Dynamic Vehicle Routing Problem With Time Window (DVRPTW): 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 January 31, 2025

ABSTRACT

Following the unprecedented growth of e-commerce and the rise of online shopping, coupled with the surge in food delivery—especially during and after the COVID-19 pandemic—there is now an immense demand for efficient item deliveries. Consumers have shifted their expectations towards faster and more reliable delivery services. Consequently, companies face increasing pressure to enhance their logistics capabilities. Retailers and delivery platforms are constantly innovating to offer improved delivery options, such as same-day and scheduled deliveries, to meet these expectations and remain competitive in the market. These challenges are collectively addressed as the Vehicle Routing Problems (VRP). The aim of this article is to address these challenges by combining three powerful metaheuristic algorithms: the Tabu Search Algorithm, the Genetic Algorithm, and the Simulated Annealing Algorithm, with the concept of multi-agent systems (MAS) using the MESA framework. We will enhance these algorithms by incorporating artificial intelligence through reinforcement learning (RL), which will be applied to the generation of neighbors, the creation of neighborhoods, and the execution of mutations in the population of individuals, all aimed at optimizing the static Vehicle Routing Problem with Time Windows (VRPTW). Additionally, we will leverage the complementary traits of these metaheuristic algorithms by constructing a reinforcement learning environment where an agent learns to choose between these three algorithms at each time step through trial and error. Finally, we will address the dynamic nature of VRPTW in real-life scenarios by studying its complexity, exploring real-time decision-making through constant data flows, and examining its diverse applications.

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 optimization problems [1]. It involves finding a set of routes that start and end at a depot, collectively covering a set of customers. Each customer has a specific demand, and no vehicle can service more customers than its capacity permits. The objective is to minimize the total distance traveled and optimize the number of vehicles dispatched. This problem originated with Dantzig and Ramser's research on the Truck Dispatching Problem, the predecessor of VRP, in 1959 [2]. Initially, it was known as the "Truck Dispatching Problem," modeling the traffic of a fleet of trucks serving several gas stations. The works of G. Clarke and J.R. Wright expanded upon this topic, leading to the generalization of the "Truck Dispatching Problem" with the introduction of new approaches, such as the "savings algorithm" proposed by Clarke and Wright in 1964. This heuristic method significantly improved the practical applicability of VRP solutions [3].

The VRP has a variety of variations, such as:

- Capacitated VRP: Considers the capacity of each dispatched vehicle.
- Multi-Depot VRP: Considers the possibility of having multiple depots.
- Green VRP: Emphasizes minimizing environmental impact.
- VRP with Time Windows: Incorporates time windows for deliveries [3, 4].

This work focuses on the latter variation. In the study titled *Heuristics for Vehicle Routing Problem: A Survey and Recent Advances* [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: Unlike constructive and improvement heuristics, metaheuristics are not problem-dependent and are generally inspired by natural phenomena [9].

In this study, our approach is based on the use of metaheuristics. Among the variety of metaheuristic algorithms, we focused on the tabu algorithm [11, 12, 16], the genetic algorithm [13, 14], and the simulated annealing algorithm [15, 16]. While we concentrated on these methods, adding more algorithms, such as the ant colony algorithm, could be interesting for further research.

Thanks to modern advances in computational power and algorithmic techniques, significant strides have been made in VRP. 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 for the revolutionizing of VRP [17, 18, 19]. A particular method, known as Multi-Agent Systems (MASs), which traces its roots back to the early developments of AI and distributed problem-solving, has found success in VRP. From the 2000s onwards, with the multitude of research studies on agent-based algorithms and agent-oriented programming, MAS has become integral in various domains, including solving complex problems beyond the capacity of a single agent [20, 21, 22].

The concept of MAS is advantageous in solving VRP problems because it creates an environment where multiple agents work collaboratively on optimizing the same solution. This allows for interaction in various ways, making this approach flexible and likely to produce excellent results [23]. The primary goal of this work is to combine metaheuristic algorithms with MAS by constructing an environment where agents, representing different metaheuristic algorithms, interact to optimize a solution while learning from their experiences through reinforcement learning.

Treating the VRPTW problem as a static problem assumes that all information relevant to route planning is known by the planner before the routing process begins and that this information does not change after the routes have been constructed. This is a simplistic approach, considering that, in reality, there is a constant flow of data that must be accounted for in decision-making [30]. Therefore, this study considers the fact that information can change after the initial routes have been constructed.

Complexity of VRP problems

Let's consider the basic vehicle routing problem. Its complexity 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

To address this NP-Complete problem, we adopt a collaborative optimization approach that leverages the strengths of various algorithms and techniques. Specifically, we set constraints that allow for the utilization of locally optimal solutions when global optimality is unattainable, ensuring practical applicability in real-world scenarios.

Our approach integrates multiple optimization methods, including the Tabu Search algorithm, Simulated Annealing algorithm, and Genetic algorithm. These algorithms collaborate within a Multi-Agent System (MAS), where each algorithm functions as an autonomous agent. This collaboration enhances the search process, combining the unique strengths of each method.

Furthermore, the integration of Reinforcement Learning (RL) significantly improves the generation of solution neighborhoods. The combination of MAS and RL guides the search towards more promising areas and mitigates the blind spots inherent in traditional metaheuristics when generating neighborhoods. This adaptive learning mechanism ensures a more directed and efficient search process.

Additionally, our approach is highly scalable and evolutionary, allowing for the creation of new agents incorporating other metaheuristics. This enrichment of the collaborative framework accelerates convergence towards the globally optimal solution, enhancing both the speed and robustness of problem-solving.

This approach is particularly beneficial for solving the Dynamic Vehicle Routing Problem with Time Windows (DVRPTW), especially when urgent orders need to be delivered and when traffic conditions are congested. By dynamically adjusting routes in response to real-time traffic data and urgent delivery requests, our system can identify and utilize alternative routes, ensuring timely deliveries and optimal resource utilization.

The innovative combination of collaborative multi-agent optimization and Reinforcement Learning offers a powerful framework for tackling complex NP-Complete problems like DVRPTW. It provides practical solutions that balance local and global optimization considerations, continuously evolve to improve performance, and adapt to dynamic, real-world conditions, ensuring high efficiency even under challenging scenarios.

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*

- C: the set of indices of nodes corresponding to clients (orders).
- s: index of the depot.

- V: the set of indices of vehicles belonging to a warehouse.
- A: the set of edges in the graph.
- $C_D, \quad C_H, \quad C_J$: coefficients in the trinomial formula: the distance coefficient, the hourly coefficient, and the daily coefficient, respectively.
- V_{moy}: the average speed of a delivery vehicle.
- $D_{ij}, \quad \forall (i,j) \in A$: the distance between nodes i and j.
- Q_{max}: the maximum capacity of a vehicle (in terms of number of pallets, packages, etc.).
- $Q_i \quad \forall i \in C$: the quantity (number of pallets, packages, weight, volume, etc.) of order i.
- $DC_i \quad \forall i \in C$: the loading date for order i (the date before which no delivery should be initiated).
- $DL_i \quad \forall i \in C$: the latest delivery date for order i (not to be exceeded).
- M: a positive variable with a large value $M \gg 1$.

2.2. *Decision variables*

- $u_{ij}^k, \quad \forall (i,j) \in A, k \in V$: Boolean variable specifying whether the edge (i,j) in the graph is traversed by vehicle k.
- $\kappa_k, \quad \forall k \in V$: Boolean variable indicating whether vehicle k is activated to make deliveries.
- $q_i^k, \quad \forall i \in A, k \in V$: Positive real variable representing the quantity of goods transported by vehicle k through node i.
- $s_i^k, \quad \forall i \in A, k \in V$: Positive real variable corresponding to the elapsed time for delivering the parcel at node i (client or order) in hours, calculated from the start of the period (31/12/2020, 0h).
- $\delta_k, \quad \forall k \in V$: Boolean variable indicating the departure period of vehicle k : $\delta_k = 0$ for departure during the first period (5h to 13h) or $\delta_k = 1$ for departure during the second period (13h to 21h).

2.3. *MILP model*

$$
\begin{aligned}
\min \quad & f = \sum_{k \in V} [(C_D + \frac{C_H}{V_{moy}}) \sum_{i,j \in A} D_{ij} u_{ij}^k + C_J \kappa_k] & & \\
\text{s.t.} \quad & u_{ij}^k, \kappa_k, \delta_k \in \{0,1\} & \forall (i,j) \in A, k \in V, \quad & (1) \\
& s_i^k \geq 0 & \forall i \in A, k \in V, \quad & (2) \\
& q_i^k \geq 0 & \forall i \in A, k \in V, \quad & (3) \\
& u_{ij}^k \leq \kappa_k & \forall (i,j) \in A, k \in V, \quad & (4) \\
& u_{ii}^k = 0 & \forall i \in A, \quad & (5) \\
& \sum_{k \in V} \sum_{j \in A} u_{ij}^k = 1 & \forall i \in C, \quad & (6) \\
& \sum_{j \in A} u_{ij}^k = \sum_{j \in A} u_{ji}^k & \forall i \in A, k \in V, \quad & (7) \\
& \sum_{i,j \in A} u_{ij}^k \leq M \sum_{i \in C} u_{si}^k & \forall k \in V, \quad & (8)
\end{aligned}
$$

$$DC_i \leq 6(1-\delta_k) + 14\delta_k + M \cdot (1 - \sum_{k\in V}\sum_{j\in A} u_{ij}^k) \quad \forall i \in C, \tag{9}$$

$$q_s^k = Q_{max} \quad \forall k \in V, \tag{10}$$

$$q_i^k - Q_i - q_j^k \leq M \cdot (1 - u_{ij}^k) \quad \forall i \in A, j \in C, k \in V, \tag{11}$$

$$q_j^k - (q_i^k - Q_i) \leq M \cdot (1 - u_{ij}^k) \quad \forall i \in A, j \in C, k \in V, \tag{12}$$

$$s_s^k = 6(1-\delta_k) + 14\delta_k \quad \forall k \in V, \tag{13}$$

$$s_j^k - s_i^k - \frac{D_{ij}}{V_{moy}} \leq M(1 - u_{ij}^k) \quad \forall (i,j) \in A, k \in V, \tag{14}$$

$$s_i^k + \frac{D_{ij}}{V_{moy}} - s_j^k \leq M(1 - u_{ij}^k) \quad \forall (i,j) \in A, k \in V, \tag{15}$$

$$s_i^k \leq DL_i \quad \forall i \in A, k \in V \tag{16}$$

2.4. *Objective function*

$$OF = Min(\sum_{k\in V}[(C_D + \frac{C_H}{V_{moy}}) \sum_{i,j\in A} D_{ij}u_{ij}^k + C_J\kappa_k]) \tag{17}$$

The objective function (17) corresponds to the minimization of several criteria. The first criterion is the cost of the total distance traveled by all dispatched vehicles. The second reflects the total cost of dispatching a certain vehicles.

