ICO - Intelligence collaborative
Vehicle routing problem

2024

Report

ILYASSE CHAOUKI

ANTOINE DESCHAMPS

VIRGILE DEVILLERS

ANTOINE GREAUME

MATTHIEU RICHARD

Table of contents:

I.1. Description **4**
I.2. Complexity **5**
I.3. State-of-the-art **6**
II.1. Tabu **7**
II.1.1. Tabu search algorithm for PTVFT 7
II.1.2. Small database 10
II.1.2.1. Chart 10
II.1.2.2. Results analysis 10
II.1.2.3. Study 1 conclusion 11
II.1.3. Full database 11
II.1.3.1. Charts 11
II.1.3.2. Results analysis 12
II.1.3.3. Study 2 conclusion 12
II.2. Simulated annealing **12**
II.2.1. Specific VRP algorithm 13
II.2.2. Small database 17
II.2.2.1. Results 17
II.2.2.2. Result analysis 21
II.2.2.3. Relevance of the study 21
II.2.3. Large Database 21
II.2.3.1. Results 21
II.2.3.2. Results analysis 25
II.2.3.3. Interest of the study 25
II.3. Genetic algorithm **25**
II.3.1. Specific VRP algorithm 25
II.3.2. Small Database 27
II.3.2.1. Results 28
II.3.2.2. Results analysis 35
II.3.2.3. Relevance of the study 35
II.3.3. Large Database 36
II.3.3.1. Results 36
II.3.3.2. Analysis of the results 43
II.3.3.3. Relevance of the study 43
II.4. Comparison of meta-heuristics **44**
III.1. SMA Ami Protocol: **45**
II.2.1. Specific SMA Ami algorithm 45
II.2.2.1. Results 47
II.3.3.2. Analysis of the results and interest of the study 48
III.2. SMA Enemy protocol **48**
III.2.1 Enemy protocol 48

III.2.1 Analysis of the results 50
III.3. Comparison of the two protocols **50**
I Principles **51**
I-Reinforcement method with Q-learning 51
I.2 - SMA and reinforcement 52
I.3 - creating neighbors 53
I.4 - Q-learning choice 53
I.5 - Choice of the next current state 54
II Results for Tabou and RS **55**
III Results for SMA and reinforcement **55**

I. Vehicles routing problem (VRP)

I.1. Description

Une tournée de véhicules consiste en un ensemble d'itinéraires optimisés pour les véhicules d'une entreprise, visant à minimiser les coûts et à améliorer l'efficacité des déplacements, que ce soit pour la livraison, la logistique ou d'autres activités similaires.
L'optimisation des tournées de véhicules constitue une préoccupation majeure pour de nombreuses entreprises, qu'elles opèrent dans le domaine de la logistique, du transport ou des services de livraison.

Le défi principal réside dans la gestion complexe et dynamique des itinéraires empruntés par les véhicules au sein d'une flotte. Les entreprises sont confrontées à une multiplicité de paramètres à prendre en compte, tels que la distance à parcourir, le temps de trajet, les contraintes horaires, les capacités des véhicules, les exigences spécifiques des clients, et les fluctuations du trafic routier.
La variabilité constante de ces facteurs rend la planification des tournées manuelle souvent inefficace et sujette à des erreurs. L'optimisation des tournées de véhicules vise donc à automatiser ce processus en utilisant des algorithmes avancés. Les objectifs considérés sont nombreux : obtenir des itinéraires plus courts, des temps d'attente réduits, une utilisation optimale des ressources et une diminution des émissions de gaz à effet de serre associées aux déplacements inutiles.

Cette problématique revêt une importance stratégique, car une planification efficace des tournées peut non seulement améliorer la rentabilité de l'entreprise en réduisant les coûts liés au carburant, à la maintenance et au temps de travail, mais également renforcer la satisfaction client en garantissant des délais de livraison plus courts et plus fiables.
Cependant, l'optimisation des tournées de véhicules n'est pas dépourvue de défis, notamment en raison de la complexité des contraintes à considérer et de la nécessité de prendre en compte les imprévus. Les avancées technologiques, telles que l'intelligence artificielle et l'apprentissage automatique, sont de plus en plus exploitées pour relever ces défis et offrir des solutions de planification de tournées plus agiles et performantes. Ainsi, cette problématique représente un enjeu crucial dans la quête constante d'efficacité opérationnelle et de compétitivité pour les entreprises dépendantes de la gestion de flottes de véhicules.

Pour introduire le problème, nous avons visualiser la position des clients à livrer.

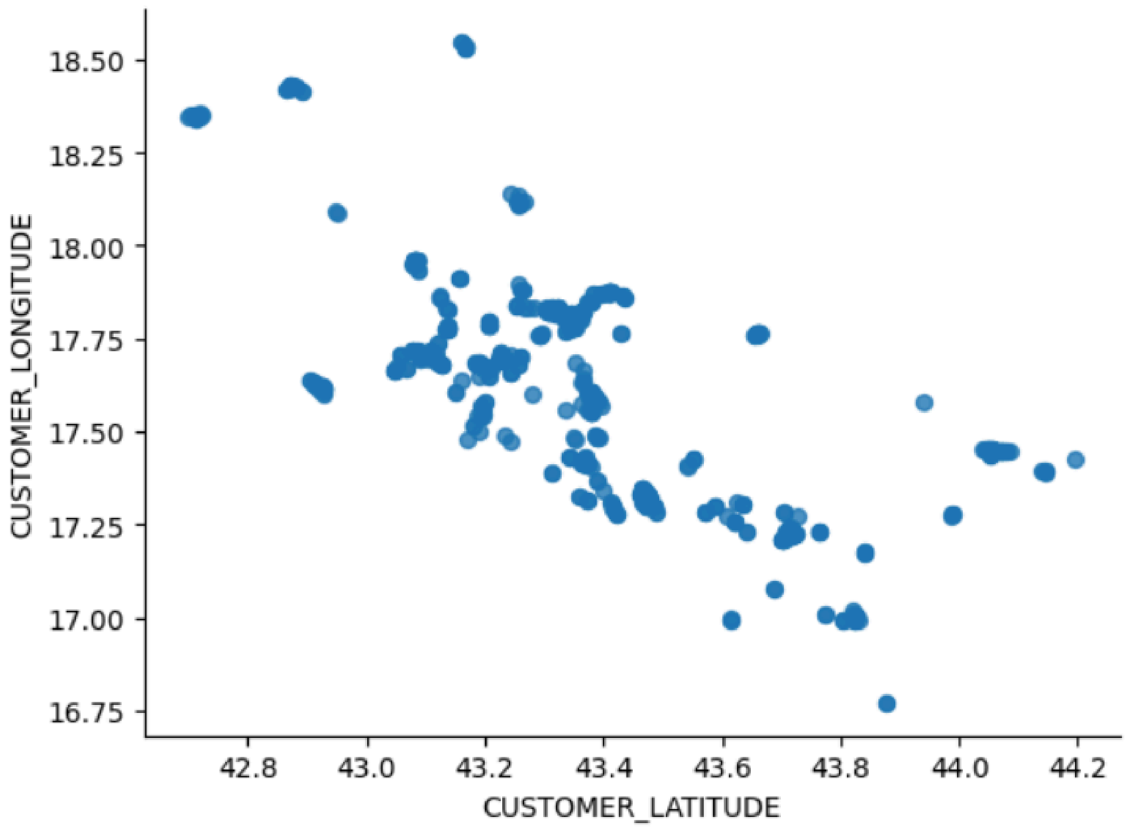


figure 1 : map of the clients to visit

Pour résoudre ce problème, nous allons commencer par créer 3 algorithmes méta-heuristiques classiques pour l'ordonnancement de tâches : l'algorithme tabou, l'algorithme du recuit simulé et l'algorithme génétique. La fonction de coût associée à ces algorithmes doit minimiser les distances parcourues par les camions.

I.2. Complexity

The Vehicle Routing Problem (VRP) falls into the category of NP-hard problems, indicating its inherent complexity and difficulty in solving. As the problem size increases, the computational effort required to solve the VRP grows exponentially. Specifically, the complexity of the VRP matches or exceeds that of the Traveling Salesman Problem (TSP) with time windows, a well-known NP-complete problem. This comparison highlights that the VRP, like the TSP with time windows, lacks any known polynomial-time algorithms capable of consistently solving it efficiently across various cases.

I.3. State-of-the-art

Currently, the most advanced approaches to the Vehicle Routing Problem (VRP) encompass both precise and heuristic solution strategies, along with sophisticated modeling and optimization techniques. Exact solutions utilize mathematical programming techniques, such as integer linear programming (ILP) and constraint programming (CP), to determine the optimal solution for the VRP.

In contrast, heuristic and metaheuristic approaches, including methods like tabu search, genetic algorithms, ant colony optimization, simulated annealing, and gradient descent, are employed to find good-quality solutions within a practical timeframe.

Furthermore, the cutting-edge in VRP also involves advanced multi-objective modeling and problem-solving techniques. These methods concurrently consider multiple criteria, such as the total distance traveled, the number of vehicles deployed, and the overall travel time.

