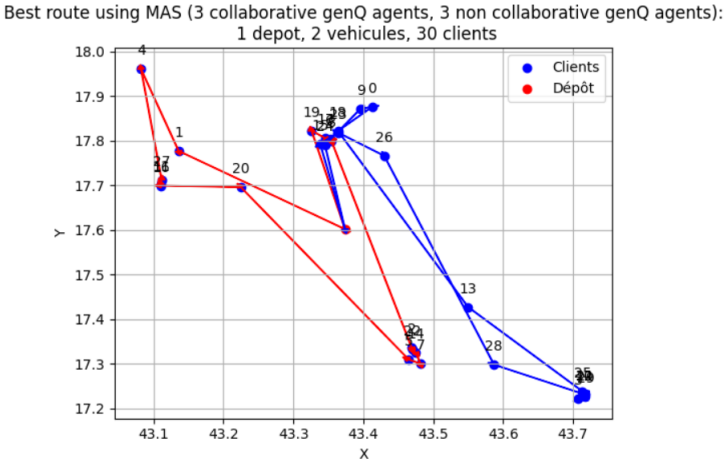
What differed from the use algorithm without Reinforcement Learning is the way the next generation is produced. In the previous sections, an elite population was chosen then crossbred. Their child resulted from fusion of genes from the original elite population. Then those genes were mutated. The way genes were put together and the mutation operation happened randomly with swapping operations. So the purpose of integrating the RL in our algorithms was the improvement of those operations and the Q-learning algorithm for the tabu algorithm's neighborhood generation was modified to mutate the child generation. However, the generation of the new population using the Q-learning algorithm was not covered in this study.

5.3.2 Charts

MAS genetic algorithm agents with reinforced learning as a function for their mutation gives the following results:



(a) Fitness evolution



(b) Best route

Figure 29: GA+MAS+RL

5.3.3 Results analysis

Overall, using MAS with RL was found less efficient than using the MAS alone with collaborative agents. This can be explained by the fact that the way the next generation of solution in each iteration is generated is not covered here by the Q-learning algorithm.

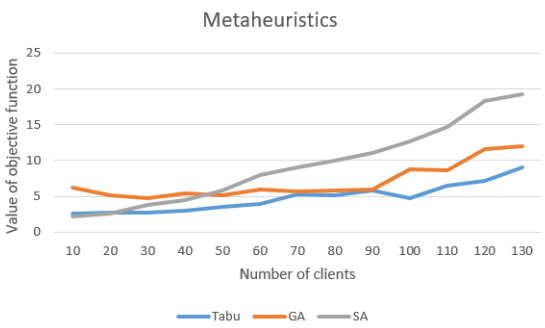
6 *Comparative and global analyses of collaborative optimisation performance*

Having developed the three optimization techniques for each metaheuristic algorithm, we conducted a comparative analysis of their performance, as shown in the table below:

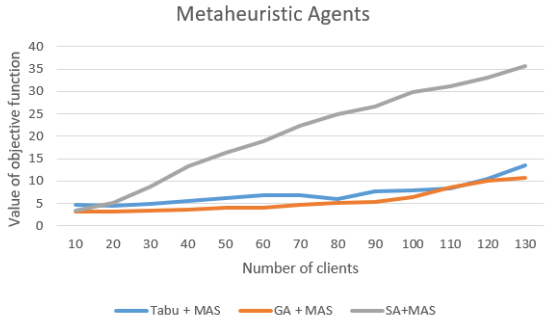
Number of clients	Metaheuristic			Metaheuristic Agents			Metaheuristic Agents + RL		
	GA(Genetic Algorithm)	SA(Simulated Annealing)	Tabu	Agent_GA	Agent_SA	Agent_Tabu	Agent_Ga	Agent_SA	Agent_Tabu
10	6.234177698	2.18	2.56	3.23846	3.496595	4.62124644	3.334186	3.561346	3.24
20	5.111437663	2.6	2.669953503	3.29722	5.205812	4.430886962	4.350874	5.20634	4.05047821
30	4.691775394	3.75	2.803551154	3.364988	8.760937	4.856152828	4.648593	7.12309	4.520738253
40	5.453231154	4.52	2.964334415	3.640948	13.225246	5.513927826	5.092763	8.22589	4.97055274
50	5.214007917	5.85	3.524208454	3.979849	16.410702	6.12512711	6.207478	10.46719	3.661755717
60	5.914872049	7.93	3.97080355	4.046102	18.940699	6.767303269	7.06032	12.105672	4.579466939
70	5.751697367	9.0685	5.342298812	4.741666	22.353174	6.802322914	7.284729	14.0761165	4.601158387
80	5.844337324	10.05	5.117032638	5.103496	24.833653	6.010774319	7.573813	16.8309425	6.262227503
90	6.003917146	11.11	5.846282274	5.299107	26.5624678	7.756596092	7.828429	19.0123467	6.234177698
100	8.828764133	12.67	4.802097193	6.456634	29.78013	7.855868144	8.81598	21.5601502	7.746309033
110	8.600249123	14.67	6.490598068	8.546352	31.124075	8.388315608	9.907713	23.67826	9.442074322
120	11.59491357	18.26	7.191206508	10.184066	33.1368902	10.59719839	13.36444	25.014527	12.40752167
130	12.063434	19.3	9.039576532	10.660602	35.6746064	13.43134648	15.664795	27.231626	14.58

Figure 30: Comparison table of the different techniques

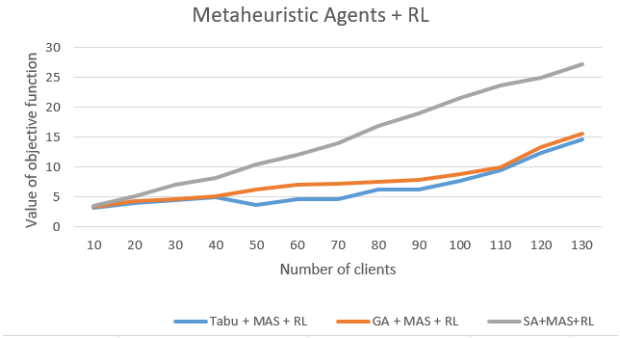
We plotted the evolution of the objective function as a function of the number of clients for each method and we found the following curves: This shows that the GA doesn't perform well for a small number of clients while on the other hand SA and Tabu algorithms are able to find better solutions. However, the more the number of clients increases the better GA performs until it becomes comparable to Tabu algorithm while SA diverges for high numbers of clients. Once we integrate MAS in the VRP problem, a noticeable improvement in GA occurs as it becomes the best algorithm for all chosen number of clients. Surprisingly, RL doesn't improve much GA but it does enable Tabu algorithm to improve its search for an optimal solution from it neighborhood generation.



(a) Standalone metaheuristics



(b) Metaheuristics + MAS



(c) Metaheuristics + MAS + RL

Figure 31: Comparison of different optimization methods

7 *Conclusion and outlook*

In this study, we utilized three metaheuristic algorithms (Tabu, GA and SA) to try and find optimal solutions for the Vehicle Routing Problem. We tuned the parameters of each algorithm and adapted them to this problem's data: number of vehicles and number of clients. We then developed Multi Agent Systems with agents as metaheuristic algorithms and enabled them to collaborate. This helped improve the optimal solutions found by these algorithms and gave reasonable routes graphs. Finally, we thought of improving the neighborhood generation and mutation processes by using Reinforcement Learning. Specifically, we used Q-learning algorithm and adapted it to this problem with states as different techniques of neighbor generation and actions as the different transitions from a current technique to another. This was found to be of great impact on these algorithms, especially the Tabu algorithm as it guided it through its choice of techniques to use to generate new neighbors that would help optimize the current solution.
Even though we took into consideration multiple constraints such as the distance to optimize, the number of vehicles to use and the time window for each client, the model we introduced to solve the VRPTW problem could include other constraints such as the maximum load for each vehicle and the exact routes that link clients.