2.5. *Constraints*

- **Constraint 4:** This constraint ensures that a client can only be served by a vehicle that is actually assigned to a route.
- **Constraint 5:** A vehicle must necessarily go to another node in the graph after leaving a given point.
- **Constraint 6:** Every order must be delivered.
- **Constraint 7:** If a vehicle k visits a client i, it must necessarily leave that client and proceed to another node in the graph.
- **Constraint 8:** If vehicle k does not depart from warehouse s on day t, no client should have their package delivered by that vehicle.
- **Constraint 9:** A delivery can only begin after the scheduled loading date. If t is earlier than this date, order i cannot be included in a route.
- **Constraint 10:** When a vehicle passes through a warehouse, it automatically reloads its cargo with packages or pallets.
- **Constraints 11/12:** The remaining quantity in a vehicle after serving client j is equal to the available quantity after serving the previous client i, reduced by the demand of client i.
- **Constraint 13:** The delivery departure time àf vehicle k.
- **Constraints 14/15:** These constraints allow the calculation and addition of travel times between the various clients visited.
- **Constraint 19:** This constraint enforces compliance with delivery time windows.

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 1: 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 2: 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 $bestSol$
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;
7  while (the desired objective function not yet obtained) or ($nbIter < maxIter$)
     do
8      Increment $nbIter$;
9      Generate a neighbor ;
10     Split the neighbor to the optimal number of vehicles ;
11     Compute the energy difference $delta = E_{new} - E_{init}$ ;
12     generate a random value $0 \leq r \leq 1$;
13     if ($delta < 0$) then
14         update $sol$ and $bestSol$ as the generated neighbor;
15         Update $bestIter$ as $nbIter$
16     end
17     else if ($r \leq e^{-delta/T}$) then
18         update sol as the generate neighbor
19     end
20     Update $sol$ as $bestNeighbor$;
21     Merge all sequences of all dispatched vehicles in $sol$;
22     Split the new sequence into the different possible numbers of vehicles to be
         sent;
23     Update $sol$ as the best solution out of the split sequences;
24     if ($nbIter - bestIter \geq iter$) then
25         increase temperature to get out of local optimum
26     else
27         Decrease current temperature by multiplying it by the decay rate $alpha$
28     end
29 end
30 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 3: 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
 1 Begin
 2 Initialization : generating an initial population;
 4 while (a fixed number of generations is not reached) do
 5     Evaluate individuals ;
 6     Parent selection for reproduction ;
 7     Apply a controlled crossover algorithm, in order to obtain viable offspring1
         and offspring2 ;
 8     Apply a controlled mutation algorithm with a probability mutationRate ;
 9     Update the population by changing the best individual by optimizing the
         number of routes after a merge and a split approach;
10 end
11 end
```

4. Multi-agent systems

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

Definition:

A Multi-Agent System (MAS) is a system that defines an environment containing multiple agents, specifying the rules of interaction between them. These interactions are defined in the MAS through a class method called `contact`. This method enables collaborative agents to exchange data with other agents if their optimized solutions at time step t offer a better objective function value.

In our implementation, we modeled a MAS with agents representing metaheuristic algorithms. To achieve this, we used the `mesa` package for Python programming. With this package, an MAS is defined by a model class and one or more agent classes in Python.

- Agent Class: This class defines the `step` method, which contains the metaheuristic algorithm, and the `contact` method, which handles agent interactions.
- MAS Model: This class initializes the desired agents in our MAS environment, selects the type of `scheduler` (the method of activity scheduling for different agents; in our case, we used `SimultaneousActivation` since it allows the algorithms to execute simultaneously at each time step), and collects relevant data to visualize 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 with a single agent 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 the Tabu algorithm and Simulated Annealing) and functions that execute mutations (in the case of the Genetic Algorithm). The purpose of this section is to enhance these functions using a technique called Reinforcement Learning (RL), thereby adding intelligence to the actions taken by the agents in the Multi-Agent System (MAS). We define an RL environment that enables the iterative optimization of selecting the most suitable MAS+RL algorithm at each time step.

Algorithm 4: 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 (RL) is a machine learning technique based on the notion of trial and error. It is inspired by behavioral psychology [27], where an agent learns to make decisions by taking actions in an environment to maximize a cumulative reward. This approach involves executing multiple actions and learning from mistakes and successes through the concept of rewards. The foundation of RL 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 discrete time step t. This interaction is summarized by three key pieces of 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:

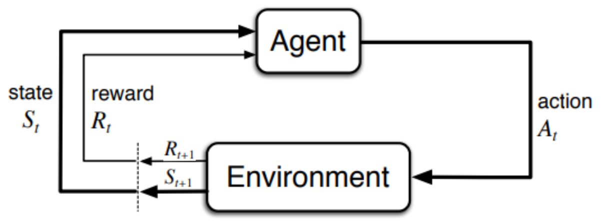


Figure 1.: agent–environment interaction [29]

The states that we considered are S ∈{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.

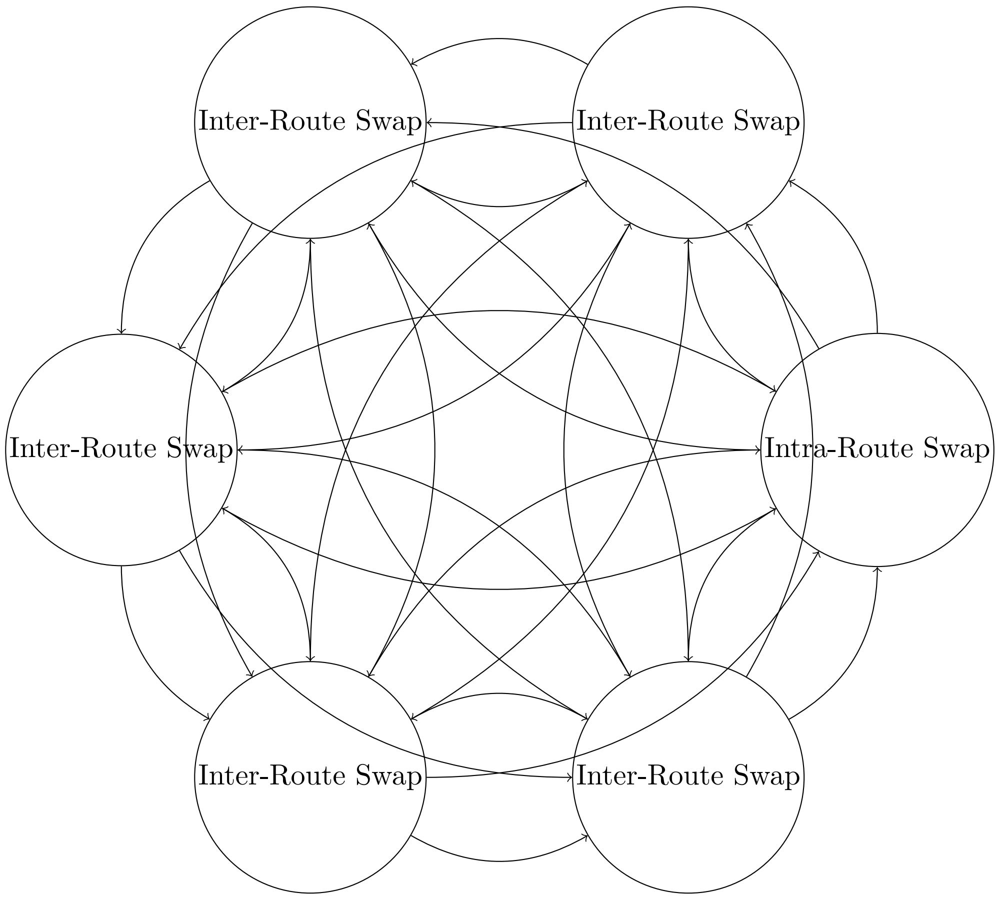
(3) Intra-Route Shift: Neighborhood function that performs moving a customer to another position on the same road.
(4) Inter-Route Shift: Neighborhood function that performs moving a customer from one route to another.
(5) Two Intra-Route Swap: neighborhood function that consists of the exchange of customers 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) can be modeled as a Markov Decision Process (MDP), where:

- S is a set of possible states of the agent in its environment.
- A is the set of different actions the agent can take.
- P is a state transition probability matrix.
- R is the reward function.

In this MDP framework, the future state, action, and reward depend only on the current state of the agent/environment, satisfying the Markov property. Thus, the different transitions can be represented by the following graph:



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

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 is a value-based method, meaning its policy is implicit and not directly optimized. Instead, it optimizes the action-value function, denoted as q.

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

In this algorithm, the policy consists of a **Policy choosing algorithm** that determines the next action to take based on the current state. Its algorithm is as follows:

Algorithm 5: Policy choosing algorithm

```
input  : current state and the type of function to apply;
         Q matrix;
         epsilon greedy rate($\epsilon$);
output: Next state to visit
1 Begin $next\_state \leftarrow 0$ ;
2 if $type\_function == 1$ then
3     $next\_state \leftarrow RandomAction()$;
4 else
5     $next\_state \leftarrow \epsilon - greedy(state, Q, \epsilon)$;
6 end
7 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).