II. Optimization using Metaheuristics

Before coding optimization algorithms, it is necessary to determine the criteria that we are trying to minimize. As part of this study, we decided to minimize the total distance traveled as well as the number of lorries used. In order to take these 2 elements into account, we have chosen to model them by the following cost function:

$$Cost(scheduling) = (\sum_{(i,j)\in scheduling} d_{i,j}) * (1 + \frac{vehicles}{20})$$

where:

$d_{i,j}$ is the distance between the clients i and j

$vehicles$ is the number of vehicles used in the scheduling

Thus, the cost function can be implemented as below :

```
f_cost(scheduling)
    cost ← 0
    vehicles ← 0
    for route in scheduling do
        distance ← 0
        last_client ← 0
        for client in route do
            distance ← distance + calcul_distance(last_client, client)
            last_client ← client
        end for
        if distance > 0 do
            vehicles ← vehicles + 1
        end if
        cost ← cost + distance
    end for
    return cost * (1 + vehicles/20)
end f_cost
```

II.1. Tabu

II.1.1. Tabu search algorithm for PTVFT

The taboo algorithm starts with an acceptable solution, and from this generates a set of neighboring solutions. The quality of these solutions is assessed using the score function, which evaluates the total distance covered. This operation is repeated to explore part of the set of possibilities, keeping a list of solutions already explored to avoid getting stuck on a local minimum.

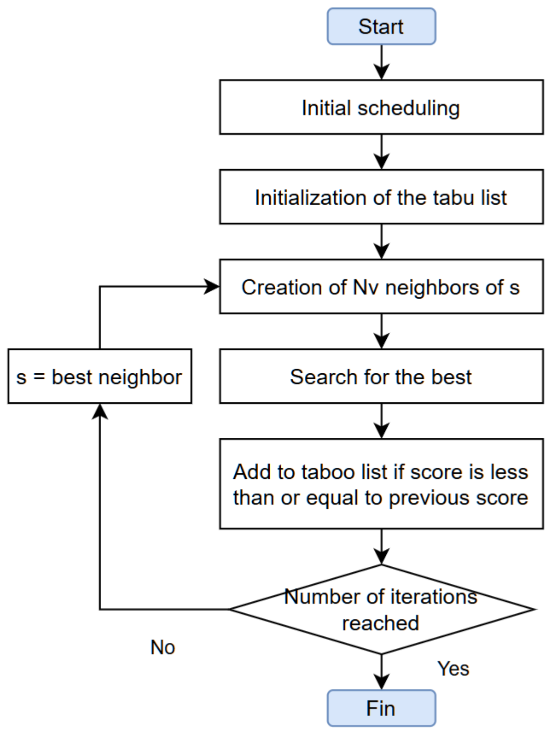

Start
Initial scheduling
Initialization of the tabu list
Creation of Nv neighbors of s
s = best neighbor
Search for the best
Add to taboo list if score is less than or equal to previous score
Number of iterations reached
No
Yes
Fin

Figure 2 : Principle of the tabu algorithm

To create the neighbors, we transform the initial list, which is a list containing n different lists, n being the number of cars, within which we have the customers in delivery order. To preserve the list transformation steps and thus create the taboo list more efficiently, we transform this list of lists into a single list, where car changes are symbolized by -1s. In this way, no information is lost, and we can perform elementary operations to create neighbors more efficiently. We therefore use these three methods, which achieve all possible theoretical solutions: exchanging two random elements, inverting a random-sized part of the list and moving a random-sized part to another location in the list.
The study of the efficiency of neighbor selection is covered in the large-scale BDD section.

ordo_to_tab_list(ordo):

[[1;2;3;...][10;11;...]...] $\longrightarrow$ [1;2;3;...;-1;10;11;...;-1,...]

List of lists *Unique list*

Figure 2 : Conversion from initial scheduling to tabu scheduling

---

Tabu algorithm :

```
algoTabou(s_depart, NbIterMax, Nbvoisins, Historique_scores):
  s ← ordo_to_tab_list(s_depart) ▹ Transform initial solution into tabu format
  s_star ← s
  nb_iter ← 0
  T ← [ ]
  meil_score = calcul_cout(tab_list_to_ordo(s_star))
  while (nb_iter < NbIterMax):
    nb_iter ← nb_iter + 1
    V ← voisins(s, Nbvoisins)
    v,M ← best neighbour out of V, and its score
    if(M<meil_score): ▹ Keep the solution as the best
      s_star, meil_score  ← v, M
      s  ← v
    else:  ▹ Add the solution to the tabu list
      T.append(v)
      s=v
  return tab_list_to_ordo(s_star) ▹ Return the best solution of all the iterations
```

Neighbors creation :

Method 1 : Swapping randomly 2 clients

```
for i in range(NbVoisins):
  a,b  ← random index of sol
  v  ←  copy of sol
  v[a],v[b]  ← v[b],v[a]
  # checks the criterias of volume and weight
  if v checks the criterias of volume and weight:
    V.append(v)
```

Method 2 : inverting a part of the list

```
for i in range(NbVoisins):
  a,b  ← random index of sol
  v  ←  copy of sol
  v[a:b] ← reverse of v[a:b]
  if v checks the criterias of volume and weight:
    V.append(v)
```

Method 3 : Moving a part of the list

```
for i in range(NbVoisins):
  a,b  ← random index of sol
  v=copy.deepcopy(sol)
  if b<a:
    a,b ← b,a
  sequence ← v[a:b]
  delete sequence from v
  c ←  random index of v
```

```
v ← v[0:c] + sequence + v[c:len(v)]
if v checks the criterias of volume and weight:
  V.append(v)
```

II.1.2. Small database

II.1.2.1. Chart

For the small database, we use only the first 20 customers of route 1. For the simulations, we set an initial schedule and then varied the parameters. The result for 100 iterations is 200 neighbors.

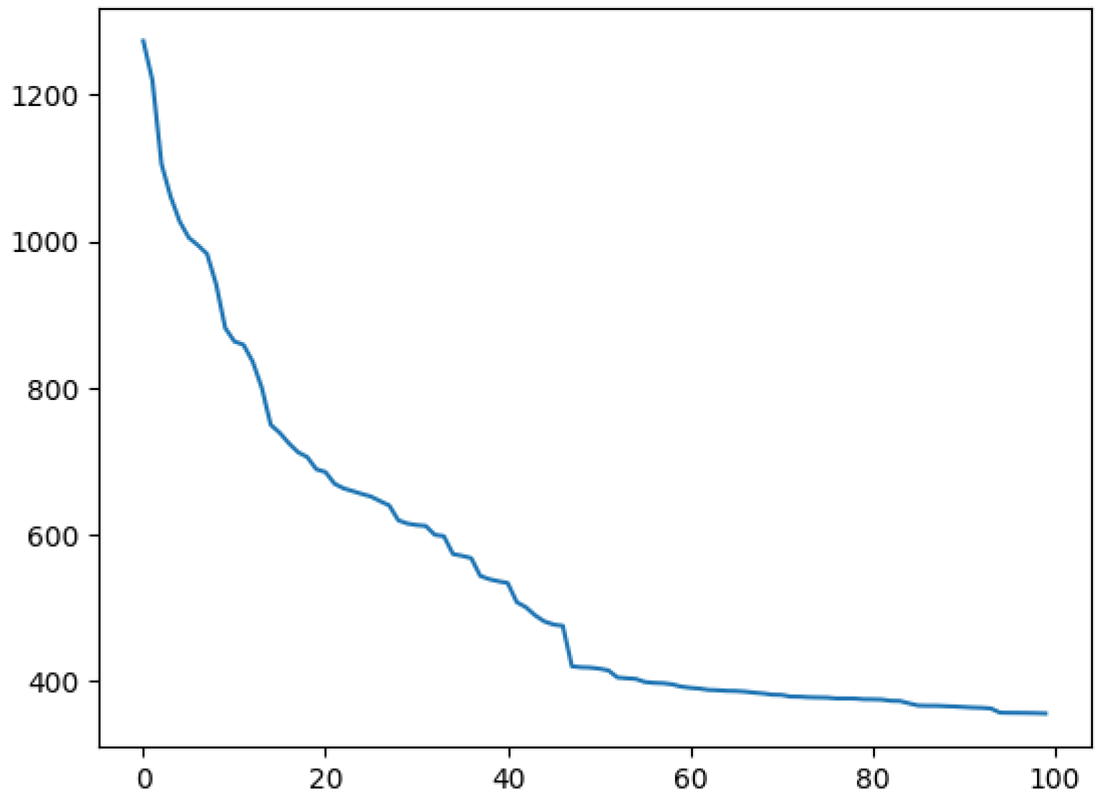


Figure 3 : Evolution of the solution cost for each iteration, in the case of 20 clients (parameters 100 iterations, 200 neighbors)

II.1.2.2. Results analysis

The shape of the curve clearly shows an improvement in the cost of solutions with each iteration.

II.1.2.3. Study 1 conclusion

This study validates the algorithm's operation.

II.1.3. Full database

II.1.3.1. Charts

First, we compare neighborhood methods for the same starting schedule. We have chosen to study the cost as a function of algorithm execution time, since a neighborhood method may be more efficient, but may take longer to execute. The average of 10 experiments is shown in the graph below.

