Vehicle Routing Problem With Time Window: Metaheuristics combined with Muti-Agent Systems and Reinforcement Learning

Farouk MOHAMED[a], Slim HAMMADI[b]

[a]Ecole Centrale of Lille, Cité Scientifique, 59650 Villeneuve d'Ascq, France;
[b]CRISTAL, University of Lille, Cité Scientifique, 59650 Villeneuve d'Ascq, France;

ARTICLE HISTORY
Compiled July 23, 2024

ABSTRACT

Following the unprecedented growth of e-commerce prompting the rise of online shopping, the soar of food delivery especially during and after COVID-19 pandemic, and the diversification of delivery needs, the world is seeing immense demands on item deliveries with a shift in consumer expectations towards faster and more reliable delivery services. Consequently, companies are being put under the pressure to enhance their logistics capabilities. Retailers and delivery platforms are constantly innovating to offer enhanced delivery options, such as same-day and scheduled deliveries, to meet these expectations and stay competitive in the market. These features are what are currently described as the Vehicle Routing Problems. The aim of this article is to combine metaheuristic algorithms consisting of tabu algorithm, genetic algorithm and simulated annealing algorithm, with the concept of multi-agent systems thanks to the framework MESA and reinforcement learning, specifically the Q-learning algorithm. This approach's purpose is to make these metaheuristic algorithms collaborate to get out of local optima and to increase the optimization speed by introducing intelligence in the generation of neighbors, in the case of tabu and simulated annealing, and the mutation in the case of the genetic algorithm.

KEYWORDS
Sequencing theory, operations research, VRPTW, metaheuristic algorithms, multi-agent systems, reinforcement learning.

1. Introduction

The vehicle routing problem (VRP) is a part of a broad category of operations research problems known as network optimisation problems [1]. It 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 and to optimize the number of vehicles dispatched. This problem started with Dantzig and Ramser's research in Truck Dispatching Problem, the predecessor of VRP, in 1959 [2]. It was previously known as the "Truck Dispatching Problem" modelling the traffic of a fleet of trucks serving several gas stations in oil. Thanks to the works of G. Clarke and J.R. Wright in this topic, the "Truck Dispatching Problem" was generalized following the introduction of new approaches such as the "savings algorithm" proposed by Clarke and Wright in 1964, a heuristic method that significantly improved the

practical applicability of VRP solutions[3] . This problem has a variety of variations. For instance, there is Capacitated VRP considering the capacity of each dispatched vehicle, Multi-Depot VRP considering the possibility of having multiple depots, Green VRP putting emphasis on the minimiziation of environmental impact, and VRP with Time Windows incorporation time windows for deliveries[3, 4]. This work focuses of the latter. In the study headlined *Heuristics for Vehicle Routing Problem: A Survey and Recent Advances 34* [5], vehicle routing heuristics have been classified into three main categories.

- Constructive heuristics: These heuristics incrementally construct a solution by adding one element at a time, ensuring that all constraints are met at each step [6, 7].
- Improvement heuristics: Improvement heuristics for the Vehicle Routing Problem (VRP) are techniques used to iteratively refine an existing solution to achieve a better one, often by exploring the neighborhood of the current solution[8].
- Metaheuristics: Contrary to constructive and improvement heuristics, metaheuristics are not problem dependent and are generally derived from natural phenomena [9].

In this study, our approach is based on the use of metaheuristics. From the variety of metaheuritic algorithms, we worked on the tabu algorithm[11, 12, 16], the genetic algorithm [13, 14]accompanied with the simulated annealing algorithm[15, 16], although adding more algorithms such as the ant colony algorithm would be interesting.
Thanks to the modern advances in computation power and algorithmic techniques, significant strides have been made in VRP. And with the rise of Big Data, machine learning, and artificial intelligence, a wide range of techniques have been introduced to the issue of routing vehicles paving the way to the revolutionizing of VRP [17, 18, 19]. A particular method, named Multi-Agent Systems(MASs), traced back to the early developments of AI and distributed problem-solving, found its success in VRP. In fact, with the multitude of research studies in the topic of agent based algorithms and agent-oriented programming, the 2000s onwards have seen MAS becoming integral in various domains including solving problems that are of a complexity beyond that of a single agent[20, 21, 22]. The concept of MAS comes in handy when solving VRP problems thanks to the fact that it builds an environment in which multiple agents work on optimizing the same solution and can interact in a multitude of ways which makes this approach flexible and prone to give great results[23]. Combining metaheuristic algorithms with MAS by building an environment where agents are different metaheuristic algorithms e=interacting together to optimize a solution while learning from their previous experiences thanks to reinforcement learning is the main purpose of this work.

Complexity of VRP problems

Let's consider the basic vehicle routing problem. Its complexity of 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 [1].

In the case of dispatching a single vehicle for delivery, the problem is the same as a TSP(Traveling Salesman Problem). Solving this problem to optimality is of $O(n!)$ complexity. This was first observed using the British Museum Algorithm, also known as the Generate and Test-search(GTS), in the case of solving TSP [24]. Adding more vehicles to the problem increases its complexity.
For two vehicles: The first vehicle can follow routes of one client($1! \cdot \binom{n}{1}$ possibilities), two clients($2! \cdot \binom{n}{2}$ possibilities), three clients($3! \cdot \binom{n}{3}$ possibilities),..., or n clients($(n)! \cdot \binom{n}{n}$ possibilities), for n = Number Of Clients. For a choice of route involving k clients for vehicle one, we have $(n-k)! \cdot \binom{n}{n-k}$ choices of routes for vehicle two. In total we have

$$\sum_{k=1}^{n} k! \cdot \binom{n}{k} \cdot (n-k)! \cdot \binom{n}{n-k} = \sum_{i=1}^{n-1} k! \cdot (n-k)! \cdot \binom{n}{k}^2 = n! \cdot (2^n - 1) = O(n! \cdot 2^n)$$

Adding one vehicle to the problem multiplied the complexity of the problem by 2^n in the case of a second vehicle.

Applying the same logic to study the complexity of the Generate and Test search when adding a vehicle shows that its complexity is $O(n! \cdot 2^n)$ for vehicles numbering more than two.

In VRPTW problems, the number of vehicles that need to be dispatched is not specified to the algorithms. In fact, the heuristics need to optimize the number of delivery vehicles by taking into account multiple constraints such as delivery costs per vehicle. This increases the complexity of the problem as the algorithms need to apply the GTS algorithm for a set number of vehicles, multiplying the $O(n! \cdot 2^n)$ complexity by the number of vehicles that can be sent.
This causes tremendous computing time needed to solve the problem for a small number of clients. Suppose that the computation time is proportional to the number of iterations of the algorithm(this hypothesis under-estimates the real time needed for finding a solution) and that the time that a single operation takes is 10ns. We find that the time needed for solving the VRP problem(finding the exact solution) for 2 vehicles ready to be sent is:

- 3 seconds for 10 clients
- 1 year and 5 months for 15 clients
- $8 \cdot 10^7$ years for 20 clients

That's why we approach this problem as an NP-Complete one. This means that we will set certain constraints that, if met, the solution found will be used, even in the case of a locally but not globally optimal solution. 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.

2. Formulation of the problem

The research presented here primarily tackles the implementation of an algorithm combining metaheuristics, multi-agent systems and reinforcement learning for the purpose of optimizing VRPTW feasible solutions.

In the next section we will introduce the mathematical model used to formulate the problem.

2.1. *Parameters*

$NbVeh$: number of vehicles ready for dipatch
$NbCli$: number of clients awaiting delivery
$d_{i,j}$: distance between clients i and j
δ_i: cost for sending vehicle i for delivery
P: penalty for delivering an item to a client out of the specified time window
TW_i: time-window for client i