Algorithm 6: $\epsilon-$greedy algorithm

```
input  : current state ;
         Q matrix;
         epsilon parameter;
output: Next state to visit
1 Begin
2 generate a random number r in [0,1] ;
3 if $r \leq \epsilon$ then
4     $next\_state \leftarrow RandomAction()$;
5 else
6     Choose the action with the least Q-value;
7 end
8 end
```

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 $\lambda$;
         Discount rate $\gamma$;
         epsilon greedy rate $\epsilon$;
         Learning rate $\alpha$;
output: Next state to visit
1  Begin
2  initialise Q;
3  $improved \leftarrow 0$;
4  $no\_improvement \leftarrow 0$;
5  $x\_best \leftarrow 0$;
6  $x \leftarrow x_0$;
7  $reward \leftarrow 0$;
8  initialise empty list of neighbors;
9  $iter \leftarrow 0$;
10 $next\_state \leftarrow choose\_action(0, 2, Q, \epsilon)$ ;    ▷ 2: initial state, prioritize random
     policy
11 while $iter \leq nb_max$ do
12     iter++;
13     if $no\_improvement == 0$ then
14         $state \leftarrow next\_state$; $next\_state \leftarrow choose\_action(state, 1, Q, \epsilon)$;    ▷ #1:
             epsilon greedy function
15     else
16         $next\_state \leftarrow choose\_action(0, 2, Q, \epsilon)$
17     end
18     $x, re \leftarrow apply\_action(x_0, next\_state)$;
19     if $Objective\_function(x) \leq Objective\_function(x\_best)$ then
20         $x\_best \leftarrow x$
21         $no\_improvement \leftarrow 0$
22     else
23         $no\_improvement \leftarrow 1$
24     end
25     $reward \leftarrow reward + re$
26     $a\_prime \leftarrow Max\_Arg(Q[next\_state])$ ▷ choose the action a_prime that most
         improves the learning
27     $Q[state, next\_state] \leftarrow Q[state, next\_state] + \alpha * (reward + \gamma *$
         $Q[next\_state, a\_prime] - Q[state, next\_state])$
28     $\epsilon \leftarrow \epsilon \times \lambda$
29     add x to list of neighbors
30 end
31 End
```

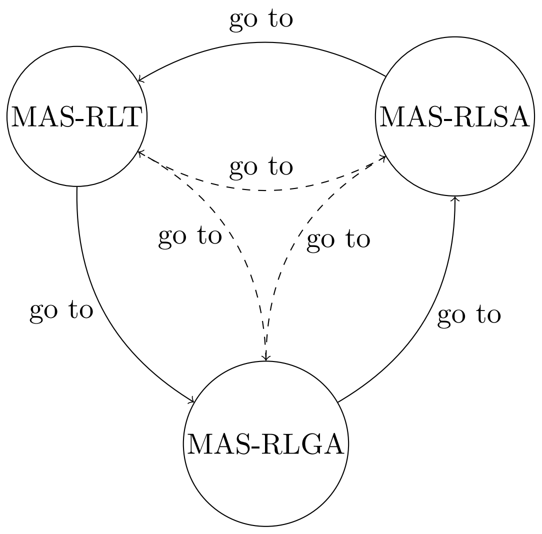
As for SA agents, the reasoning is the same: they take actions based on what has been previously experienced. However, some changes to the Adaptive Search Algorithm were necessary. In fact, this new version of the Q-learning algorithm takes as input additional data: the agent's Q-learning matrix, its epsilon-greedy parameter, and a parameter indicating if it has improved or not, denoted as $no_improvement$. The reason behind this is that SA agents need only one neighbor at each iteration, so they need to remember the data resulting from previously generated neighbors to learn from their trial-and-error experiences. In contrast, the Tabu and GA agents' Adaptive Search Algorithms iterate multiple times (more than 50 times) before generating the desired neighborhood. Therefore, when a Tabu agent generates a new neighborhood, it doesn't need to remember the previously generated data, as it will learn sufficiently through a large number of iterations.
The Q-matrix used in the Adaptive Search Algorithm is a 6-by-6 matrix, with six states as rows and six actions as columns. Each coefficient of the matrix (initialized to zero) represents the quality of an action based on past experiences. In other words, when exploiting in the current state S, the agent takes the action with the highest quality, i.e., the highest Q-value in the Q-matrix for the row corresponding to 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 $\epsilon$;
        reward;
        learning rate $\alpha$;
        decay rate$\lambda$;
        discount rate $\gamma$ ;
        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 $\epsilon$;
        Updated $no_improvement$;
        Updated reward;
1  Begin
2  $x \leftarrow x_0$;
3  if $no\_improvement == 0$ then
4      $next\_state \leftarrow choose\_action(state, 1, Q, \epsilon)$ ;
5                                          ▷ #1: epsilon greedy function
6  else
7      $next\_state \leftarrow choose\_action(state, 2, Q, \epsilon)$
8  end
9  $x, re \leftarrow apply\_action(x_0, next\_state)$;
10 $reward \leftarrow reward + re$
11 $a\_prime \leftarrow Max\_Arg(Q[next\_state])$    ▷ choose the action a_prime that most
     improves the learning
12 $Q[state, next\_state] \leftarrow Q[state, next\_state] + \alpha * (reward + \gamma *$
     $Q[next\_state, a\_prime] - Q[state, next\_state])$
13 $\epsilon \leftarrow ep \times decay\_rate$
14 if $Objective\_function(x) \leq Objective\_function(x_0)$ then
15     $no\_improvement \leftarrow 0$
16 else
17     $no\_improvement \leftarrow 1$
18 end
19 End
```

5.3. *Reinforcement Learning: Alternating between MAS algorithms*

Following the last section, we now have MAS algorithms with intelligent agents. In this section, we will refer to MAS with intelligent Tabu agents as MAS-RLT, MAS with intelligent simulated annealing agents as MAS-RLSA, and MAS with intelligent genetic algorithm agents as MAS-RLGA. As demonstrated in the next section, simulations of MAS+RL agents reveal complementary behaviors among these algorithms. The objective of this section is to build a reinforcement learning environment where an agent learns to choose between MAS with intelligent agents: MAS-RLT, MAS-RLSA, and MAS-RLGA.
Let (S, A, P, R) denote the Markov Decision Process (MDP) for the RL framework enabling iterative alternation between MAS algorithms. includes executing MAS-RLT, MAS-RLSA, or MAS-RLGA at each time step. The set of actions A consists of transitions between these states. These actions can be represented by the following graph:



The decision policy follows the same approach as developed in the previous section. For the Q-learning algorithm, it utilizes the same method as the Adaptive Q-learning Algorithm used for Tabu and GA agents. The only difference lies in the actions that can be taken.

6. The Dynamic VRPTW

In real-life situations, Vehicle Routing Problems (VRP) are dynamic as information gets updated over time. This information can include:

- **Dispatched Vehicle Information**: The dispatcher must continuously track the position of all vehicles. In the event of urgent issues, such as the likelihood of late deliveries due to traffic jams, the dispatcher can take proactive measures, such as deploying delivery drones.
- **Available Vehicles Information**: Due to dispatch schedules, maintenance, and the limited number of vehicles, real-time tracking of available vehicles is crucial in dynamic VRPTW scenarios.
- **Client Information**: Delivery requests are not static and can vary. While probabilistic or machine learning models can forecast these requests, they cannot be

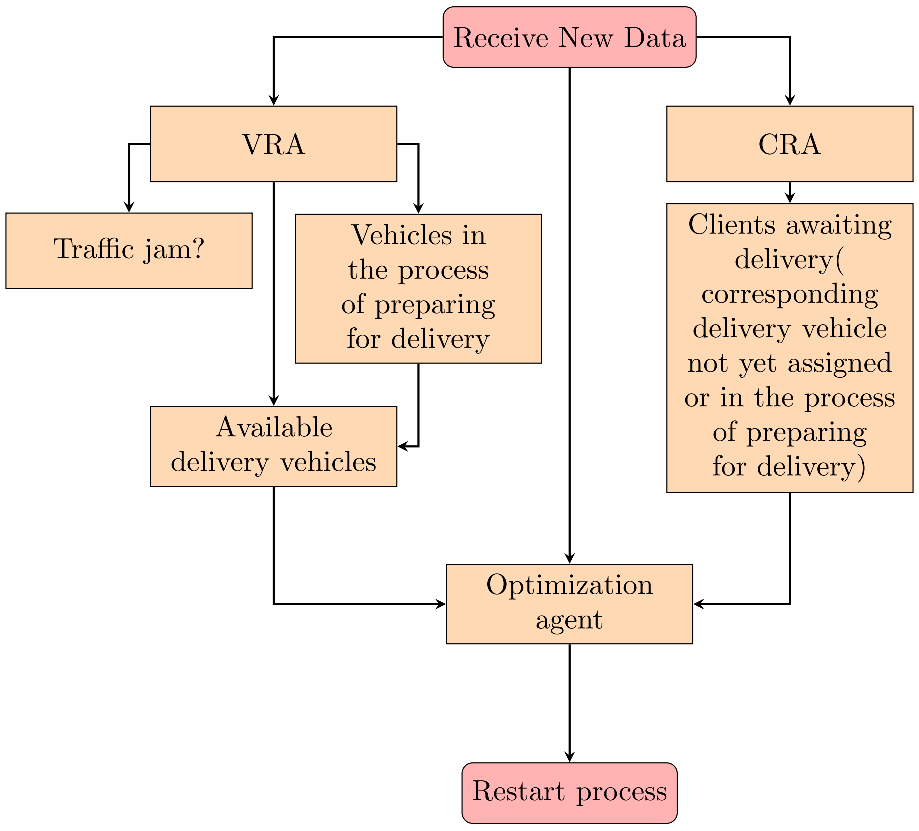
predicted with absolute certainty. In dynamic routing problems, the future is almost never known with certainty.