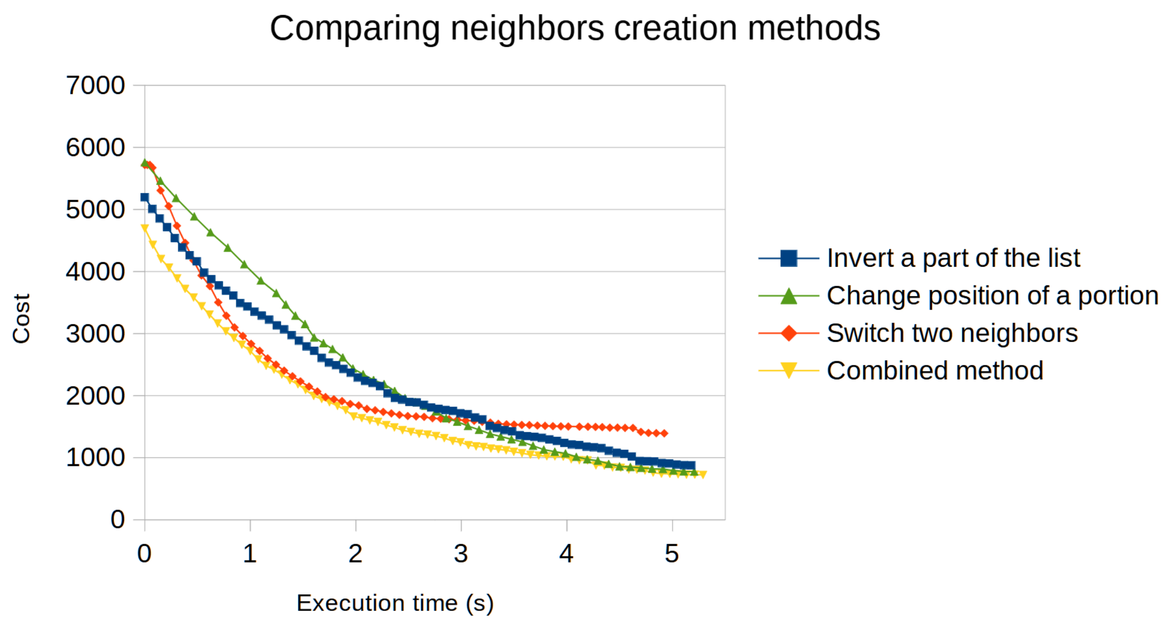


Figure 4 : Comparing neighbors creation methods

Next, we apply the algorithm with the combined neighborhoods method, with the following parameters: 100 iterations, 140 neighbors

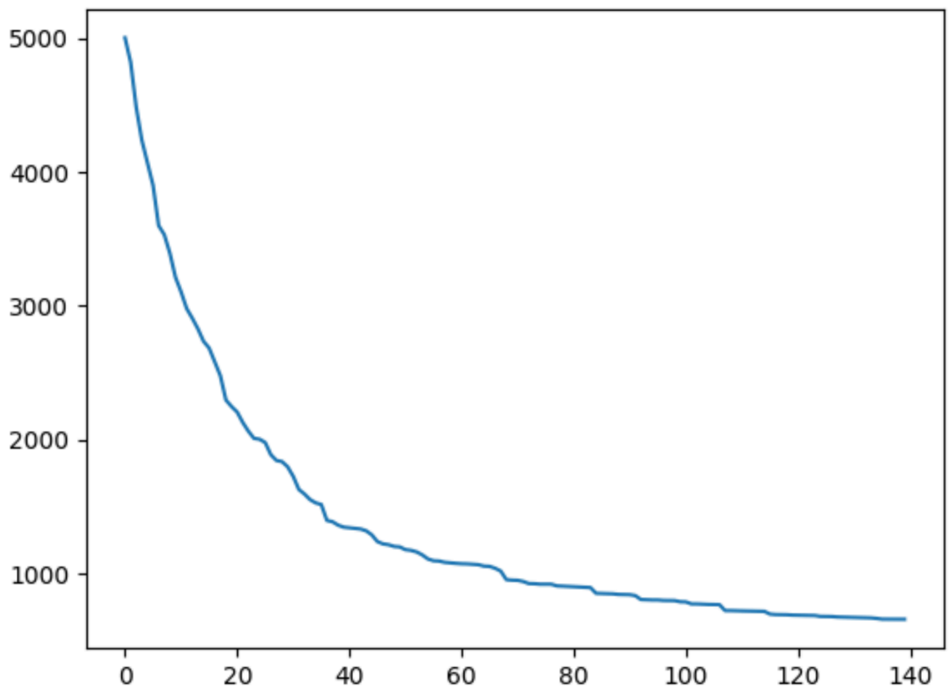


Figure 4 : Evolution of the solution cost for each iteration, in the case of 107 clients (parameters 100 iterations, 140 neighbors)

II.1.3.2. Results analysis

For the first graph, we can see that for each neighbor used, the shape of the curve is different. For example, the two-neighbor exchange method is faster at the beginning of the algorithm, but seems less efficient at the end. We note that the combined method is below the other curves at all times, showing that it integrates the advantages of each neighborhood method.
For the second graph, we tested several parameters for iterations and number of neighbors created, and the best results were given with the following parameters: 100 iterations for 140 neighbors.

II.1.3.3. Study 2 conclusion

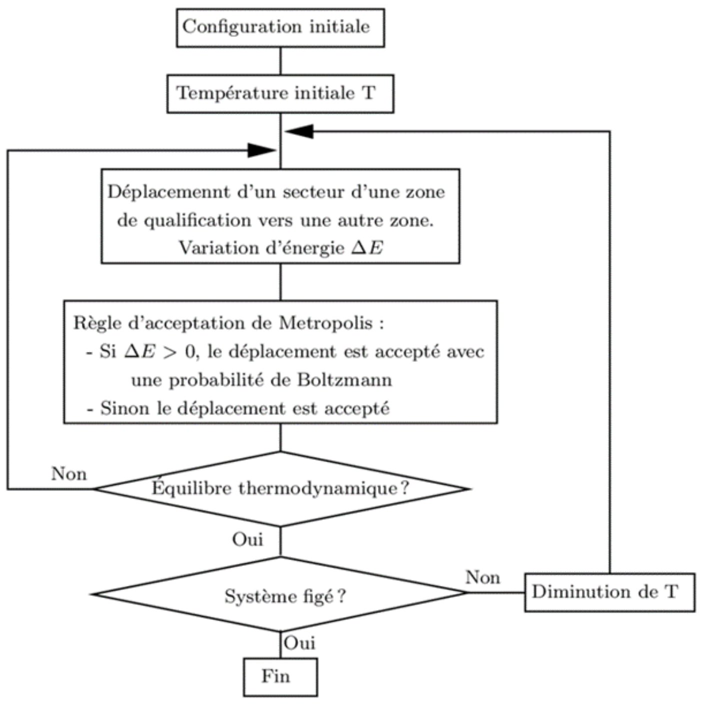
Using all three types of neighborhoods improves the results of the taboo algorithm.
Applying this combined method results in a cost of around 600.
The ratio rule obtained NbIteration = 1.4*NbNeighbors is retained for the rest of the study in the case of this algorithm.

II.2. Simulated annealing

II.2.1. Specific VRP algorithm

The simulated annealing algorithm is an optimization method that mimics the slow cooling process of a molten material. The algorithm seeks to explore the solution space in such a way as to avoid missing the global optimum by falling into a local valley, using temperature to control the balance between exploration and exploitation. In some cases, it is possible to move up the valley. It works as follows:

- **Initialization:** The algorithm starts with a random initial solution and a high initial temperature. This temperature represents a tolerance threshold for accepting lower-quality solutions in order to explore the search space.
- **Generation of Neighbors:** At each iteration, the algorithm generates a solution close to the current one. This generation is based on small, random modifications to the current solution.
- **Evaluation and Acceptance:** The new solution generated is evaluated. If it is better than the current solution, it is accepted as the new current solution. If it is worse, it can still be accepted with a certain probability, which decreases with temperature and the difference between the qualities of the two solutions. The formula for this probability is : exp(- dE /kB*T)
- **Cooling:** The temperature is gradually lowered. This process initially allows a wide exploration of the search space (including the acceptance of lower-quality solutions) and gradually favors the exploitation of promising regions, reducing the probability of accepting lower-quality solutions.
- **Stop criterion**: The algorithm terminates when the temperature reaches a low final value (close to zero) or after a predefined number of iterations without improvement, indicating that a sufficiently optimal solution has been found or that the algorithm is unable to find better solutions.



Configuration initiale
Température initiale T
Déplacemennt d'un secteur d'une zone de qualification vers une autre zone. Variation d'énergie ΔE
Règle d'acceptation de Metropolis :
- Si $\Delta E > 0$, le déplacement est accepté avec une probabilité de Boltzmann
- Sinon le déplacement est accepté
Non
Équilibre thermodynamique ?
Oui
Système figé ?
Non
Diminution de T
Oui
Fin

figure 5 : Explanatory diagram of the simulated circuit

Initialization :

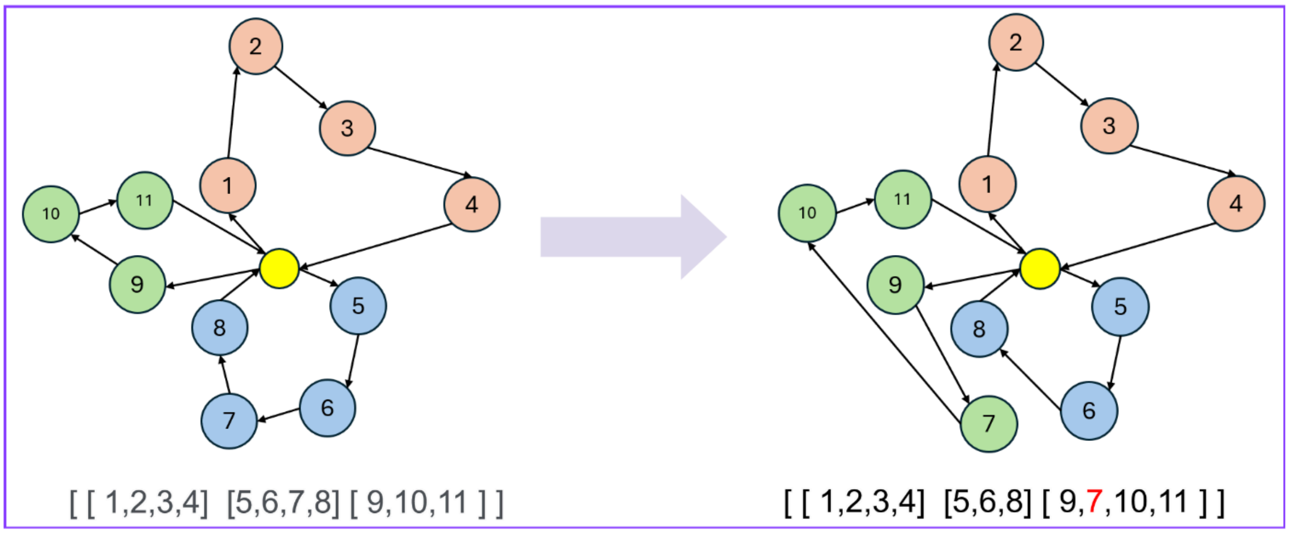
For initialization, we implemented a function that generates a **random schedule** with a given number of trucks and a given number of customers to be delivered.