2.2. *Decision variables*

κ_i: a Boolean operator, set to 1 if the vehicle i sent for delivery, set to 0 otherwise
$u^k_{i,j}$: a Boolean operator, set to 1 if clients i and j are delivered their items in their respective orders by vehicle k, set to 0 otherwise
μ^i_j: boolean, set to 1 if client j received the corresponding item by vehicle i out of the specified time window, set to 0 otherwise.
$\gamma^k_{j,i}$: a Boolean operator, set to 1 if client j precedes(directly or not) client i in the delivery by vehicle k

$$\min \quad f = \sum_{k=1}^{NbVeh} \sum_{i=1}^{NbCli} \sum_{j=1}^{NbCli} u_{i,j}^k d_{i,j} \quad + \sum_{i=1}^{NbVeh} \delta_i \cdot \kappa_i + P \cdot \sum_{i=1}^{NbVeh} \sum_{j=1}^{NbCli} \mu_j^i \tag{1}$$

$$\text{s.t.} \quad \mu_j^i, \kappa_i \in \{0,1\} \qquad \forall i \in [\![1, NbVeh]\!] \quad \forall j \in [\![1, NbCli]\!], \tag{1a}$$

$$\kappa_k \geq u_{i,j}^k \qquad \forall k \in [\![1, NbVeh]\!], \forall i \neq j \in [\![1, NbCli]\!], \tag{1b}$$

$$\kappa_k \leq \sum_{\substack{i,j=1 \\ j \neq i}}^{NbCli} u_{i,j}^k d_{i,j} \qquad \forall k \in [\![1, NbVeh]\!], \tag{1c}$$

$$\sum_{j=1}^{NbCli} \gamma_{j,i}^k \sum_{h=1}^{NbCli} u_{jh}^k d_{j,h} \geq TW_i \cdot MV_k - M(1 - \mu_i^k) \qquad \forall i \in [\![1, NbCli]\!] \quad \forall k \in [\![1, NbVeh]\!], \tag{1d}$$

$$\sum_{j=1}^{NbCli} \gamma_{j,i}^k \sum_{h=1}^{NbCli} u_{jh}^k d_{j,h} \leq TW_i \cdot MV_k + M\mu_i^k \qquad \forall i \in [\![1, NbCli]\!] \quad \forall k \in [\![1, NbVeh]\!], \tag{1e}$$

$$\gamma_{j,i}^k \geq \sum_{h=1}^{NbCli} u_{j,h}^k \gamma_{h,i}^k \qquad \forall i, j \in [\![1, NbCli]\!] \forall k \in [\![1, NbVeh]\!], \tag{1f}$$

$$\gamma_{j,i}^k \geq u_{j,i}^k \qquad \forall i, j \in [\![1, NbCli]\!] \quad \forall k \in [\![1, NbVeh]\!], \tag{1g}$$

$$\gamma_{j,i}^k = 1 - \gamma_{i,j}^k \qquad \forall i, j \in [\![1, NbCli]\!] \quad \forall k \in [\![1, NbVeh]\!], \tag{1h}$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{NbCli} u_{i,j}^k \leq 1 \qquad \forall k \in [\![1, NbVeh]\!] \quad \forall i \in [\![1, NbCli]\!], \tag{1i}$$

$$\sum_{k=1}^{NbVeh} \sum_{\substack{i=1 \\ j \neq i}}^{NbCli} u_{i,j}^k = 1 \qquad \forall k \in [\![1, NbVeh]\!] \quad \forall i \in [\![1, NbCli]\!], \tag{1j}$$

$$\sum_{h=1}^{NbCli} \gamma_{h,i}^k - \sum_{h=1}^{NbCli} \gamma_{h,j}^k + (\sum_{h=1}^{NbCli} \gamma_{h,1}^k - 1) u_{i,j}^k \leq \sum_{h=1}^{NbCli} \gamma_{h,1}^k - 2 \quad \forall k \in [\![1, NbVeh]\!] \quad \forall i \neq j \in [\![1, NbCli]\!] \tag{1k}$$

$$\sum_{j=2}^{NbCli} u_{j,1}^k = \kappa_k \qquad \forall k \in [\![1, NbVeh]\!], \tag{1l}$$

$$\sum_{j=2}^{NbCli} u_{1,j}^k = \kappa_k \qquad \forall k \in [\![1, NbVeh]\!], \tag{1m}$$

$$\sum_{\substack{j=1 \\ j \neq i}}^{NbCli} u_{i,j}^k = \sum_{\substack{j=1 \\ j \neq i}}^{NbCli} u_{j,i}^k \qquad \forall k \in [\![1, NbVeh]\!] \quad \forall i \in [\![1, NbCli]\!] \tag{1n}$$

2.3. *Assumptions*

Throughout this study, we assume that all vehicles available for dispatch have sufficient capacity for us to not consider this as a constraint in our mathematical model. We also assume that they have equal cost for dispatch.

2.4. *Objective function*

$$OF = Min(\sum_{k=1}^{NbVeh}\sum_{i=1}^{NbCli}\sum_{j=1}^{NbCli} u_{i,j}^{k} d_{i,j} \quad + \sum_{i=1}^{NbVeh} \delta_i \cdot \kappa_i + P \cdot \sum_{i=1}^{NbVeh}\sum_{j=1}^{NbCli} \mu_j^i) \qquad (2)$$

The objective function (2) corresponds to the minimization of several criteria. The first criterion is the total distance traveled by all dispatched vehicles. The second reflects the total cost of dispatching a certain vehicles. The third reflects the penalty related to the out-of-time window delivery.

2.5. *Constraints*

The constraint(1b) guarantees the fact that a vehicle needs to be sent ($\kappa_k = 1$) if there are two clients of more that will be delivered their items by vehicle k. As for the constraint (1d), it indicates the fact that a vehicle can't be sent if the total distance to be be traveled by vehicle k is zero.
Constraints (1d) to (1h) correspond to the time window constraint in VRPTW. In fact, constraint (1d) and (1e) are the Integer Linear Programming formulation of the TW condition:

Algorithm 1: Time Window Constraint

input : Client i and vehicle k
Begin
if *Time to client i is greater than the specified time window* **then**
  $\mu_i^k = 1$;
else
  $\mu_i^k = 0$;
end
end

Constraints (1f) to (1g) define the decision variable $\gamma_{j,i}^k$. For instance, constraint (1f) translates the fact that if the client h delivered after client j precedes client i, than forcibly client j precedes client i. Constraint (1g) indicates that client j precedes client i if it is delivered directly before client i. And finally, constraint (1h) guarantees that if client i precedes client j, then j doesn't precede i.
Constraints (1i) to (1k) are the exact Miller-Tucker-Zemlin Formulations for the TSP problem, adapted to the variables used in this model.
Constraint, (1l) and (1m) indicate that the depot (assigned client number 1) needs to have one preceding client (constraint (1l)) and one succeeding client(constraint (1m)) with vehicle k if the vehicle k is sent ($\kappa_k = 1$) and none otherwise.
Finally, constraint (1n) ensures that a vehicle k doesn't end its route at a client's

adress.

3. Adaptation of metaheuristics to VRPTW

As previously emphasized, our approach is based on adapting metaheuristics to the problem of routing vehicles. The metaheuristics that we chose to work on are:

- The tabu algorithm
- The genetic algorithm
- The simulated annealing algorithm

3.1. *The 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 one. 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.
The algorithm of this heuristic is as shown in algorithm 1. We used to types of aspiration. The first one is the selection of a feasible solution from the tabu list if a number of iterations without improvement is surpassed. The second aspiration criteria used is the selection of the best neighbor **that isn't in the tabu list**. That way, we decrease the likelihood of premature convergence. At the end of each iteration, we merge the current feasible solution consisting of lists each containing the sequencing of clients for a vehicle. Then, we split the merged sequence to the most optimal number of vehicles to be dispatched when it comes to this sequence of clients. Thus, the algorithms takes into account all vehicles throughout all iterations.

Algorithm 2: the VRPTW tabu algorithm

input : Maximum number of iterations *maxIter*; Maximum length of the tabu list *maxLen*; number of iterations inducing aspiration *Aspiration*

output: the optimized solution *bestSol*

```
1  Begin
2  Initialization : randomly shuffle the list of clients and split it into a random
     number of sublists(each representing a vehicle) sol;
3  initialize the best solution bestSol as sol;
4  initialize an iteration count variable nbIter as 0;
5  initialize the best iteration bestIter as 0;
6  initialize count variable count as Aspiration;
8  while (the desired objective function not yet obtained) or (nbIter < maxIter)
     do
9      increment nbIter;
10     if nbIter-bestIter≥ Aspiration then
11         if count is equal to Aspiration then
12             Choose a solution from the tabu list;
13         end
14         decrement count;
15     end
16     if count is equal to 0 then
17         update count as Aspiration
18     end
19     Generate neighbors ;
20     Choose the best neighbor bestNeighbor ;
21     while bestNeighbor is in tabuList do
22         Remove bestNeighbor from neighborhood;
23         Choose the new bestNeighbor;
24     end
25     if bestNeighbor is better than sol then
26         update bestSol as bestNeighbor ;
27         update bestIter as nbIter;
28     end
29     Add bestNeighbor to the tabu list;
30     if length of the tabu list is greater than maxLen then
31         remove the oldest tabu candidate;
32     end
33     Update sol as bestNeighbor;
34     Merge all sequences of all dispatched vehicles in sol;
35     Split the new sequence into the different possible numbers of vehicles to be
         sent;
36     Update sol as the best solution out of the split sequences;
37 end
38 end
```

3.2. *The simulated annealing algorithm*

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) [25]. 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 [26]. 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. A simulated 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 prevent being stuck at a local minimum [26]. 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.
The simulated annealing algorithm adapted to VRPTW is as presented in algorithm 2.