The constant flow of data and its impact on decision-making increases the complexity of the problem. Given that the static VRPTW is an NP-hard problem, and since static VRPTW problems are a subset of dynamic VRPTW problems, we conclude that DVRPTW is also NP-hard.

To simulate DVRPTW, we use dynamic programming by dividing the complex structure of this problem in sub-problems. Our approach consists of implementing an MAS environment with multiple agents, each having its own role to play in simulating the DVRPTW. These agents are as follows:

(1) Client Request Agent (CRA) : This agent receives delivery requests related to agents.
(2) Vehicle Request Agent (VRA) : This agent receives requests related to delivery vehicles. These requests can be either updates on the availability of certain vehicles or the progress of a current delivery.
(3) Optimization Agent (OA) : This agent's role consists of applying the static VRPTW optimization algorithm developed in this article on the data collected by CRA and VRA.

The simulated environment consist of a DVRPTW in which data updates occur every T time unit, T being the average computing time needed by the optimization algorithm to find an optimized feasible solution. The following flow chart represents the course of action taking place in the MAS environment.


Receive New Data
VRA
CRA
Traffic jam?
Vehicles in the process of preparing for delivery
Clients awaiting delivery(corresponding delivery vehicle not yet assigned or in the process of preparing for delivery)
Available delivery vehicles
Optimization agent
Restart process

7. Simulation results

7.1. *Metaheuristics*

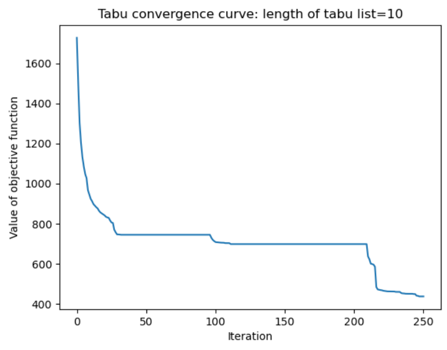
7.1.1. Tabu algorithm

The VRPTW Tabu algorithm has several parameters with significant impact on both convergence time and the optimality of the final feasible solution. These parameters are:

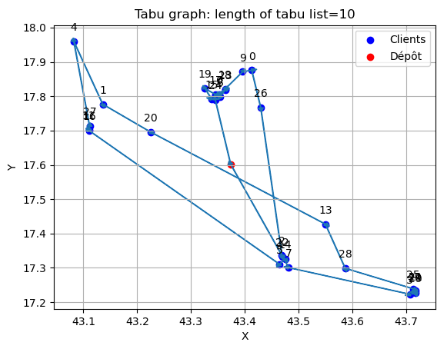
(1) The maximum number of iterations without finding a new best solution (Aspiration): This parameter serves as the first aspiration criterion.
(2) The limit objective function: This designates a value of our objective function that, if achieved, signals the stopping condition for the algorithm.
(3) The maximum size of the tabu list: This parameter sets the maximum size of the tabu list. When this size is reached, the oldest move or solution is removed from the list. Surprisingly, this parameter has a substantial impact on the optimized solution found by the algorithm, as illustrated in Figure 2 ;

    From these tests, we observe that a higher maximum length of the tabu list leads to faster stagnation (premature convergence) and longer times to escape local optima. Setting a large value for this parameter can decrease the algorithm's efficiency, requiring significant computing time and/or power to find the optimal solution. Conversely, smaller tabu list sizes lead to rapid convergence but may compromise the quality of the solution. Therefore, a balance must be struck between algorithmic efficiency and optimization speed.
(4) The number of neighbors to generate in each iteration: In each iteration, we generate a specific number of neighbors for our solution. This neighborhood generation can be achieved through various methods:
    - Generate neighbors randomly: This method yields suboptimal results and does not significantly improve the solution.
    - Generate neighbors by swapping clients in a particular route: This method performs better than the random approach but still confines clients to their initial routes, potentially limiting the solution's optimality.
    - Generate neighbors by applying intra-route and inter-route swapping: This approach enables the algorithm to explore a broader solution space, enhancing its ability to reach the desired objective function.

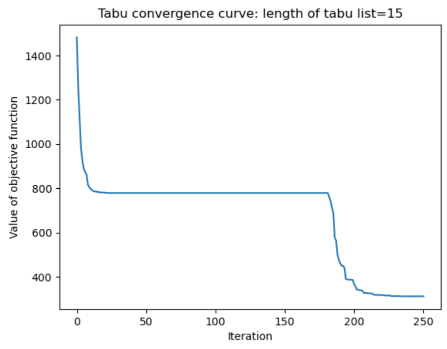
    The graphs and convergence curves in Figure 3 demonstrate that a higher number of neighbors results in faster convergence and improved final solutions. However, increasing the number of neighbors also raises the algorithm's computational cost, necessitating a trade-off. We chose to use 100 neighbors for the rest of the study.



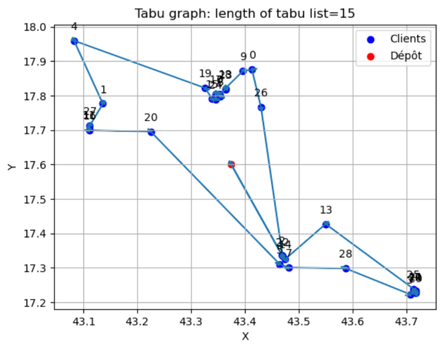
(a) Tabu maximum length of tabu list = 10 convergence curve



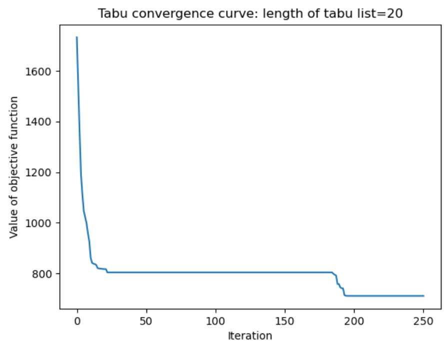
(b) Tabu maximum length of tabu list = 10 routes graph



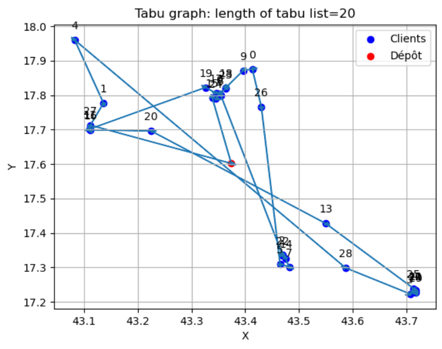
(c) Tabu maximum length of tabu list = 15 convergence curve



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

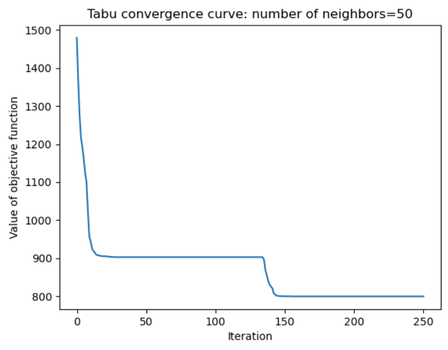


(e) Tabu maximum length of tabu list = 20 convergence curve

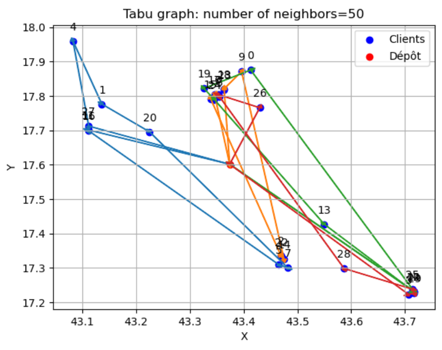


(f) Tabu maximum length of tabu list = 7 routes graph

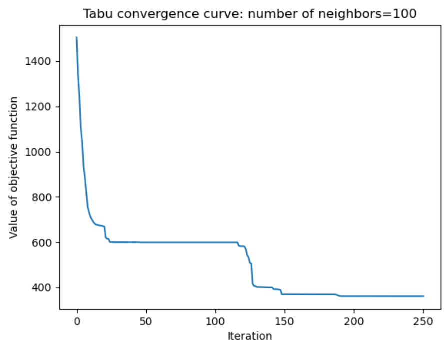
Figure 2.: Impact of the tabu list's maximum size on the algorithm



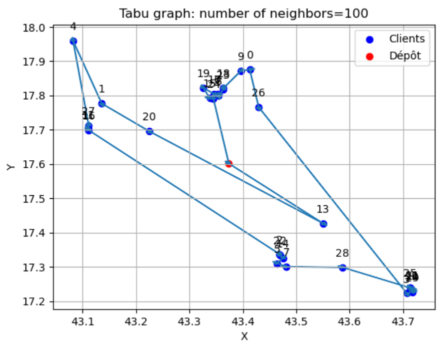
(a) Tabu number of neighbors = 50 convergence curve



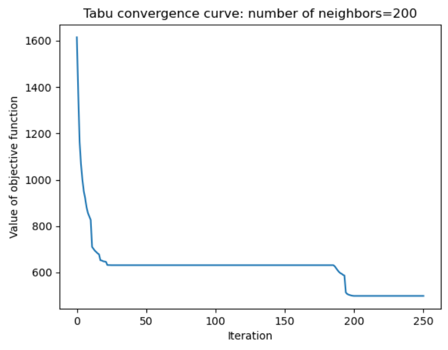
(b) Tabu number of neighbors = 50 routes graph



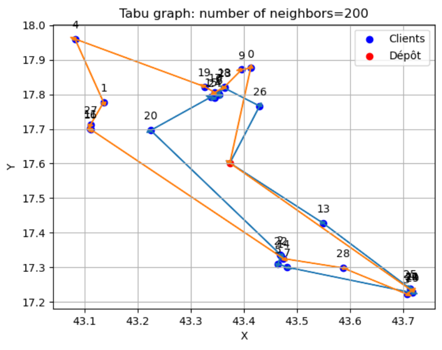
(c) Tabu number of neighbors = 100 convergence curve



(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

7.1.2. Genetic algorithm

The genetic algorithm also has several key parameters that influence its performance:

(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.