Generation of Neighbors :

We have implemented **four strategies** for finding **neighboring schedules**:

1) Random Customer Move

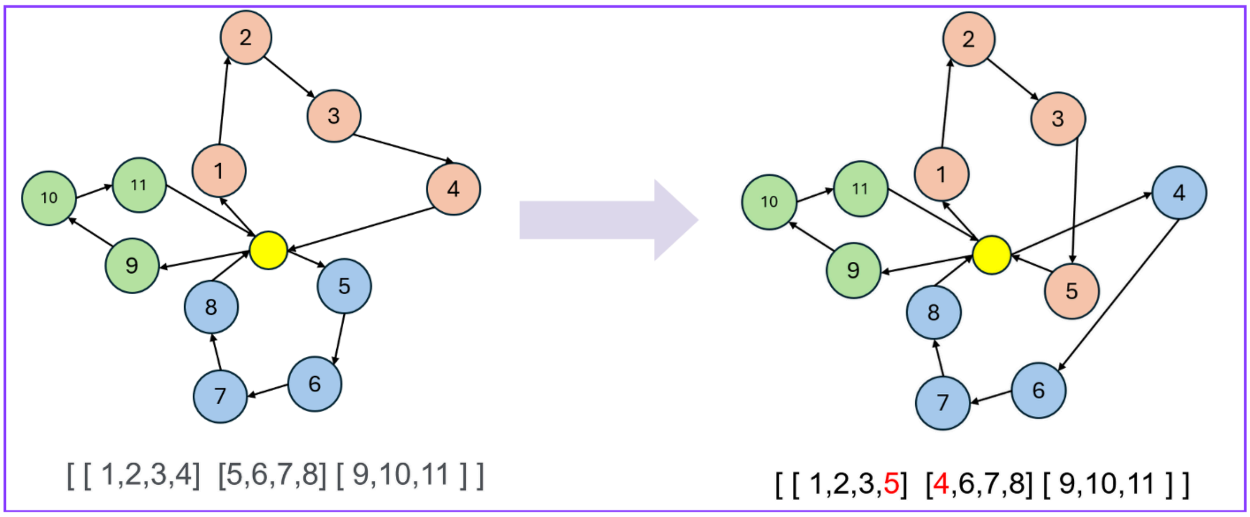
This strategy consists of randomly selecting a customer from a circuit (a circuit is the route taken by a truck) and placing it at a random position in another circuit or the same circuit. The truck can make several loops around the depot during its tour, and is then moved to another circuit (another truck).



Here, customer 7 has been randomly selected. It is moved from the blue circuit to a random position in the green circuit.

2) Intercircuit Customer Exchange

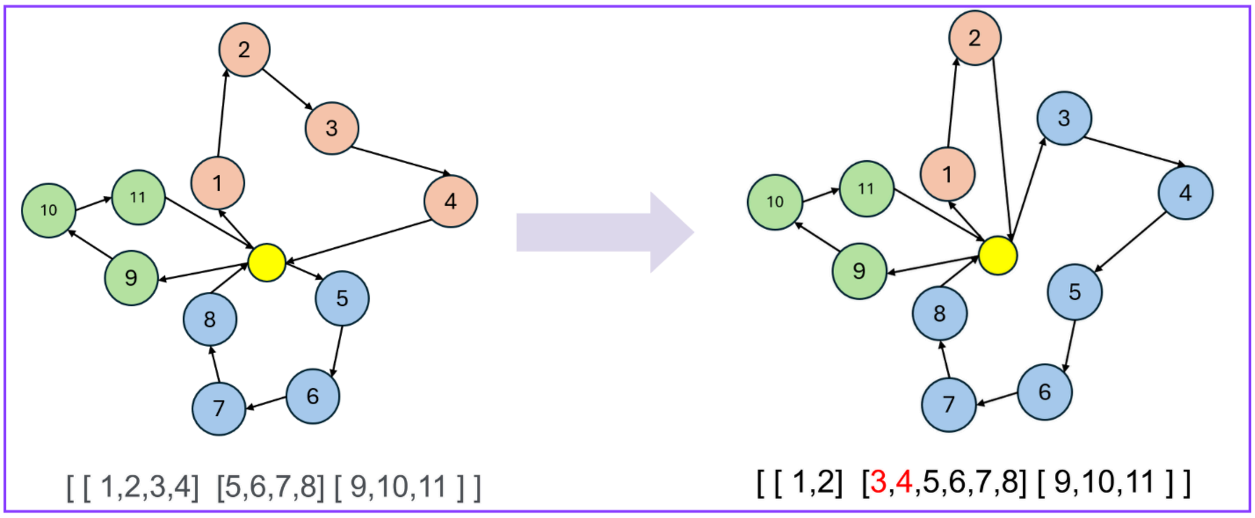
This strategy involves selecting two different routes and exchanging one customer from each route. This method is similar to the previous one, but places greater constraints on scheduling. Each truck retains the same number of customers to deliver as in the initial state.



Here, customers 4 and 5 have swapped places in the red and blue circuits.

3) Merging and splitting circuits

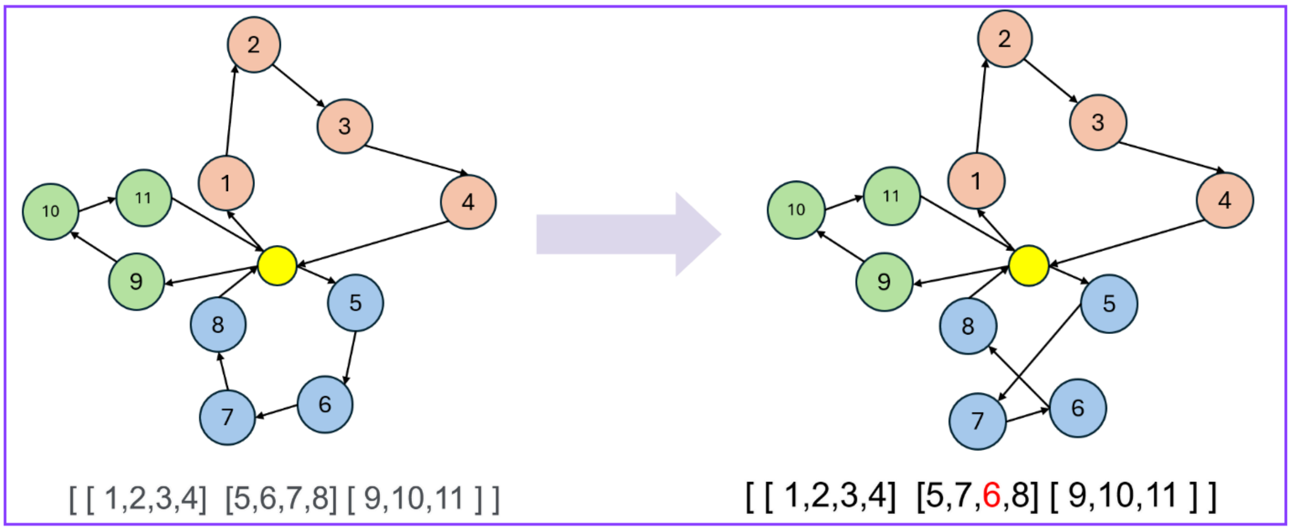
This strategy selects two random circuits, merges them into one large circuit, then splits this combined circuit into two new circuits at a random split point. In this way, it does not vary the number of circuits. This strategy significantly modifies the scheduling at each iteration, allowing the number of customers assigned to each truck to evolve.



Here, the red and blue routes were merged and then the vision took place at customer 3.

4) Moving a customer within the same route

Instead of moving a customer from one route to another, this strategy moves a customer to another position within the same route. This makes it possible to explore potential scheduling improvements without altering the distribution of customers between routes.



Here, customer 6 is randomly moved around the circuit.

<u>Evaluation and Acceptance:</u>

To assess the quality of a method, we have implemented a cost function that calculates **the sum of the distances covered by the trucks, weighted according to the number of trucks used**. The best schedule is then the one that minimizes this function.

All schedules generated using neighborhood methods must meet several criteria. A function guarantees the validity of the scheduling by ensuring that the constraints of the problem, such as **vehicle capacity in terms of weight and volume**, are respected after the modification.