Algorithm 3: the VRPTW SA algorithm

input : Maximum and minimum temperature $Tmax$ and $Tmin$; the maximum number of iterations $maxIter$; the desired objective function; The temperature decay rate *alpha*

output: the optimized solution *bestSol*

Begin
Initialization : randomly shuffle the list of clients and split it into a random number of sublists(each representing a vehicle) *sol*;
initialize the best solution *bestSol* as sol;
initialize an iteration count variable *nbIter* as 0;
initialize the best iteration *bestIter* as 0;
while *(the desired objective function not yet obtained) or (nbIter* $<$ *maxIter)* **do**
    Increment *nbIter*;
    Generate a neighbor ;
    Split the neighbor to the optimal number of vehicles ;
    Compute the energy difference $delta = E_{new} - E_{init}$;
    generate a random value $0 \leq r \leq 1$;
    if *(delta* $<$ *0)* **then**
        update *sol* and *bestSol* as the generated neighbor;
        Update *bestIter* as *nbIter*
    end
    else if $(r \leq e^{-delta/T})$ **then**
        update sol as the generate neighbor
    end
    Update *sol* as *bestNeighbor*;
    Merge all sequences of all dispatched vehicles in *sol*;
    Split the new sequence into the different possible numbers of vehicles to be sent;
    Update *sol* as the best solution out of the split sequences;
    if *(nbIter* $-$ *bestIter* $\geq$ *iter)* **then**
        increase temperature to get out of local optimum
    else
        Decrease current temperature by multiplying it by the decay rate *alpha*
    end
end
end

3.3. *The genetic algorithm*

Genetic algorithms have been inspired by the natural selection mechanism introduced by Darwin. They apply certain operators to a population of 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 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.

Algorithm 4: the VRPTW genetic algorithm

input : Size of the population *popSize*; size of the sample of elite individuals *eliteSize*; the mutation rate *mutationRate*: The number of generations *nbGen*
output: the best solution population *pop*

```
Begin
Initialization : generating an initial population;
while (a fixed number of generations is not reached) do
    Evaluate individuals ;
    Parent selection for reproduction ;
    Apply a controlled crossover algorithm, in order to obtain viable offspring1
     and offspring2 ;
    Apply a controlled mutation algorithm with a probability mutationRate ;
    Update the population by changing the best individual by optimizing the
     number of routes after a merge and a split approach;
end
end
```

4. The dynamic process: a multi-agent system

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 environ-

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

4.1. *MAS agents*

Integrating the tabu, simulated annealing and genetic algorithms in a Multi-Agent System consists of creating an agent class with its `step` method corresponding to a single iteration in each algorithm. This could be interpreted by the fact that executing our MAS algorithm for N steps is the same as an algorithm being run with N as its $maxIter$ parameter.

4.2. *Agent interactions*

In a MAS environment, agents are prone to interact with each other. These interactions can be:

- **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 feasible solution better than its current one.
- **Non-collaborative:** this means that an agent doesn't share nor take into account the solutions found by other agents.

The contact algorithm is as shown in algorithm 5:

5. Collaborative Optimisation: Metaheuristics with MAS and Reinforcement Learning

In the three metaheuristic algorithms studied above, there are functions that generate neighbors of a feasible solution(in the case of tabu algorithm and simulated annealing) and functions that execute mutations(in the case of genetic algorithm). The purpose of this section is to improve these functions using a technique called Reinforcement Learning adding intelligence to the actions taken by the MAS algorithm agents.

Algorithm 5: MAS contact method

```
   input  :  MAS collaborative agent: agent0
   output: Updated MAS agent
 1 Begin
 2 for agent in MAS environment do
 3 |   if (agent is collaborative) and (agent found a better solution than agent0 )
   |    then
 4 |   |   if agent0 is GA then
 5 |   |   |   Replace agent0's best individual by agent's best found route at time
   |   |   |    step t;
 6 |   |   else
 7 |   |   |   replace agent0's best found solution and current
   |   |   |    solution(optimisation in progress) by agent's best found solution
 8 |   |   end
 9 |   end
10 end
11 end
```

5.1. *Reinforcement Learning*

Reinforcement Learning is a technique based on the notion of trial and error. It is a subset of machine learning inspired by behavioral psychology[27], where an agent learns to make decisions by taking actions in an environment to maximize some notion of cumulative reward. It consists of executing multiple actions and learning from its mistakes and successes thanks to the concept of reward. This concept is based on Bellman's dynamic programming [28] and the temporal difference (TD) learning algorithm introduced by Andrew Barto, Richard Sutton, and Charles Anderson[29].
In reinforcement learning, an agent interacts with its environment at each time step t(we suppose that time steps are discrete). This interaction is resumed in three information:

- The environment's state S_t
- The action chosen by the agent A_t
- the reward following this action R_t

This can be represented by the following diagram:

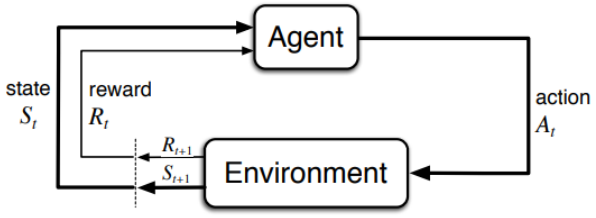


Figure 1.: agent–environment interaction [29]

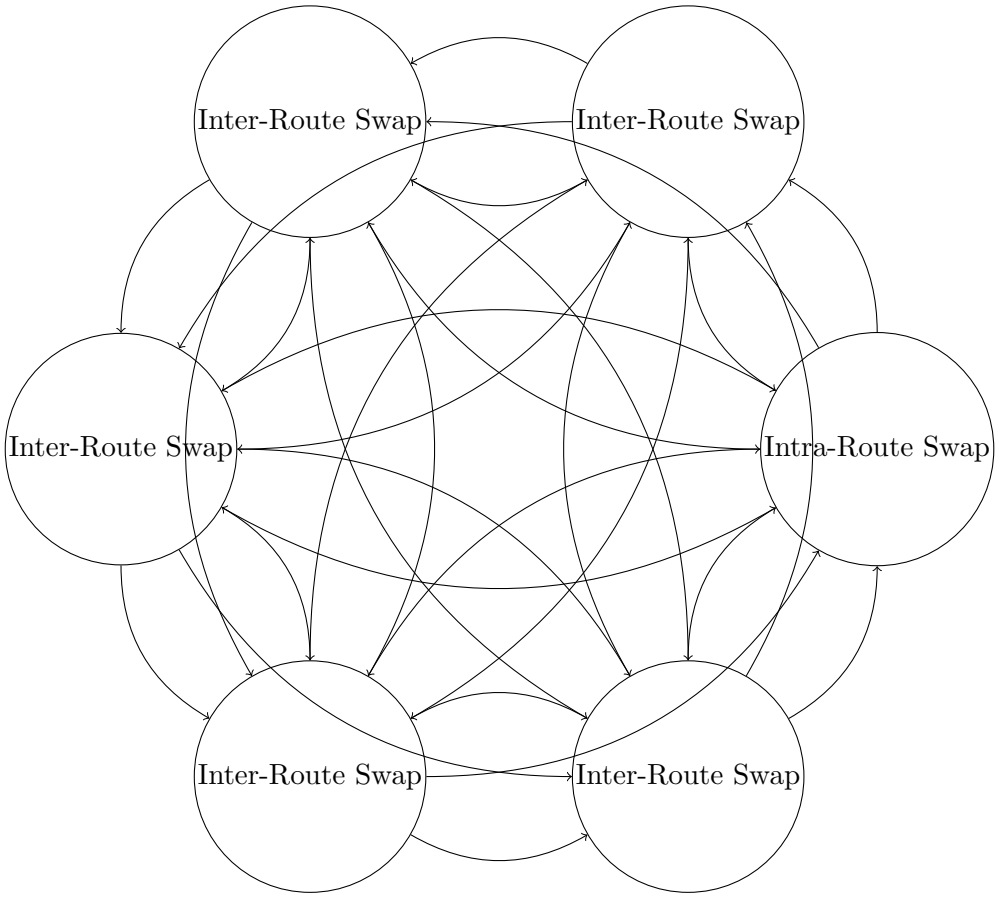
The states that we considered are $S \in$ {InterRouteShift ; IntraRouteShift ; InterRouteSwap ; IntraRouteSwap ; TwoIntraRouteSwap ; TwoIntraRouteShift}:

(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.

The decision process (S,A,P,R) where S is a set of possible states of the agent in its environment, A is the set of the different actions the agent could take, P is a state transition probability matrix and R is the reward function, can be seen as a Markov Decision Process(MDP). In fact, the future state, action and reward can be supposed to be dependent only on the current state of the agent/environment. Thus the different transitions can be represented by the following graph:



There are multiple algorithms for reinforcement learning. In this study, we used the **Q-learning** algorithm.

5.2. *Q-learning algorithm*

The purpose of integrating RL in our algorithms is the improvement of neighbor generation and population mutation. This can be achieved by maximizing the total rewards the agent will get from the environment. The function to maximize is called the expected discounted return, function that we denote as G.