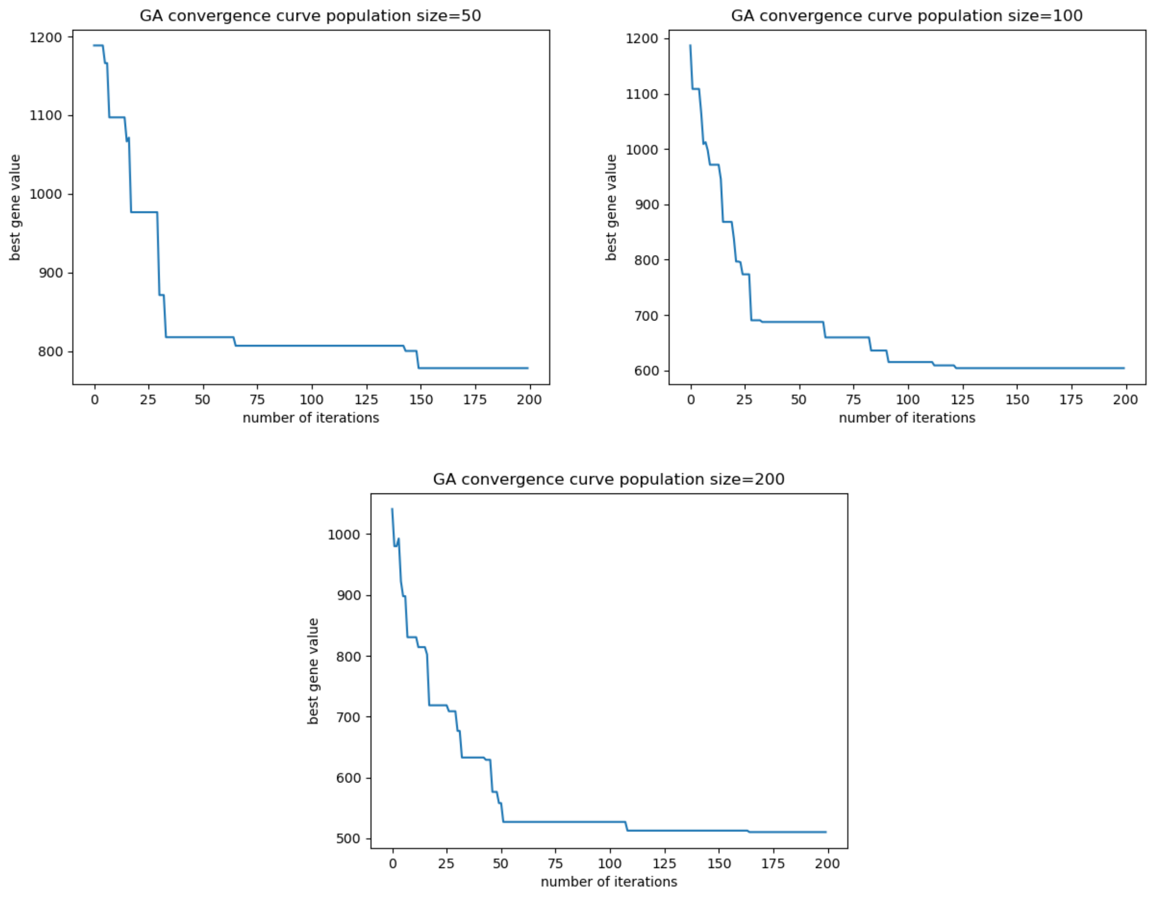


Figure 4.: Impact of population size parameter on the convergence curve

(2) This parameter represents the total number of iterations the algorithm will execute. In each iteration, a new generation of solutions is created through crossbreeding and mutation. While this parameter initially plays a significant role in the algorithm's performance, its impact diminishes once the algorithm reaches a point of stagnation. Our tests indicated that for a problem involving 30 clients, 200 iterations are sufficient for the algorithm to converge, as it tends to get stuck in local optima beyond this point. Therefore, increasing the number of generations beyond this threshold does not lead to significant improvements in the solution quality.

(3) The Mutation Rate: This parameter is crucial in genetic algorithms as it determines the frequency at which mutations occur in the genetic material of the

solutions. It essentially dictates how often the algorithm introduces genetic variations by generating mutated genes. Testing different values for this parameter yields various convergence curves, as shown in Figure 5. Adjusting the mutation rate allows us to balance exploration and exploitation, where a higher mutation rate encourages exploration of new solution spaces, and a lower rate focuses more on exploiting the current solution's neighborhood.

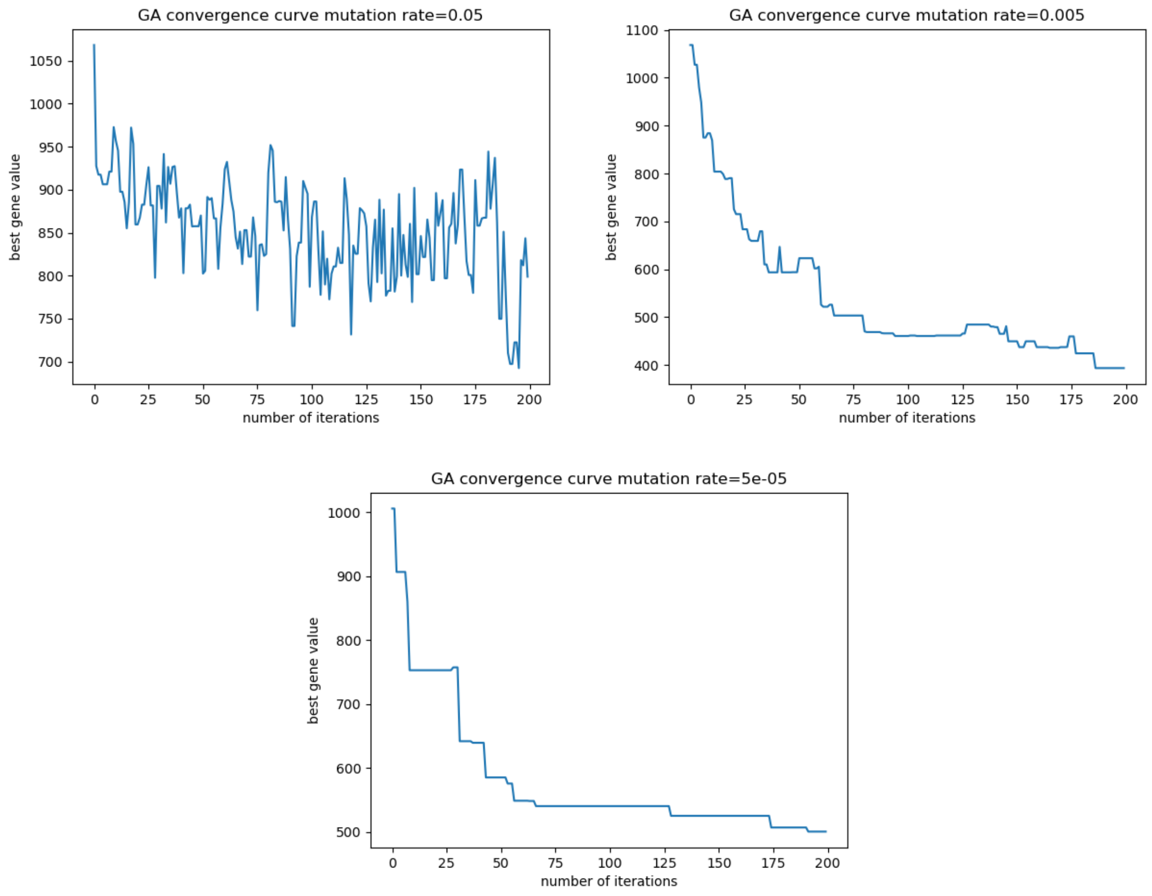


Figure 5.: impact of mutation rate

7.1.3. *Simulated Annealing algorithm*

The simulated annealing algorithm requires several parameters as inputs: a decay rate called α, a minimum temperature, a maximum temperature, and a maximum number of iterations. These parameters are used to control the algorithm's execution and stopping conditions. The minimum temperature, maximum temperature, and maximum iterations serve to limit the runtime of the algorithm. Specifically, the algorithm begins at the maximum temperature, and as the system's current temperature decreases due to the decay rate α, the algorithm continues until it reaches the specified minimum temperature. Additionally, if the number of iterations exceeds the maximum number specified, the algorithm will stop.

The decay rate, α, is a critical parameter that must be carefully tuned to ensure the algorithm produces good results. Below (Figure 12)are graphs and convergence curves illustrating the impact of this parameter.