Cooling and Stop criterion:

The implemented annealing algorithm takes several parameters into account, including the neighborhood strategy to be applied, the initial scheduling, the cooling factor and the maximum number of iterations to be performed. We'll look at their influence in the results analysis section.

II.2.2. Small database

II.2.2.1. Results

For the small database, we use only the first 20 customers of route 1. For the simulations, we set an initial schedule and then vary the parameters. Each time, we compare the four neighbor generation strategies.

Comparison of neighborhood strategies :

For the following parameters, we obtain the graph below: temperature_initiale=100,iterations_par_cycle=1000,facteur_refroidissement=0.95

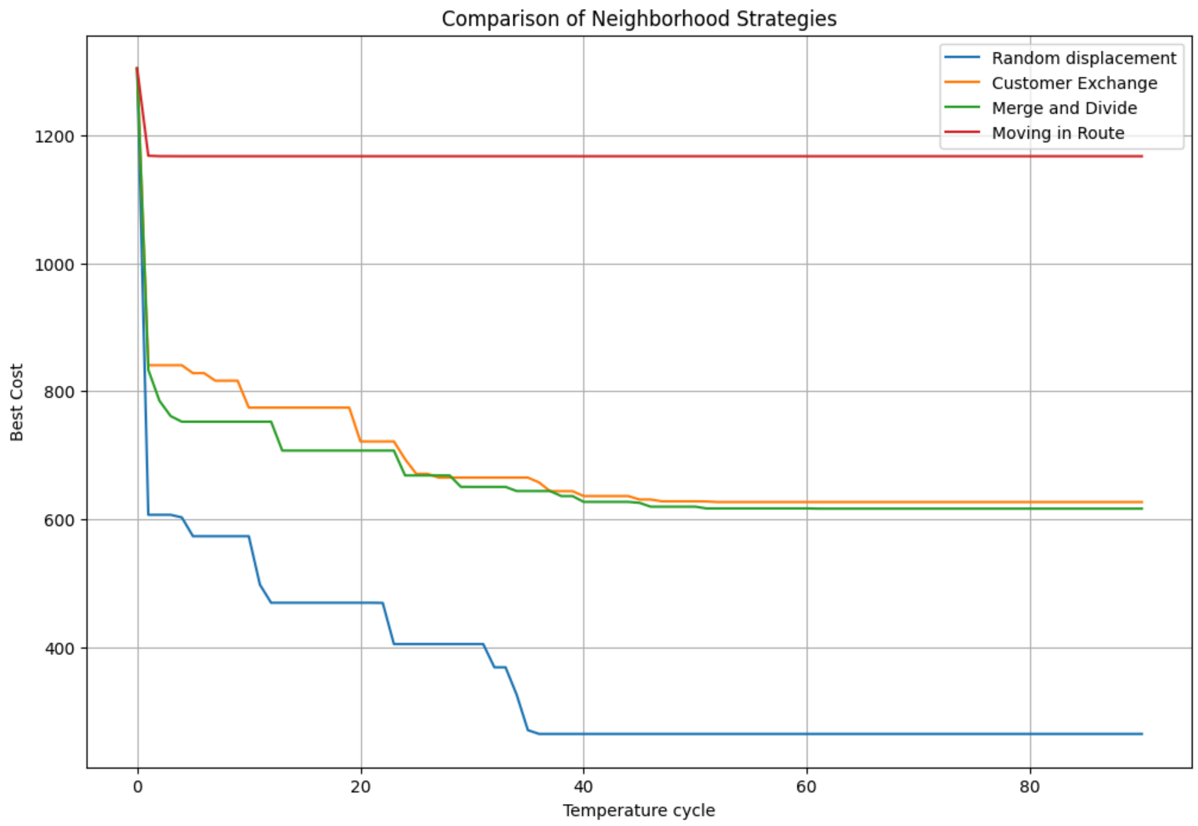


Figure 6 : Comparison of four neighborhood methods for simulated annealing on a small database

Result modification cooling factors :

For the following parameters, we obtain the graph below: temperature_initiale=100,iterations_par_cycle=1500,facteur_refroidissement=0.8

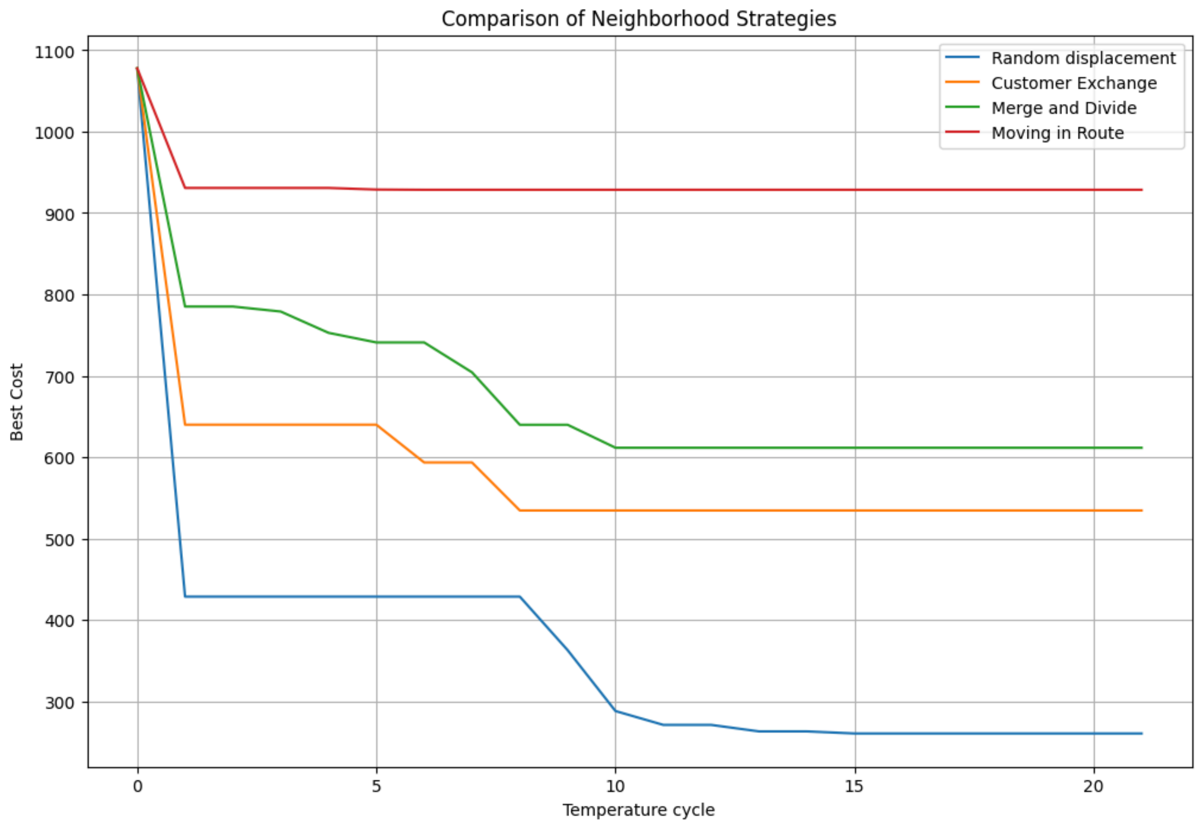


Figure 7 : Evolution of the cost of the best solution for the four neighborhood methods with a cooling factor of 0.8 on a small database

For the following parameters, we obtain the graph below: temperature_initiale=100,iterations_par_cycle=1500,facteur_refroidissement=0.8

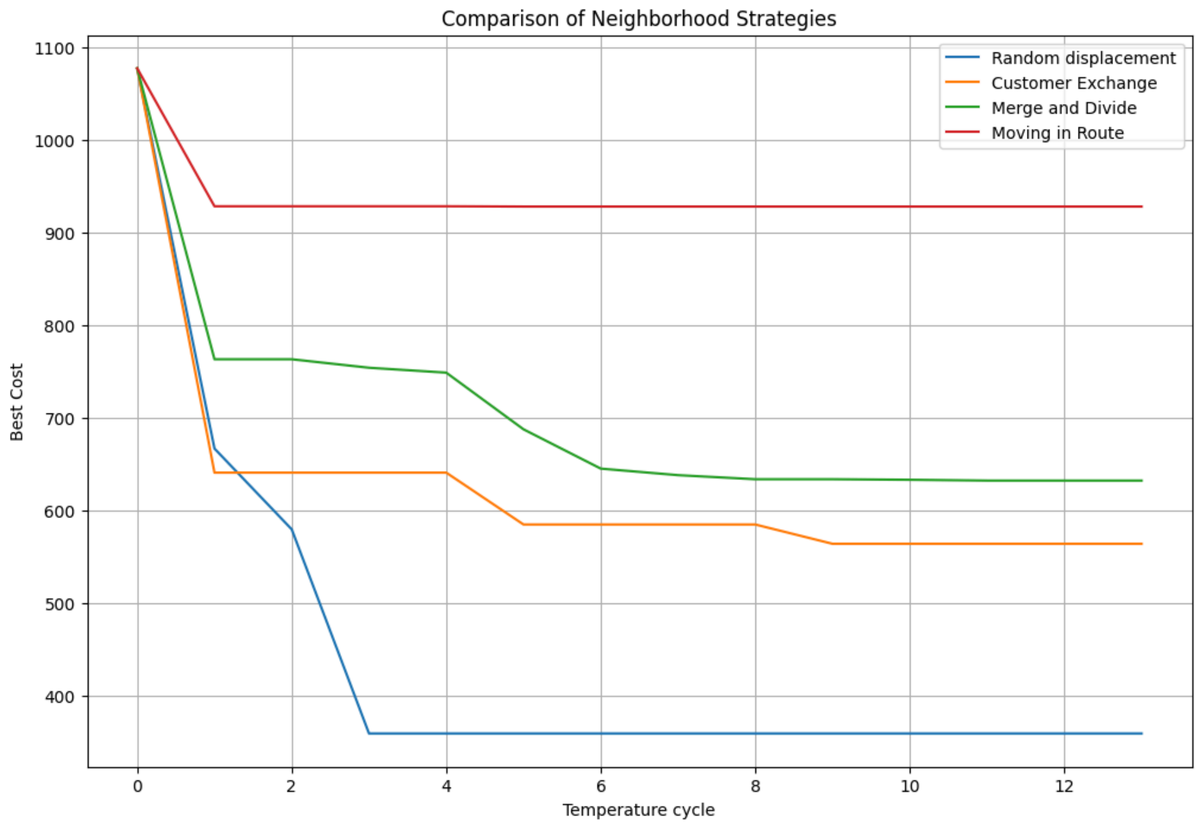


Figure 8 : Evolution of the cost of the best solution for the four neighborhood methods with a cooling factor of 0.7 on a small database