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{i=t+1}^{\infty} \gamma^{i-t-1} R_i = R_{t+1} + \gamma G_{t+1}$$

In order for the agent to do that, it needs to optimize its policy. The policy is a probability distribution of actions over states:

$$\pi(a|s) = P(A_t = a|S_t = s)$$

The Q-learning algorithm being a value-based method, its policy is implicit and isn't directly optimized, it's the action-value function q that is optimized.

$$q_{new}(s,a) = (1-\alpha) q_{old}(s,a) + \alpha(R_{t+1} + \gamma \max_{a'} q(s',a'))$$

In this algorithm, the policy consists of an algorithm **choose_policy** that, depending on the current state, decides what next action needs to be done. Its algorithm is as follows:

Algorithm 6: Policy choosing algorithm

input : current state and the type of function to apply;
Q matrix;
epsilon greedy rate(ϵ);
output: Next state to visit

Begin $next\_state \leftarrow 0$;
if $type\_function == 1$ **then**
  $next\_state \leftarrow RandomAction()$;
else
  $next\_state \leftarrow \epsilon - greedy(state,)$;
end
end

This policy is a combination of a random policy and epsilon-greedy policy. When $type_function$ is equal to one, the algorithm prioritizes a random policy since the agent is in its initial state. Otherwise, the algorithm chooses an epsilon-greedy policy since it enables the agent to explore its environment and exploit its experience. The ϵ-greedy function helps balance between exploration(taking new actions and going to new states that weren't previously visited) and exploitation(taking actions that are most prone to giving good rewards from what have been learned/experienced before).
When it comes to generating neighborhoods (tabu agents) and generating mutated children(Genetic Algorithm agents), the RL algorithms are the same. We developed the following Q-learning algorithm for tabu and GA agents(algorithm 7).

Algorithm 7: Adaptive Search Algorithm with Q-Learning

input : An initial feasible solution x_0;
Decay rate λ;
Discount rate γ;
epsilon greedy rate ϵ;
Learning rate α;
output: Next state to visit

Begin
initialise Q;
$improved \leftarrow 0$;
$no\_improvement \leftarrow 0$;
$x\_best \leftarrow 0$;
$x \leftarrow x_0$;
$reward \leftarrow 0$;
initialise empty list of neighbors;
$iter \leftarrow 0$;
$next\_state \leftarrow choose\_action(0, 2, Q, \epsilon)$; ▷ 2: initial state, prioritize random policy
while $iter \leq nb_max$ **do**
  iter++;
  if $no\_improvement == 0$ **then**
    $state \leftarrow next\_state$; $next\_state \leftarrow choose\_action(state, 1, Q, \epsilon)$; ▷ #1: epsilon greedy function
  else
    $next\_state \leftarrow choose\_action(0, 2, Q, \epsilon)$
  end
  $x, re \leftarrow apply\_action(x_0, next\_state)$;
  if $Objective\_function(x) \leq Objective\_function(x\_best)$ **then**
    $x\_best \leftarrow x$
    $no\_improvement \leftarrow 0$
  else
    $no\_improvement \leftarrow 1$
  end
  $reward \leftarrow reward + re$
  $a\_prime \leftarrow Max\_Arg(Q[next\_state])$ ▷ choose the action a_prime that most improves the learning
  $Q[state, next\_state] \leftarrow Q[state, next\_state] + \alpha * (reward + \gamma * Q[next\_state, a\_prime] - Q[state, next\_state])$
  $\epsilon \leftarrow \epsilon \times \lambda$
  add x to list of neighbors
end
End

As for SA agents, the reasoning is the same, taking actions based on what have been previously experienced by the agents. However, some changes to the Adaptive Search Algorithm needed to be made. In fact, this new version of the Q-learning algorithm takes as input additional data: the agent's Q-learning matrix, its epsilon greedy parameter, a parameter indicating if it has improved or not $no_improvement$. The reason behind this is the fact that SA agents need only one neighbor at each iteration, so it needs to remember the data resulting from the previously generated neighbors to learn from its test and trial experience. Whereas, the tabu and GA agents' Adaptive Search Algorithm iterates multiple times (more than 50 times) before generating the desired neighborhood. Thus, when a tabu agent desires to generate a new neighborhood, it doesn't need to remember the previous data generated, as it will manage to learn well enough through a sufficient number of iterations.
The Q-matrix used in the Adaptive Search Algorithm 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.

Algorithm 8: Adaptive Search Algorithm with Q-Learning

input : An initial feasible solution x_0;
An initial Q matrix;
An initial state;
Initial epsilon ϵ;
reward;
learning rate α;
decay rateλ;
discount rate γ ;
A binary variable ($no\_improvement$) that's equal to 1 if the previous neighbor improved the solution, 0 if it didn't;

output: A newly generate neighbor of x_0;
Updated Q matrix;
Updated ϵ;
Updated $no_improvement$;
Updated reward;

Begin
$x \leftarrow x_0$;
if $no\_improvement == 0$ **then**
  $next\_state \leftarrow choose\_action(state, 1, Q, \epsilon)$;
  ▷ #1: epsilon greedy function
else
  $next\_state \leftarrow choose\_action(state, 2, Q, \epsilon)$
end
$x, re \leftarrow apply\_action(x_0, next\_state)$;
$reward \leftarrow reward + re$
$a\_prime \leftarrow Max\_Arg(Q[next\_state])$ ▷ choose the action a_prime that most improves the learning
$Q[state, next\_state] += \alpha * (reward + \gamma * Q[next\_state, a\_prime] - Q[state, next\_state])$
$\epsilon \leftarrow ep \times decay\_rate$
if $Objective\_function(x) \leq Objective\_function(x_0)$ **then**
  $no\_improvement \leftarrow 0$
else
  $no\_improvement \leftarrow 1$
end
End

6. Simulation results

6.1. *Metaheuristics*

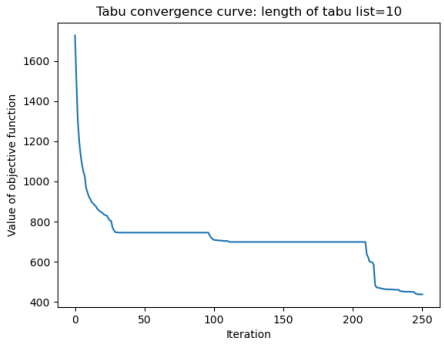
6.1.1. Tabu algorithm

The VRPTW Tabu algorithm has multiple parameters with great impact on the convergence time and the optimality of the final feasible solution. These parameters are as follows:

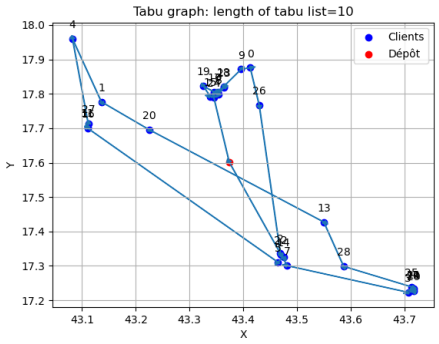
(1) The maximum number of iterations without finding a new best solution *Aspiration*: This parameter serves as a first aspiration criteria.
(2) The limit objective function: This designates a value of our objective function that we would be satisfied with if obtained. This parameter serves as a stopping 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 2;

    From these tests, we notice that higher the value of the maximum length of tabu algorithms tabu list, the faster it stagnates(premature convergence) and the longer it takes for it to get out of the local optimum it found. 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. In the case of small sizes of the tabu list, convergence occurs rapidly. Even though this helps reduce computation time and cost, a compromise needs to be made between the efficiency of the algorithm(giving it time to explore for a longer time) and speed of optimisation.
(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.

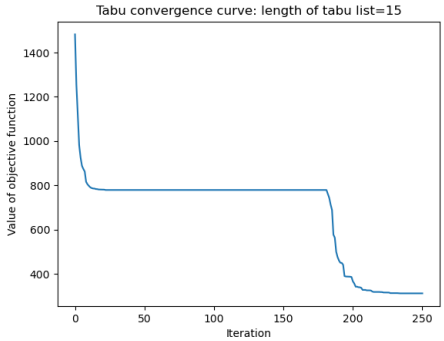
    The graphs and convergence curves shown in *figure 3* illustarte the fact that the higher the number of neighbors we set in our algorithm, the faster the convergence occurs and the better the final solution is. However, higher neighbors to generate means higher computational cost of the algorithm resulting 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



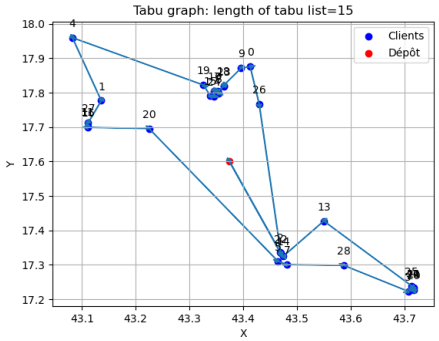
(a) Tabu maximum length of tabu list = 10 convergence curve



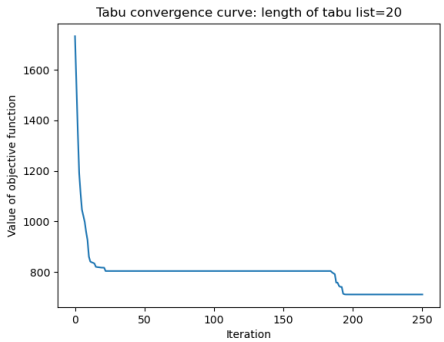
(b) Tabu maximum length of tabu list = 10 routes graph