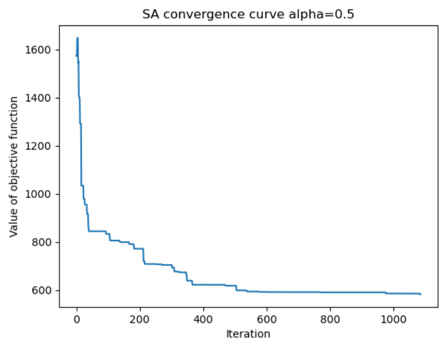


Figure 6.: convergence curve for decay rate alpha=0.5

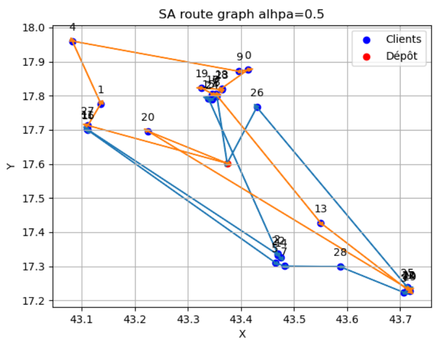


Figure 7.: Route graph for decay rate alpha=0.5

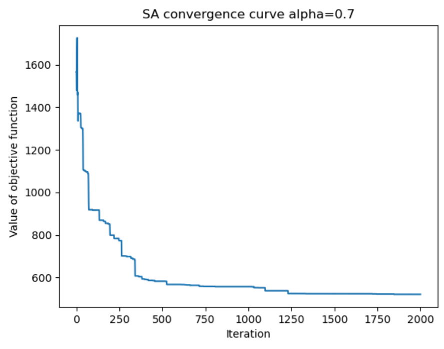


Figure 8.: convergence curve for decay rate alpha=0.7

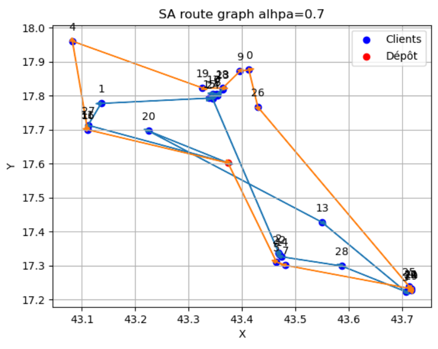


Figure 9.: Route graph for decay rate alpha=0.7

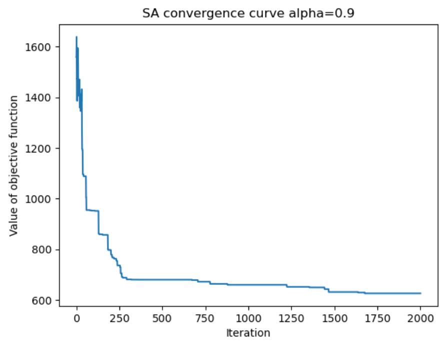


Figure 10.: convergence curve for decay rate alpha=0.9

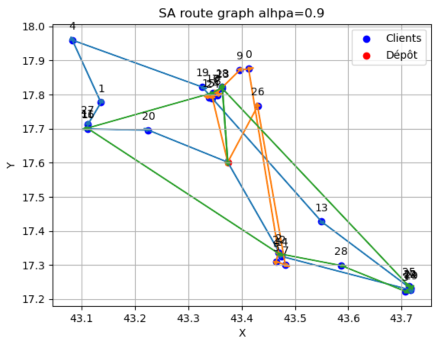


Figure 11.: Route graph for decay rate alpha=0.9

Figure 12.: Impact of the deacy rate on SA

These convergence curves show that as we increase the decay rate $\alpha$of the temperature, the algorithm converges more quickly. This is due to the fact that a high decay rate rapidly decreases the temperature T, which results in a low value for $e^{\frac{-\Delta E}{T}}$. Consequently, the likelihood of replacing the current solution with a neighboring solution decreases. In contrast, when T is high, the algorithm is more prone to explore less optimal solutions due to the higher acceptance probability of worse neighbors. However, using a high decay rate α increases the likelihood that the algorithm will accept a worse solution in terms of its objective function (i.e., energy). By setting α=0.995 as the input to the algorithm, we observe the following convergence curve and route graph.

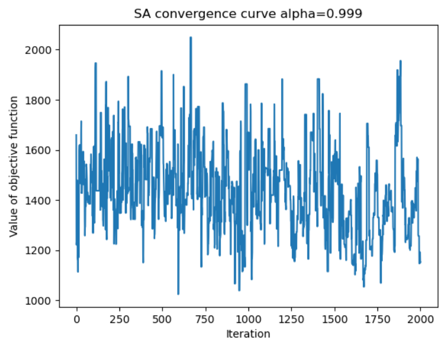


Figure 13.: convergence curve for decay rate alpha=0.999

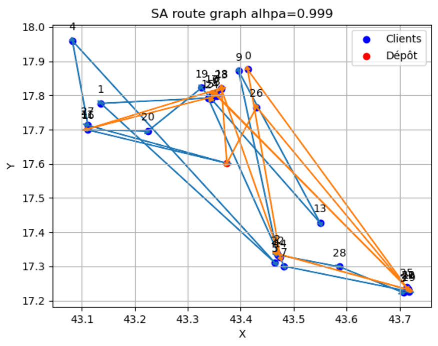
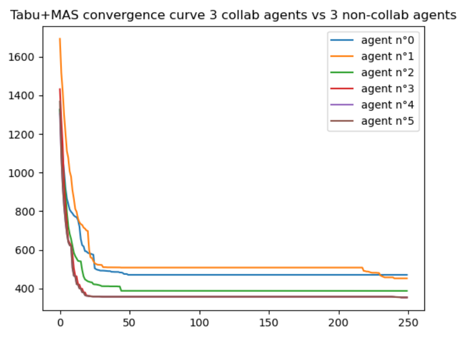


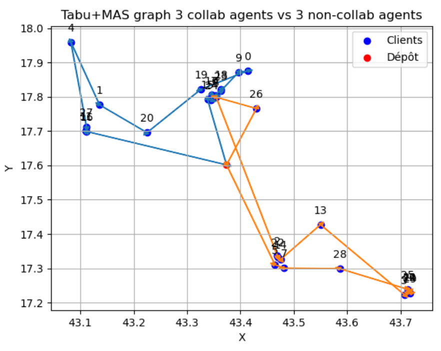
Figure 14.: Route graph for decay rate alpha=0.999

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

The simulation results for each metaheuristic algorithm demonstrated that even with careful parameter tuning, certain challenges persisted. These included premature convergence in the Tabu algorithm, unpredictable behavior in Simulated Annealing (SA), and the requirement for a relatively large number of iterations in the Genetic Algorithm (GA). The rationale behind using Multi-Agent System (MAS) environments is to facilitate collaboration among these algorithms, as their behaviors complement each other. Specifically, the Tabu algorithm's tendency to stagnate and SA's unpredictability can be mitigated by GA's steady behavior, while GA's optimization process can be accelerated by the fast convergence of Tabu and SA.

By first implementing MAS environments where agents are restricted to a single algorithm, we observed significant improvements in results. For instance, when Tabu agents collaborated, simulations showed that although stagnation still occurred relatively early, the agents were able to further enhance the final solution, as illustrated in Figure 15.



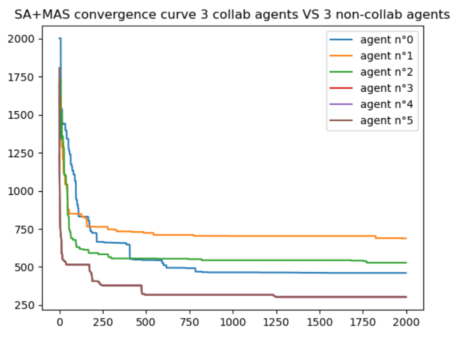


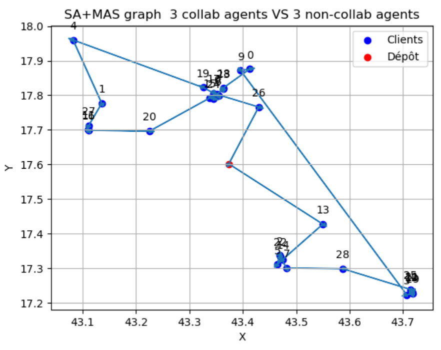
(a) Convergence curve

(b) Route graph

Figure 15.: 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) exhibit remarkably faster convergence compared to their non-collaborative counterparts (agents 0 to 2). For Simulated Annealing (SA) agents, the collaborative agents were able to rapidly optimize the solution compared to the non-collaborative ones, as demonstrated in Figure 16.



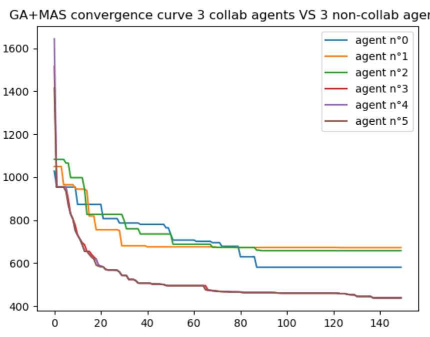


(a) Convergence curve for 3 collaborative agents and 3 non-collab agents

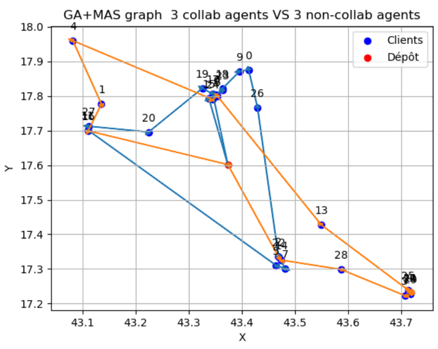
(b) Graph of routes for 3 collaborative agents and 3 non-collab agents

Figure 16.: MAS+Simulated Annealing

Finally, thanks to collaboration between genetic algorithm agents, the convergence of the algorithm became noticeably faster as shown in Figure 17:



(a) Fitness evolution



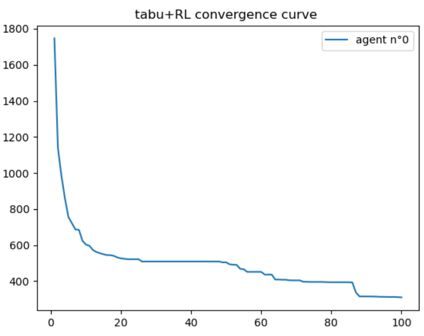
(b) Best route

Figure 17.: Genetic algorithm with MAS

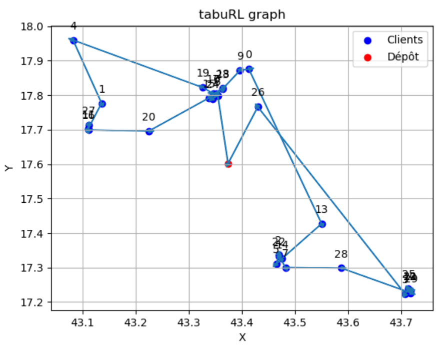
7.3. *Reinforcement learning*

7.3.1. *Collaborative Optimisation*

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 18:



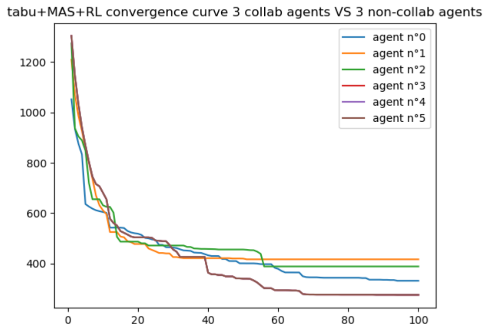
(a) Convergence curve



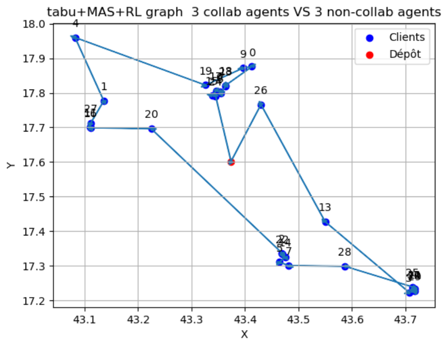
(b) Route graph

Figure 18.: Tabu + Reinforcement learning

The implementation of a Multi-Agent System (MAS) environment with six intelligent Tabu agents, including three collaborative agents, resulted in the convergence curve shown in Figure 19a. Initially, the non-collaborative agents performed on par with the collaborative ones. However, as the agents began to learn through Reinforcement Learning (RL), the collaborative agents accelerated their optimization, ultimately finding a significantly better final solution.



(a) Convergence curve



(b) Route graph

Figure 19.: Tabu + MAS + Reinforcement learning

When it came to Simulated Annealing (SA), tests of its performance using Reinforcement Learning (RL) to generate neighbors exhibited unpredictable behavior. While some agents were able to find feasible solutions with low cost, others converged prematurely and remained stuck at local optima for the majority of the iterations. This result is depicted in the convergence curve of SA with RL and six Multi-Agent System (MAS) agents. We also observed that after learning through 300 to 400 iterations, some SA agents showed a sudden improvement in their optimization of the feasible solution. A similar improvement could potentially have occurred for the two other SA agents that remained in a local optimum until the 500th iteration.

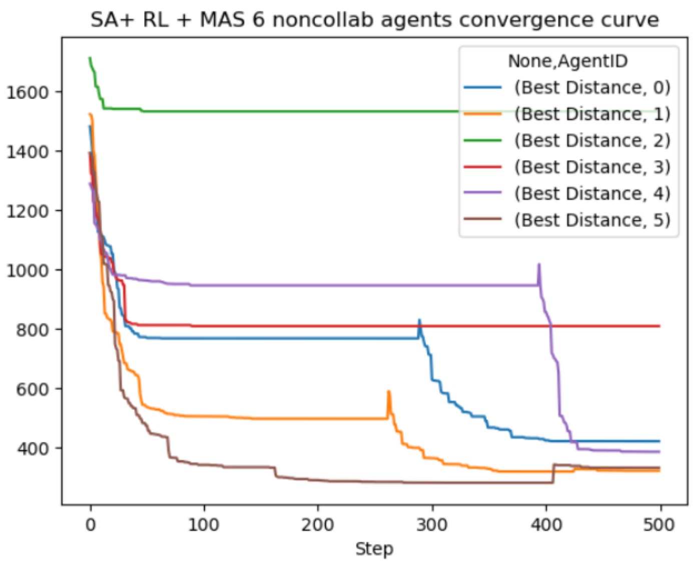
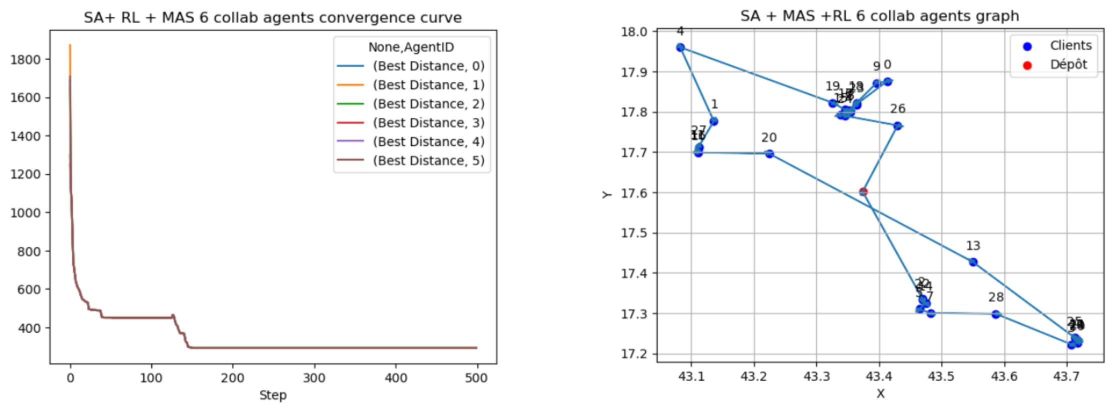


Figure 20.: Convergence curve of 6 non collaborative SA agents with RL

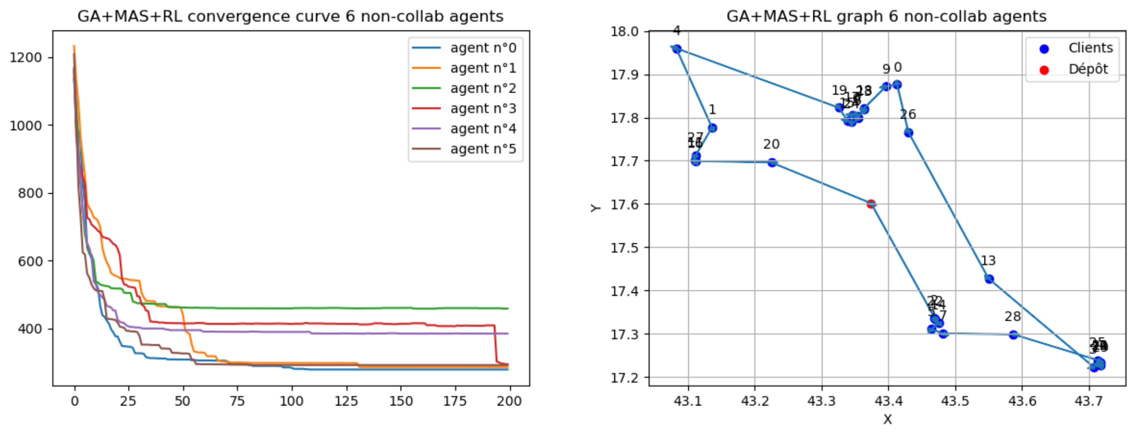
Collaborative mode proved invaluable in mitigating the unpredictability of Simulated Annealing (SA) agents, as shown in Figure 20. By sharing their feasible solutions, SA agents were able to reduce randomness and increase the speed of optimization.



(a) Convergence curve (b) Route graph

Figure 21.: SA + MAS + Reinforcement learning

The simulation results of Genetic Algorithm (GA) agents implemented with Reinforcement Learning (RL) revealed similar unpredictable behavior to that observed with Simulated Annealing (SA) agents using RL. However, unlike SA agents, the final feasible solutions found by GA agents did not exhibit objective function values of widely differing scales. This is illustrated in Figure 22.



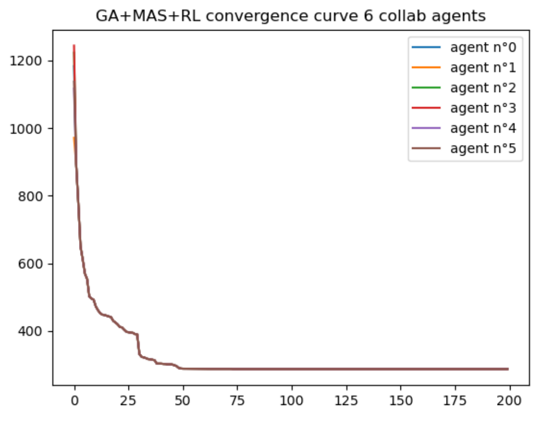
(a) Convergence curve (b) Route graph

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

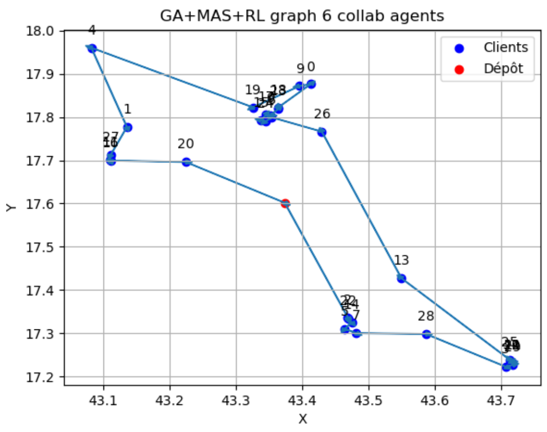
When we launched the algorithm with 6 collaborative agents, we observed a significant increase in the speed of problem optimization. Specifically, after only 50 time steps, the MAS with collaborative GA + RL agents identified a solution comparable to that found by 200 time steps of 6 non-collaborative GA + RL agents. This improvement is illustrated in Figure 22.