The results of the influence of the cooling factor are summarized in a graph :

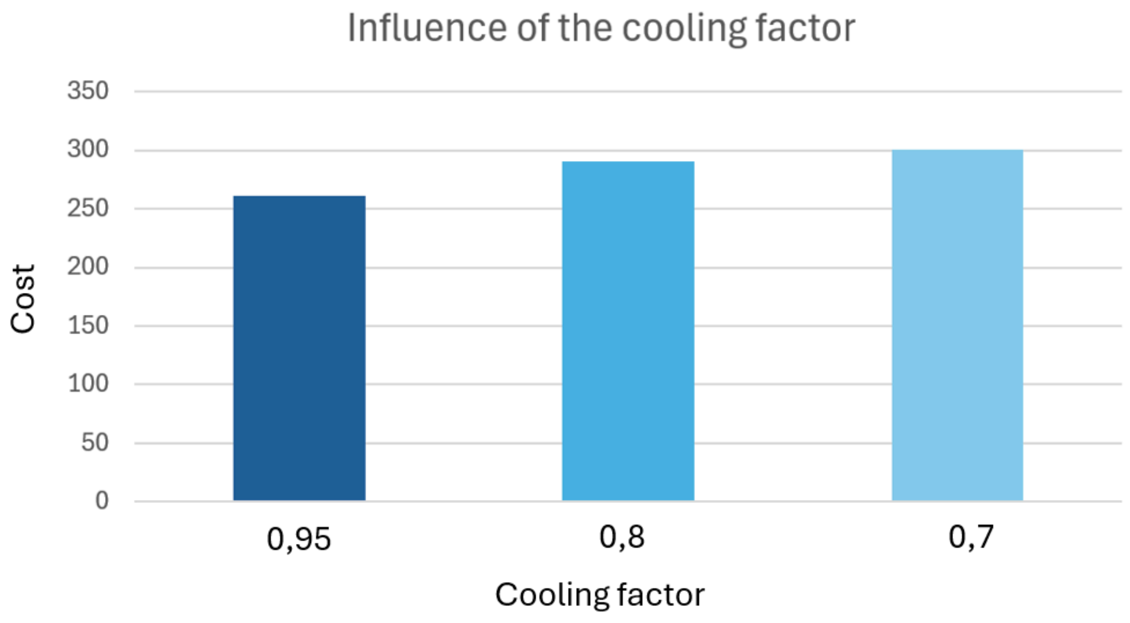


Figure 9 : Evolution of the cost of the best solution according to the cooling factor on a small database

II.2.2.2. Result analysis

Best neighbor strategy :

We can see that all the neighborhood strategies improve the initial scheduling. However, the "Random move" strategy achieves the best result with a minimized cost function around 260.

Best cooling factor:

The choice of cooling factor seems to be important. Indeed, with a cooling factor equal to 0.95, the cost is around 260, whereas with a cooling factor equal to 0.7, the cost is around 300. The range giving the best results is around 0.90 for the cooling factor.

Number of iterations and initial temperature :

Increasing the number of iterations improves the result. For 300 iterations, we obtain a better cost at 280 and for 1500 iterations a better cost at 255. We didn't notice any significant changes when we changed the initial temperature.

II.2.2.3. Relevance of the study

This initial study on a small database has enabled us to verify that the programs are operational. It also gives an initial idea of the influence of the choice of neighborhood methods and initial parameters. The method of randomly moving a client and a cooling factor of around 0.90 seems to be optimal. This needs to be verified on a larger database.

II.2.3. Large Database

II.2.3.1. Results

For the large database, we use the set of customers to be delivered from route 1.

Comparison of neighborhood strategies :

For the following parameters, we obtain the graph below:
temperature_initiale=100,iterations_par_cycle=1500,facteur_refroidissement=0.95

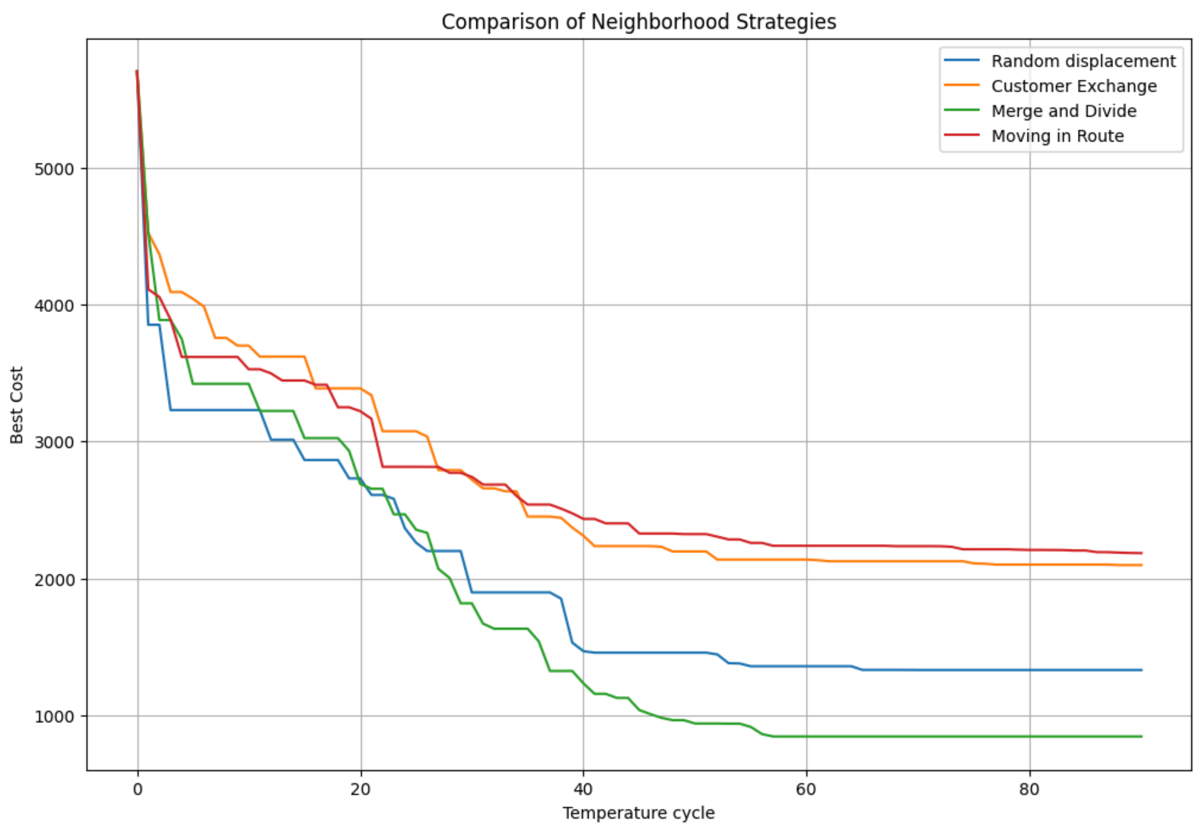


Figure 10 : Comparison of four neighborhood methods for simulated annealing on a big database

Result modification cooling factors :

For the following parameters, we obtain the graph below: temperature_initiale=100,iterations_par_cycle=1500,facteur_refroidissement=0.8