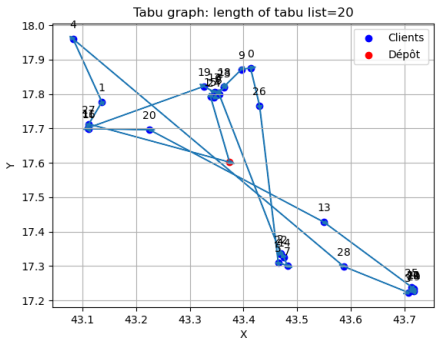
(c) Tabu maximum length of tabu list = 15 convergence curve



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



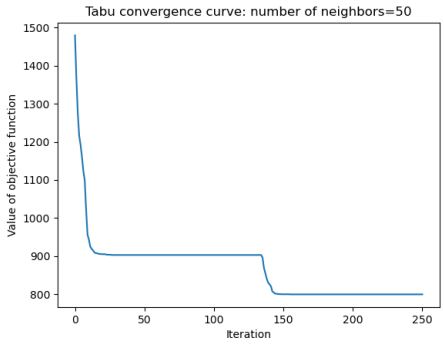
(e) Tabu maximum length of tabu list = 20 convergence curve



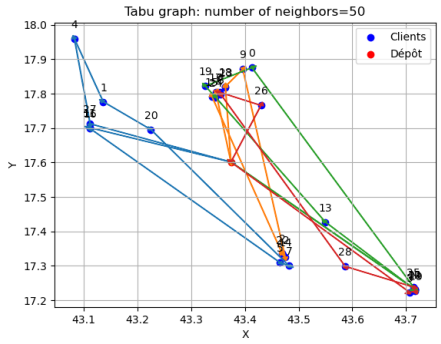
(f) Tabu maximum length of tabu list = 7 routes graph

Figure 2.: Impact of the tabu list's maximum size on the algorithm

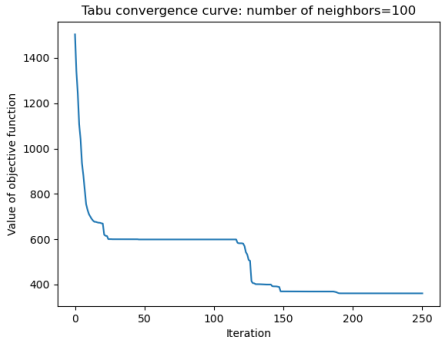
equal to 100 for the rest of the study.



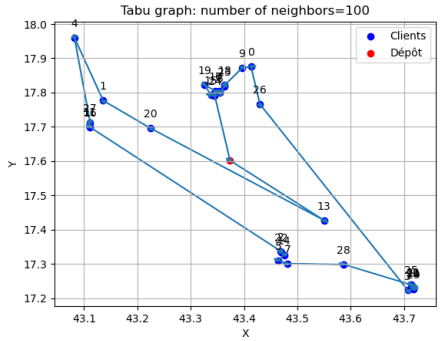
(a) Tabu number of neighbors = 50 convergence curve



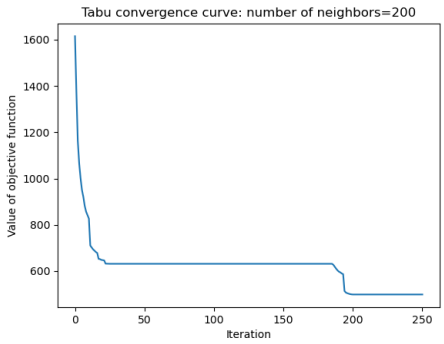
(b) Tabu number of neighbors = 50 routes graph



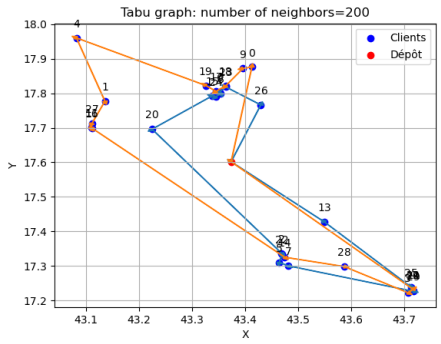
(c) Tabu number of neighbors = 100 convergence curve



(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 3.: Impact of the number of neighbors on the algorithm

6.1.2. Genetic algorithm

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 4. This shows that the greater the value of population size parameter, the less stagnation occurs and the faster optimality is reached.

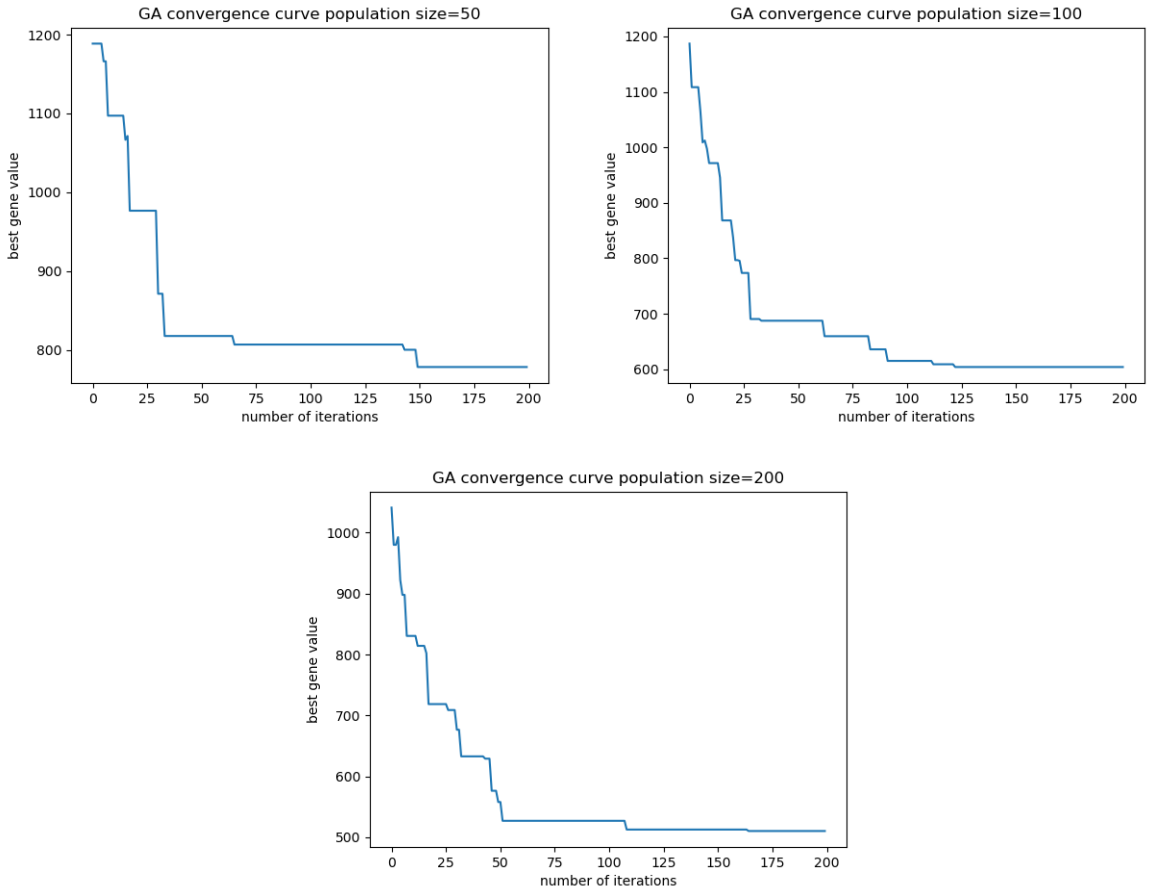


Figure 4.: Impact of population size parameter on the convergence curve

(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.

(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 5.