7.4. *Reinforcement Learning: Alternating between MAS algorithms*

We implemented a Reinforcement Learning (RL) environment where an agent iteratively selects between executing one of three MAS algorithms: MAS-RLT, MAS-RLGA, or MAS-RLSA. This approach is referred to as AMAS (Alternating between



(a) Convergence curve

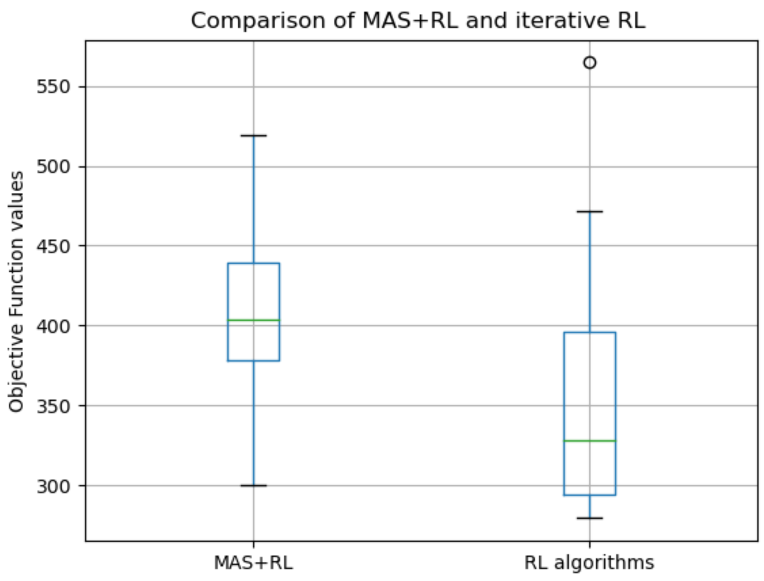


(b) Route graph

Figure 23.: GA + MAS + Reinforcement learning: 6 collab agents

MAS algorithms). We compared AMAS to a Global MAS (GMAS) environment, which simultaneously employs all three algorithms. For the comparison, both AMAS and GMAS were executed 10 times each. GMAS was parameterized to perform 150 steps per iteration, while AMAS was parameterized to perform 10 steps per iteration. The total optimization time for 10 runs of AMAS was 4 minutes, compared to 6 minutes for GMAS.

The boxplot below illustrates the comparative results of these tests. Overall, AMAS demonstrated superior performance relative to GMAS. However, it is important to note that AMAS occasionally produced less satisfactory feasible solutions, as evidenced by the outlier in the boxplot.



7.5. *Comparative and global analyses of collaborative optimisation performance*

Having developed and implemented three distinct optimization techniques for each metaheuristic algorithm, we conducted a comprehensive comparative analysis of their performance. The table below summarizes the key results and metrics obtained from this analysis. The results shown consist of the values of the objective function of the optimized feasible solution in each case. The unit of these values is kilometer, but it doesn't correspond to the exact distance to travel since there re two other components of the objective function(time window and cost per vehicle).

Number	Metaheuristic			Metaheuristic + MAS			MAS-RL		
of clients	GA	SA	Tabu	GA	SA	Tabu	GA	SA	Tabu
20	348.8	295.3	417.1	314.8	284.5	268.7	269.3	293.1	284.5
30	576.7	297.6	493.7	452.0	323.2	300.4	291.8	275.8	298.4
40	566.3	506.0	544.2	585.8	496.2	547.8	398.1	491.6	476.8
50	1112.9	792.8	776.6	1003.8	786.2	602.4	626.6	733.2	488.7
60	984.1	1125.8	1030.1	825.9	742.7	537.2	615.7	770.4	677.1
70	1359.4	1175.6	1328.9	1191.6	1021.4	711.0	840.7	680.1	668.0
80	1866.9	1347.7	1107.6	1589.8	1384.9	658.4	793.7	1226.9	768.4
90	2367.2	1340.5	1072.7	1490.0	1308.9	786.6	1032.6	1196.8	907.4
100	2576.7	1932.7	1703.7	2011.5	1843.5	752.9	1168.1	1083.9	1173.3
110	2916.8	1898.8	1425.2	2385.9	1652.3	1034.3	1714.9	1557.3	1568.8
120	3455.8	2738.8	1665.2	2895.2	2072.4	1398.1	1950.5	1748.8	1728.0
130	4352.5	2689.2	2112.7	3390.9	2497.4	1451.0	2014.1	2011.8	1971.5

We plotted the evolution of the objective function as a function of the number of clients for each metaheuristic algorithm under three scenarios: without MAS, with MAS, and with MAS+RL. The resulting curves are shown in Figure 23.

The key observations from these curves are as follows:

(1) **GA and SA Agents**:
Implementing MAS and adding intelligence to agents significantly improves the optimization of the problem. Both GA and SA agents demonstrate enhanced performance when MAS and RL are integrated, leading to better convergence and solution quality.

(2) **Tabu Agents**:
The benefits of MAS and RL are apparent for tabu agents only when dealing with a number of clients fewer than 80. For larger client numbers, traditional MAS tabu agents outperform MAS-RLT agents, indicating that the added complexity of reinforcement learning does not always translate to better results in highly complex scenarios.

(3) **Overall Performance**:
Globally, tabu-based methods consistently show better results than GA and SA-based methods, especially in handling larger problem instances. This suggests that tabu search's inherent strengths in exploring solution spaces make it more robust in certain contexts. These findings highlight the importance of selecting the appropriate optimization strategy based on the specific characteristics of the problem, such as the number of clients involved.

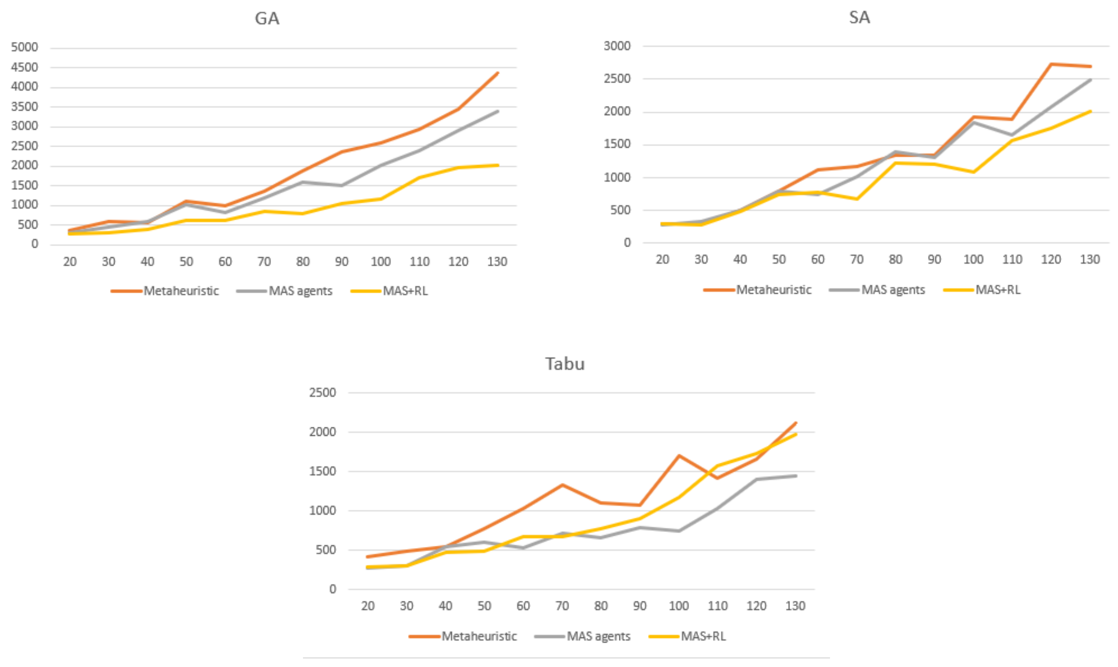


Figure 24.: Comparison of different optimization methods

8. Conclusion and future work

In this study, we explored the application of three metaheuristic algorithms—Tabu Search, Genetic Algorithm (GA), and Simulated Annealing (SA)—to tackle the Vehicle Routing Problem (VRP). By meticulously tuning the parameters of each algorithm to align with the problem's specific requirements, such as the number of vehicles and clients, we aimed to achieve optimal solutions. Furthermore, we enhanced the performance of these algorithms by developing a Multi-Agent System, where each agent represented a metaheuristic algorithm capable of collaboration. This collaborative approach significantly improved the quality of the solutions, producing more efficient route graphs.

To further refine our approach, we incorporated Reinforcement Learning, specifically Q-learning, into the neighborhood generation and mutation processes. By defining states as various techniques of neighbor generation and actions as transitions between these techniques, we successfully adapted the Q-learning algorithm to this problem. This adaptation had a substantial impact on enhancing the performance of the metaheuristic algorithms, yielding more optimal solutions.

Additionally, we implemented a reinforcement learning environment that enabled the iterative selection of algorithms among MAS with RL and Tabu, MAS with RL and GA, and MAS with RL and SA. By adapting the Q-learning algorithm, the system learned to dynamically choose the most effective algorithmic approach based on the specific problem instance. This adaptability further improved solution quality and efficiency, demonstrating the powerful synergy between reinforcement learning and metaheuristic optimization.

The simulation results showed that implementing MAS environments with collaboration between agents noticeably improved the optimization process. Adding intelligence to the neighbor generation and mutated children generation was a step further towards efficiently optimizing the VRPTW problem. Even though the tabu algorithm showed the best results in every optimization approach(metaheuristic, MAS, MAS + RL), its performance with RL regressed for numbers of clients greater than 80 while GA and SA showed constant improvement of their optimization throughout the different optimization approaches tackled in this study.

While our model accounted for several constraints, including distance optimization, vehicle count, and client time windows, there remains room for further improvement. Future work could incorporate additional constraints, such as maximum vehicle load capacities and precise client-to-client routes, to enhance the model's applicability and accuracy in solving the Vehicle Routing Problem with Time Windows (VRPTW). By addressing these aspects, we can create even more robust solutions that better meet the complex demands of modern logistics and delivery systems.

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*
[30] Allan Larsen *The Dynamic Vehicle Routing Problem*