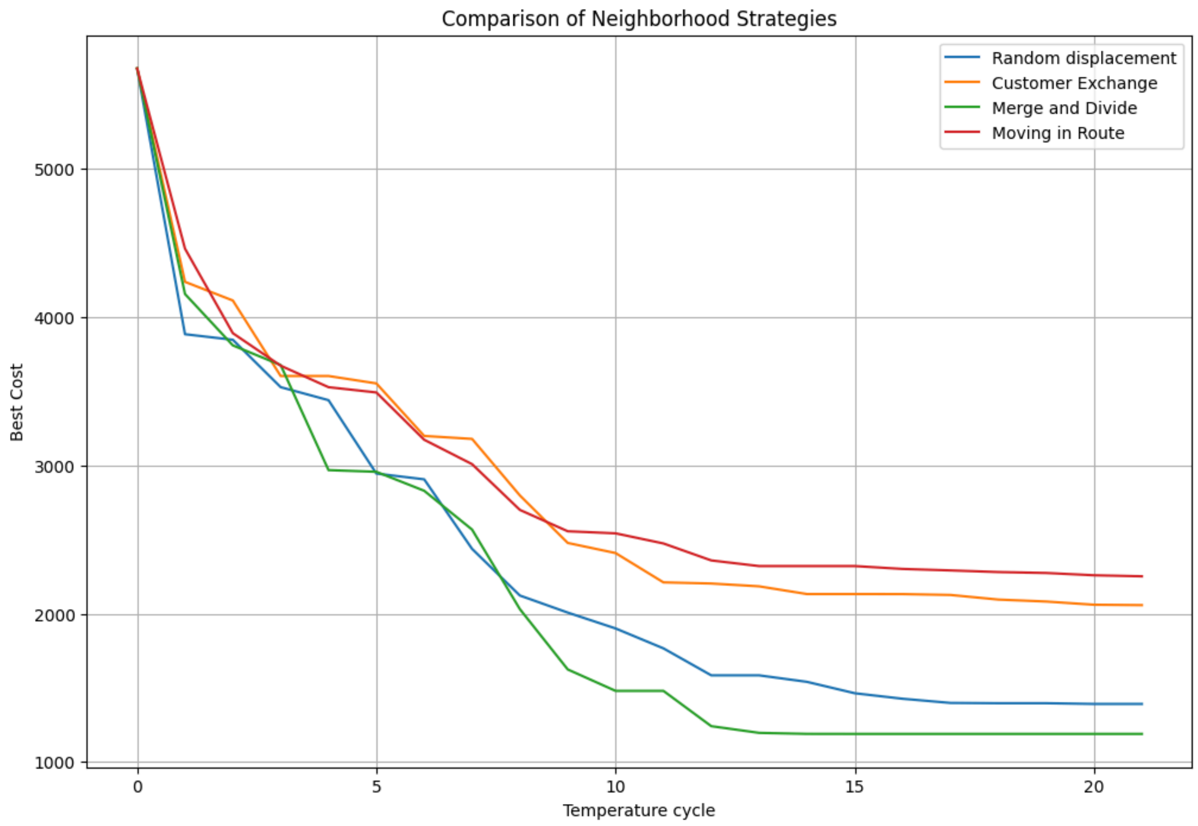


Figure 11 : Evolution of the cost of the best solution for the four neighborhood methods with a cooling factor of 0.8 on a big database

For the following parameters, we obtain the graph below:
temperature_initiale=100,iterations_par_cycle=1500,facteur_refroidissement=0.7

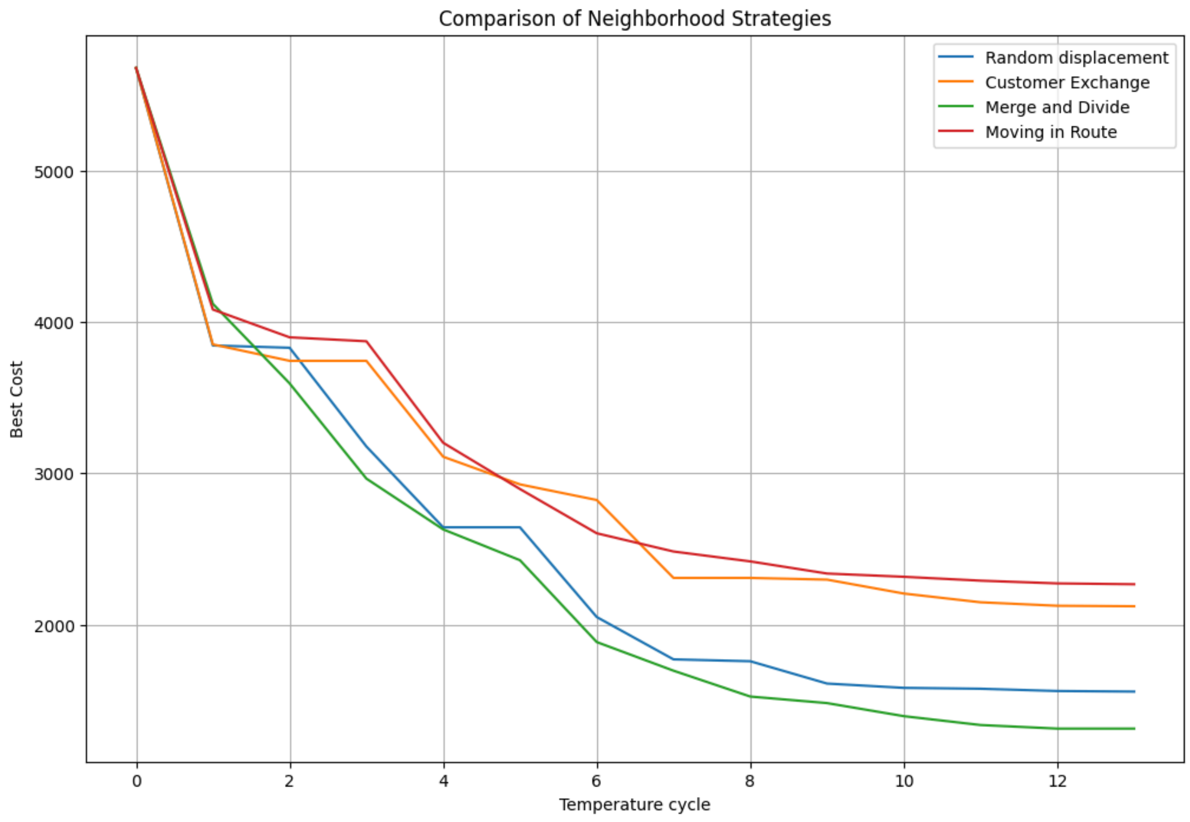


Figure 12 : Evolution of the cost of the best solution for the four neighborhood methods with a cooling factor of 0.7 on a big database

The results of the influence of the cooling factor are summarized in a graph.

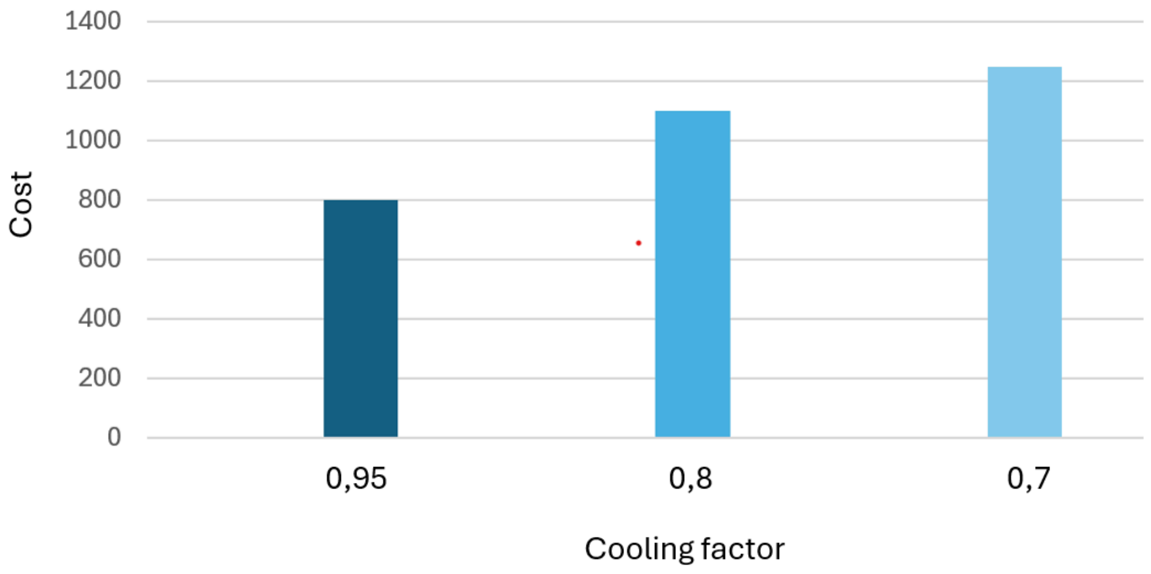


Figure 13 : Evolution of the cost of the best solution according to the cooling factor

II.2.3.2. Results analysis

Best neighborhood strategy:

For the large database, the "Merge and Divide" strategy achieves the best result, with a minimized cost function of around 800. In fact, this is the strategy that enables the fastest shuffling of schedules by moving an entire sequence of commands.

The size of the database seems to be an important criterion to consider when choosing the right neighborhood strategy. Indeed, on the small-sized database, it was the random move method that gave the best results.

Best cooling factor :

The choice of cooling factor is also very important. Indeed, with a cooling factor equal to 0.95, the cost is around 800, whereas with a cooling factor equal to 0.7, the cost is around 1250. The range giving the best results is around 0.90 for the cooling factor.

Number of iterations and initial temperature :

Increasing the number of iterations improves the result. For 300 iterations, we obtain a better cost at 1600 and for 1500 iterations a better cost at 800. We didn't notice any significant changes when we changed the initial temperature.

II.2.3.3. Interest of the study

This study enables us to determine the optimal input parameters on a large database. This will be used, among other things, to parameterize our agents in the multi-agent system. The "Merge and divide routes" neighborhood method seems to be the best, with a cooling rate of around 0.9.

II.3. Genetic algorithm

II.3.1. Specific VRP algorithm

The genetic algorithm is a heuristic research technique inspired by biological evolution. This approach uses a population of individuals to solve a given problem.
The algorithm begins by creating an initial population of individuals, where each individual represents a possible solution to the problem.