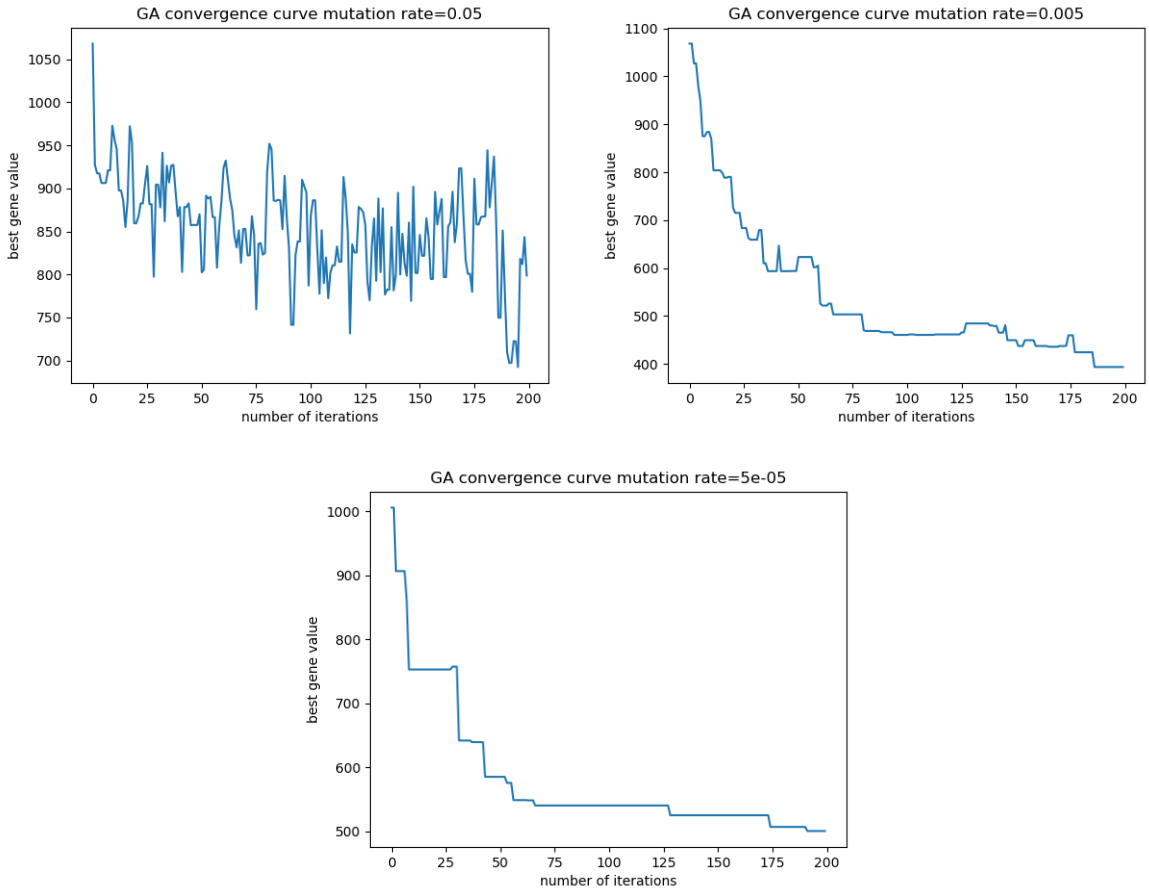


Figure 5.: impact of mutation rate

6.1.3. Simulated Annealing algorithm

The simulated annealing algorithm needs a decay rate called alpha, a minimum temperature, a maximum temperature and a maximum number of iterations as inputs.
The minimum temperature, maximum temperature and maximum iterations parameters are used for limiting the run time of the algorithm. In fact, once the current temperature of the system(starting at maximum temperature) reaches the specified minimum temperature due to the effect of the decay rate alpha, and/or when the number of iterations exceeds the maximum number of iterations set as an input, the algorithms stops.

As for the decay rate **alpha**, this parameter is of great importance as it has to be tuned for the algorithm to give good results. Below are graphs and convergence curves showing the impact of this parameter.

These convergence curves show that the more we increase the decay rate of the temperature, the faster the algorithm converges. This is explained by the fact that a small decay rate decreases the value of the temperature T extremely fast which gives a low value for $e^{\frac{-\Delta E}{T}}$. Thus, replacing the current solution with a neighbor occurs less than in the case of high values of T. However, having a high decay rate alpha increases

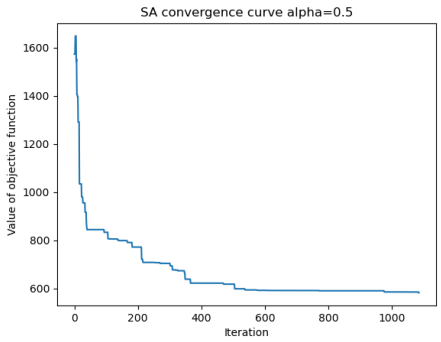


Figure 6.: convergence curve for decay rate alpha=0.5

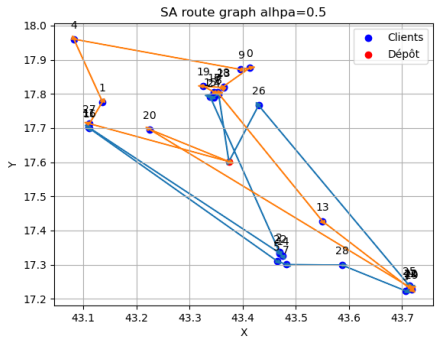


Figure 7.: Route graph for decay rate alpha=0.5

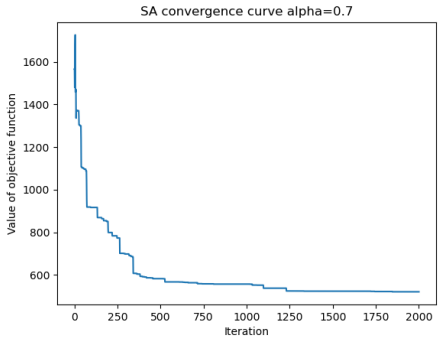


Figure 8.: convergence curve for decay rate alpha=0.7

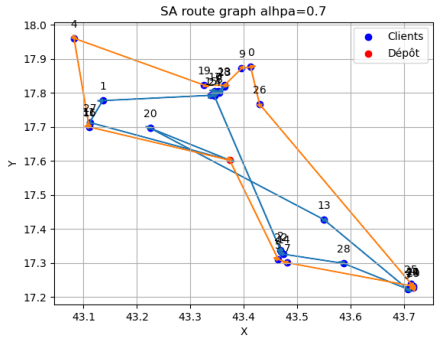


Figure 9.: Route graph for decay rate alpha=0.7

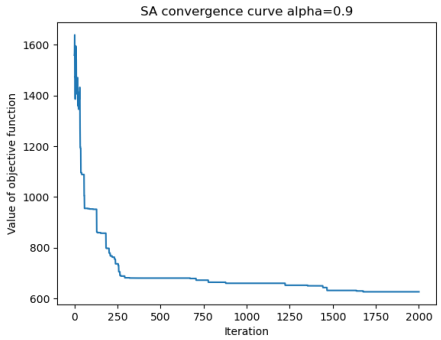


Figure 10.: convergence curve for decay rate alpha=0.9

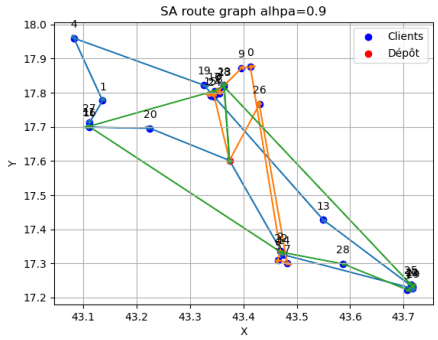


Figure 11.: Route graph for decay rate alpha=0.9

the cases where the algorithm replaces the solution by a neighbor that is worse when it comes to its objective function(i.e Energy). Setting alpha=0.995 as input to the algorithm gives the following convergence curve and route graph.

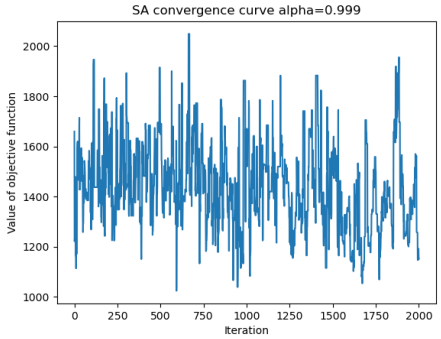


Figure 12.: convergence curve for decay rate alpha=0.999

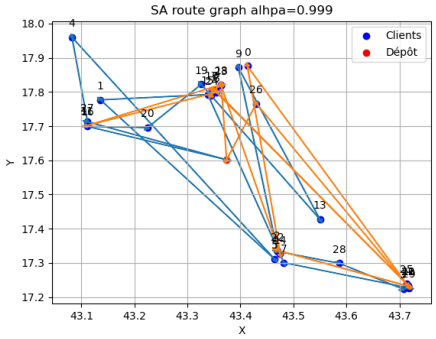
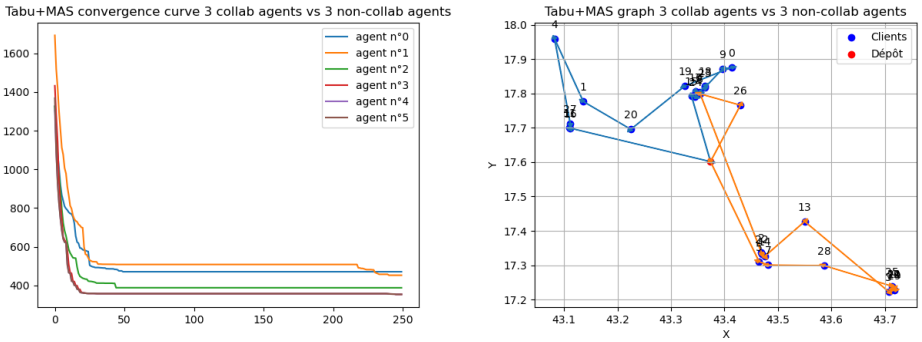


Figure 13.: Route graph for decay rate alpha=0.999

6.2. *Multi-Agent System: metaheuristics as agents*

The results of the simulations of each metaheuritic algorithm showed that, even though parameters were thoroughly tuned, phenomena such as premature convergence(in the case of tabu algorithm), unpredictable behaviour(in the case of SA) and the need for a relatively large number of iterations(in the case of GA), were noticed. The idea behind putting to use MAS environments is to make these algorithms collaborate since their behaviors are complementary: tabu algorithm's stagnation and SA's unpredictability could be resolved by GA's steady behavior and GA's optimisation could be accelerated by tabu and SA's fast convergence.

Starting by implementing MAs environments with agents restricted to a single algorithm, we noticed that the results given by these algorithms drastically improved. In fact, in the case of tabu agents collaborating, simulations showed that, even though stagnation still occurred early, the agents were able to further improve the final solution, as shown in the figure 14.

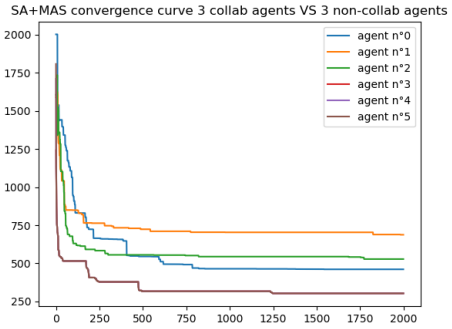


(a) Convergence curve

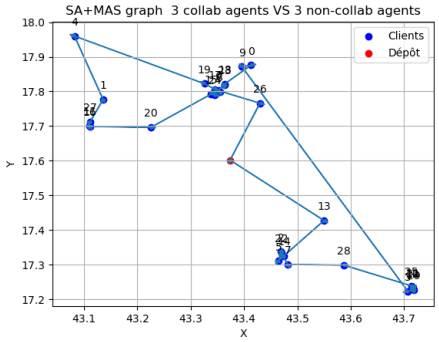
(b) Route graph

Figure 14.: MAS+Tabu 3 collaborative agents vs 3 non collaborative agents

In this figure, a comparison between collaborative and non-collaborative agents is presented. The collaborative agents(agents 3 to 5) show a remarkably fast convergence compared to their counterparts(agents 0 to 2). When it comes to SA agents, collaborative agents were able to rapidly optimize the solution compared to non-collaborative ones, as presented in the figure 15.



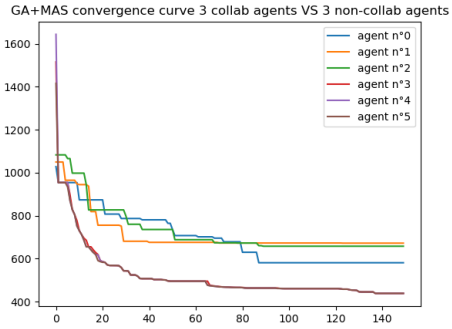
(a) Convergence curve for 3 collaborative agents and 3 non-collab agents



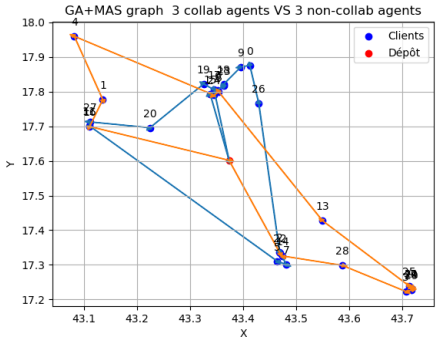
(b) Graph of routes for 3 collaborative agents and 3 non-collab agents

Figure 15.: MAS+Simulated Annealing

Finally, thanks to collaboration between genetic algorithm agents, the convergence of the algorithm became noticeably faster as shown below:



(a) Fitness evolution

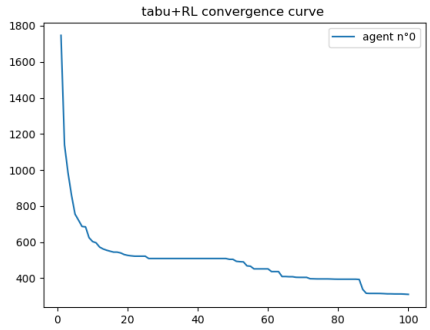


(b) Best route

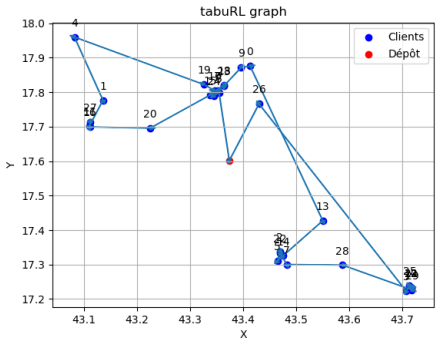
Figure 16.: Genetic algorithm with MAS

6.3. *Reinforcement learning*

We launched the Tabu algorithm with RL at first, without MAS. The results found showed that the now intelligent tabu agent doesn't converge prematurely and further optimizes the solution compared to the classic Tabu algorithm, as shown in figure 17:



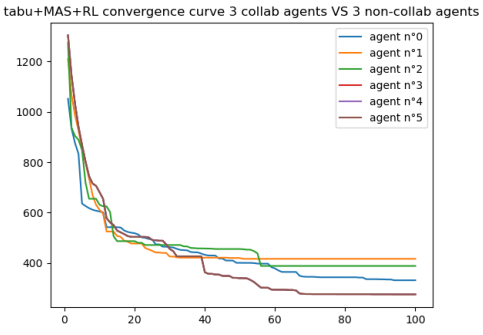
(a) Convergence curve



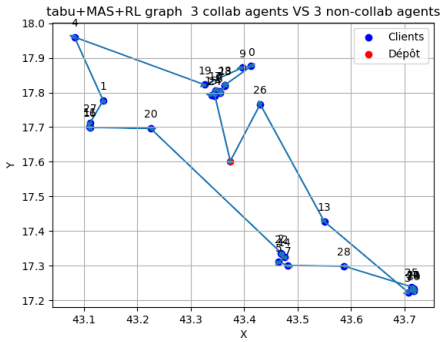
(b) Route graph

Figure 17.: Tabu + Reinforcement learning

The implementation of an MAS environment with 6 intelligent tabu agents three of whom are collaborative gave the convergence curve shown in figure 18(a). We noticed that at the beginning, non-collaborative agents were as good as collaborative ones. However, as the agents started to learn with RL, the collaborative agents were able to accelerate their optimization and thus finding a much better final solution.



(a) Convergence curve



(b) Route graph

Figure 18.: Tabu + MAS + Reinforcement learning

When it came to Simulated Annealing, tests of its performance when using RL to generate neighbors unpredictable behaviour. In fact, even though some agents were able to find feasible solutions with lost cost, other agents converged prematurely and remained stuck at local optima throughout the majority of the iterations. This result is shown in the convergence curve of SA with reinforcement learning and 6 Multi Agent System agents. We also noticed that after learning throughout 300 to 400 iterations, SA agents showed sudden improvement in their optimization of the feasible solution. The same could have happened to the two other SA agents that remained in a local optima until the 500th iteration.

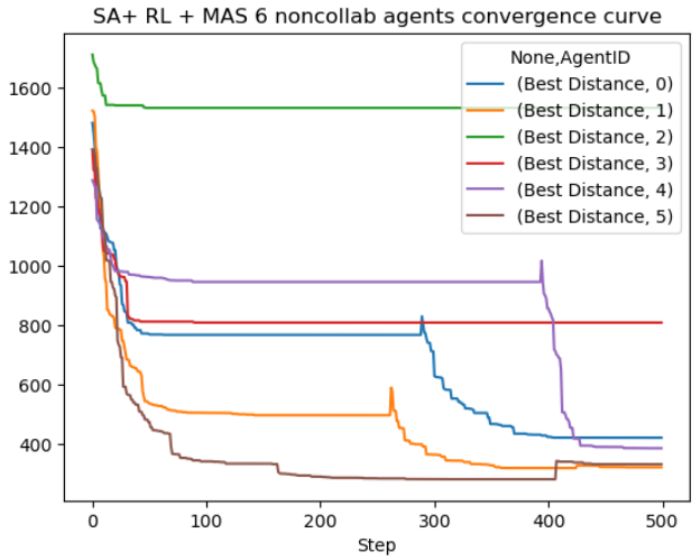
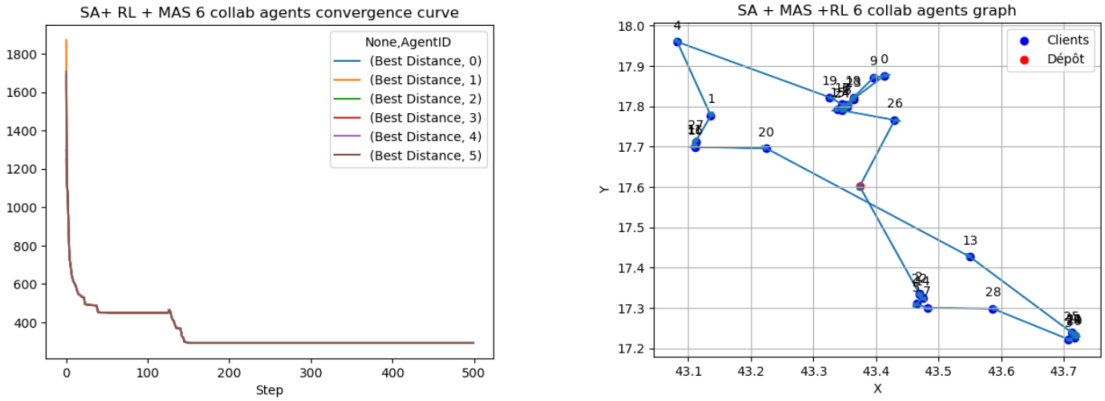


Figure 19.: Convergence curve of 6 non collaborative SA agents with RL

Collaborative mode came in handy as it helped get rid of this unpredictability, as shown in figure 20. The fact that SA agents collaborated by sharing their feasible solutions enabled them to restrict their unpredictability and increase the speed of optimization.



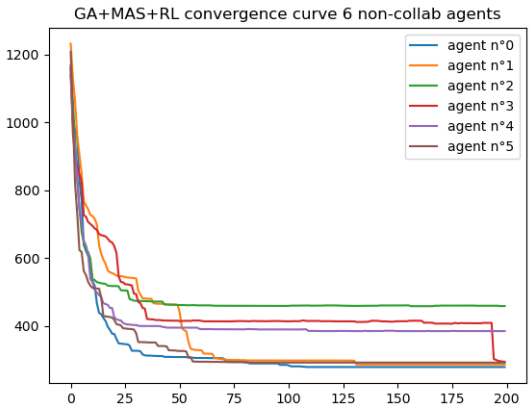
(a) Convergence curve

(b) Route graph

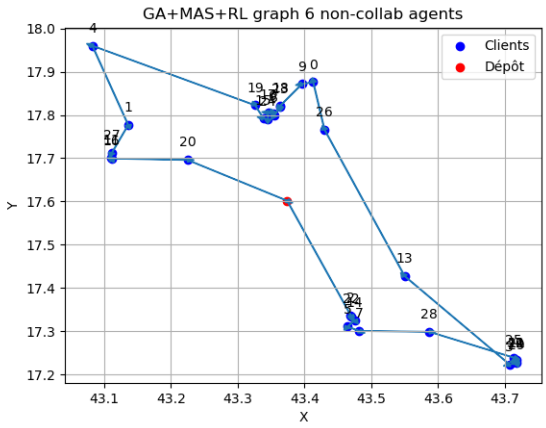
Figure 20.: SA + MAS + Reinforcement learning

The simulation results of genetic algorithm agents implemented with reinforcement learning showed that MAS GA agents have unpredictable behaviour, same as SA agents with RL, but the final feasible solutions found by GA agents don't have objective functions of largely different scales as the case of SA agents. This is shown in figure 21.

When we launched the algorithm with 6 collaborative agents, we noticed an increase in the speed of problem optimization(after only 50 time steps the MAS collaborative GA + RL agents found a similar solution to the one found by the 200 time steps of 6 non-collaborative GA+RL agents), as shown in figure 22.

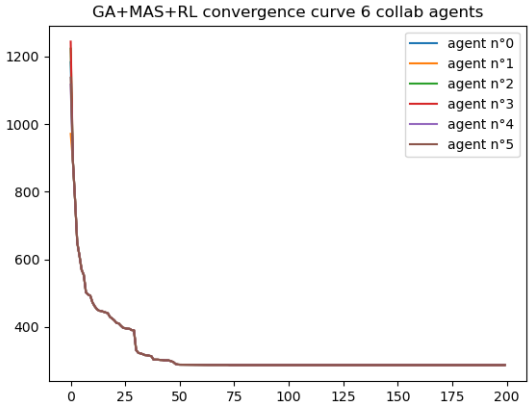


(a) Convergence curve

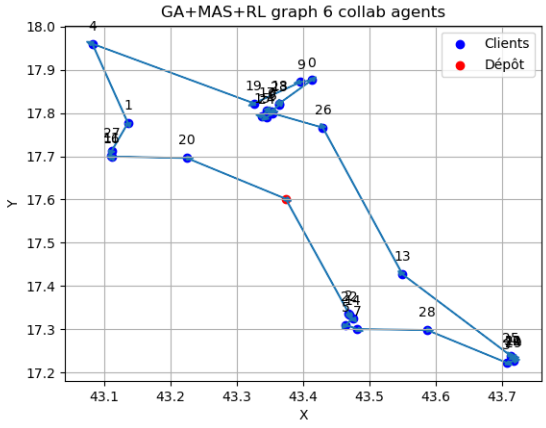


(b) Route graph

Figure 21.: GA + MAS + Reinforcement learning: 6 non-collab agents



(a) Convergence curve



(b) Route graph

Figure 22.: GA + MAS + Reinforcement learning: 6 non-collab agents

7. Conclusion and future work

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.

References

[1] M. P. Seixas. *HEURISTIC AND EXACT METHODS APPLIED TO A RICH VEHICLE ROUTING AND PROGRAMMING PROBLEM*

[2] G. B. DANTZIG AND J. H. RAMSER 2001. *Truck Dispatching Problem*

[3] Fadlah Tunnisaki and Sutarman *Clarke and Wright Savings Algorithm as Solutions Vehicle Routing Problem with Simultaneous Pickup Delivery (VRPSPD)*

[4] Bruce Golden, Xingyin Wang , Edward Wasil *The Evolution of the Vehicle Routing Problem—A Survey of VRP Research and Practice from 2005 to 2022*

[5] Fei Liua, , Chengyu Lu , Lin Gui, Qingfu Zhang , Xialiang Tong , Mingxuan Yuan *Heuristics for Vehicle Routing Problem: A Survey and Recent Advances*

[6] Roberto García-Torres, Alitzel Adriana Macias-Infante, Santiago Enrique Conant-Pablos José Carlos Ortiz-Bayliss Hugo Terashima-Marín*Combining Constructive and Perturbative Deep Learning Algorithms for the Capacitated Vehicle Routing Problem*

[7] Ankan Bose and Dipak Laha *Efficient clustering-based constructive heuristics for capacitated vehicle routing*

[8] Kris Braekers, Katrien Ramaekers, Inneke Van Nieuwenhuyse*The Vehicle Routing Problem: State of the Art and Future Directions*

[9] Gendreau, M., Potvin, J.Y., et al., 2010. *Handbook of metaheuristics. volume 2. Springer*

[10] C. Legrand et al. *New Neighborhood Strategies for the Bi-objective Vehicle Routing Problem with Time Windows*

[11] Golden et al. (1998) *Tabu Search Heuristics for the Vehicle Routing Problem*

[12] Gendreau et al. (2008) *A Tabu Search Algorithm for the Vehicle Routing Problem with Two-Dimensional Loading Constraints*

[13] I. Yusuf et al *Applied Genetic Algorithm for Solving Rich VRP— 958-961*

[14] Joanna Ochelska-Mierzejewska et al *Selected Genetic Algorithms for Vehicle Routing Problem Solving*

[15] Afifi et al *A Simulated Annealing Algorithm for the Vehicle Routing Problem with Time Windows and Synchronization Constraints*

[16] Osman et al *Metastrategy Simulated Annealing and Tabu Search Algorithms for the Vehicle Routing Problem*

[17] Edyvalberty Alenquer Cordeiro, Anselmo R. Pitombeira-Neto *Deep reinforcement learning for the dynamic vehicle dispatching problem: An event-based approach*

[18] Aigerim Bogyrbayeva, Meraryslan Meraliyev , Taukekhan Mustakhov, Bissenbay Dauletbayev *Learning to Solve Vehicle Routing Problems: A Survey*

[19] Ruibin Bai et al*Analytics and Machine Learning in Vehicle Routing Research*

[20] Victor Lesser, Daniel Corkill*Distributed Artificial Intelligence: Theory and Praxis 1980*

[21] Gerhard Weiss *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence 1999*

[22] Yoav Shoham *Agent-Oriented Programming*

[23] Erik Cuevas et al*A new metaheuristic approach based on agent systems principles*

[24] R. Fatimah *Utilization of The Generate and Test Algorithm In Shortest Route Search CaseInternational Journal of Information System & Technology Akreditasi No. 158/E/KPT/2021 — Vol. 5, No. 5, (2022), pp. 541-547*

[25] S. Kirkpatrick, C. D. Gelatt, Jr., M. P. Vecchi *Optimization by Simulated Annealing*

[26] Dimitris Bertsimas; John Tsitsiklis*Simulated Annealing*

[27] B F Skinner*Reinforcement Theory ; 1938 (Process Theory)*

[28] R Bellman *Dynamic Programming (1957)*
[29] Andrew Barto, Richard Sutton, and Charles Anderson *Reinforcement Learning: an introduction*