The genetic algorithm continues these stages of selection, crossover, and mutation until a stopping criterion is reached, such as the maximum number of iterations:

1. A tournament **selection** feature is used to select parents for breeding. In this function, a number of individuals are randomly selected from the population. The parents selected are the ones with the lowest cost.
2. The principle of **crossover** is inspired by the sexual reproduction of living beings. The idea is to create new individuals by combining the genes of two parents to create an offspring. In the context of the problem of vehicle routing, the candidates are represented as a list of lists, where each sublist corresponds to a tour made by a vehicle. The process is to take two routes and transform them into ordered sequences of clients by flattening them. Then, we select and swap two random parts between the two sequences. As a result, we get two new sequences that we use to generate two new child routes
3. A **mutation** function is rarely used to simulate the natural process by slightly modifying the children (neighborhood function) to avoid convergence to a local minimum.

5 parameters are required to run the genetic algorithm which are **nb_gen** (the size of the initial population), **nb_iter** (the number of itérations), f_cost (the cost function used), **P_cross** (the probability of crossover) and **P_mut** (the probability of mutation). Here is the pseudo-code of this algorithm :

```
genetic(nb_gen, nb_iter, f_cost, P_cross, P_mut)
    P ← [ ]                                          # list of random schedulings
    nb_client ← len(clients)
    for _ in range(nb_gen) do                        # creating random solutions
        s ← generate_random_scheduling(nb_client, len(vehicles))
        while not is_scheduling_valid(s) do
            s ← generate_random_scheduling(nb_client, len(vehicles))
        end while
        P.append(s)
    end for
    best_solution ← min(P, key=lambda x : f_cost(x))
    hist_costs ← [f_cost(best_solution)]             # storing the best costs
    St_1 ← P
    for _ in range(nb_iter) do
        St ← selection(St_1, f_cost)                 # stage 1
        St_1 ← [ ]
        for i in range(nb_gen//2) do                 # stage 2
            if random() < P_cross then
                A, B ← crossover(St[2i], St[2i+1], nb_client)
                while not is_scheduling_valid(A) or not is_scheduling_valid(B)
    do
                    A, B ← crossover(St[2i], St[2i+1], nb_client)
                end while
                St_1.append(A)
                St_1.append(B)
            end if
            else do
                St_1.append(St[2i])
                St_1.append(St[2i+1])
            end else
        end for
        for i in range(nb_gen) do                        # stage 3
            if random() < P_mut then
                m_i ← mutation(St_1[i], nb_client)
                while not is_scheduling_valid(m_i) do
                    m_i ← mutation(St_1[i], nb_client)
                end while
                St_1[i] ← m_i
            end if
        end for
        best_solution ← min(St_1, key=lambda x : f_cost(x))
        hist_costs ← f_cost(best_solution)
    end for
    plot(hist_costs)
    m ← 0
    m_e ← f_cost(St_1[0])
    for k in range(1, nb_gen) do
        e ← f_cost(St_1[k])
        if e < m_e then
            m ← k
            m_e ← e
        end if
    end for
return m_e, St_1[m]
```

II.3.2. Small Database

In the same way as with the previous algorithms, the database is first reduced to 20 clients. The objective of this first approach is to get a general overview of the influence of each parameter. Below is an example of the evolution of the best cost of the scheduling through each iteration.

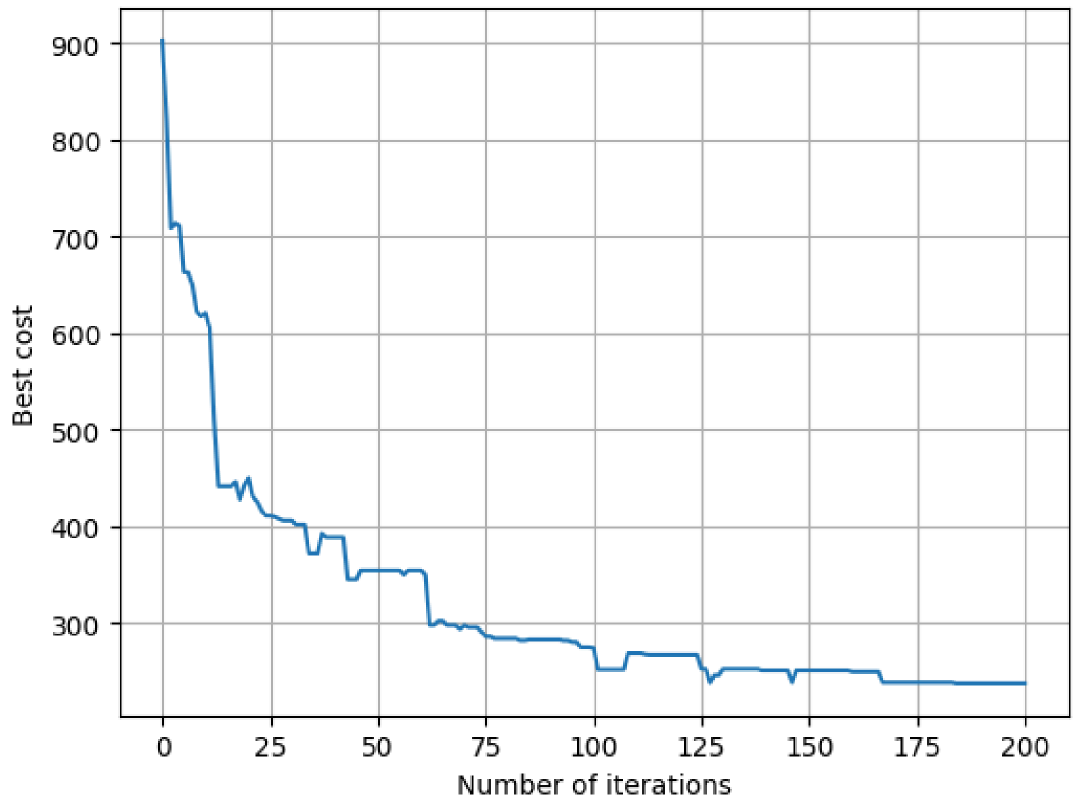


figure 14 : Evolution cost of the genetic algorithm

II.3.2.1. Results

Among the 5 parameters used for the genetic algorithm, only the cost function won't change. As a consequence, it is useful to test different values for the other parameters in order to find out the best ones to get the most optimized solutions possible. In the following tests, we vary one parameter at a time and fix the others.

First, nb_gen represents the number of the solutions considered. The higher nb_gen the more diverse the solutions will be. Thus one can expect better results when increasing this parameter.

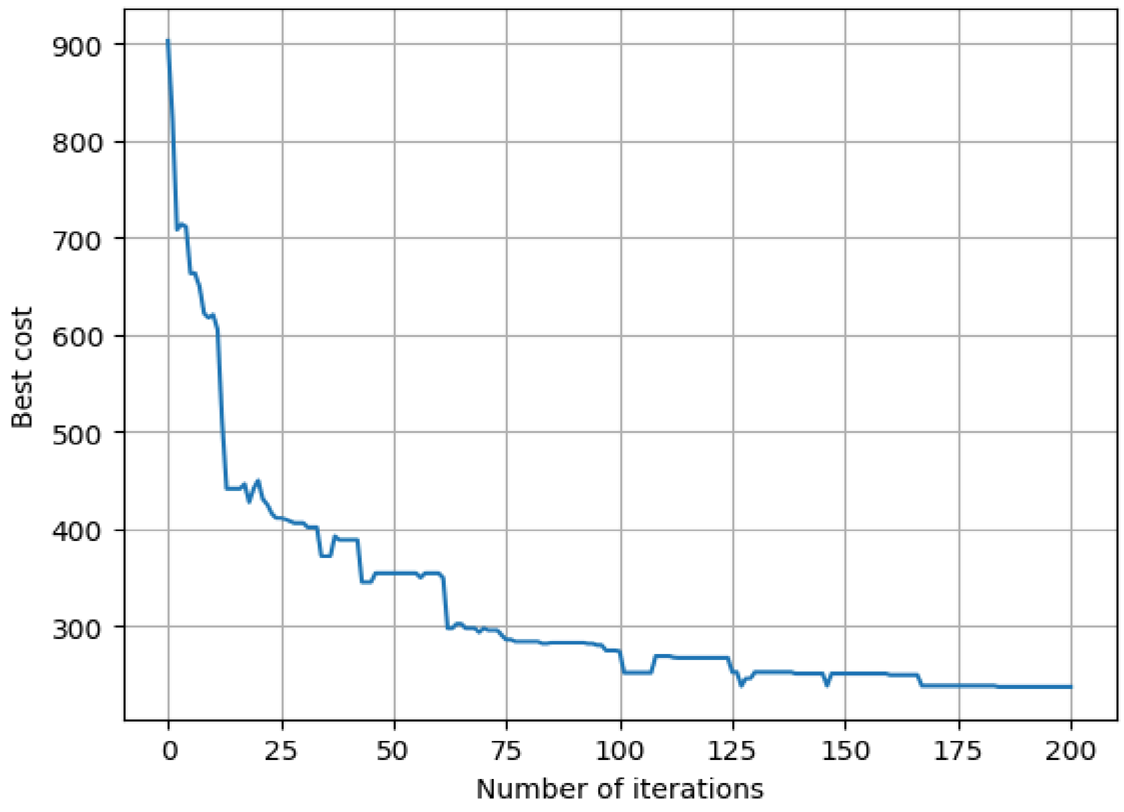


figure 15 : Results with nb_gen=100

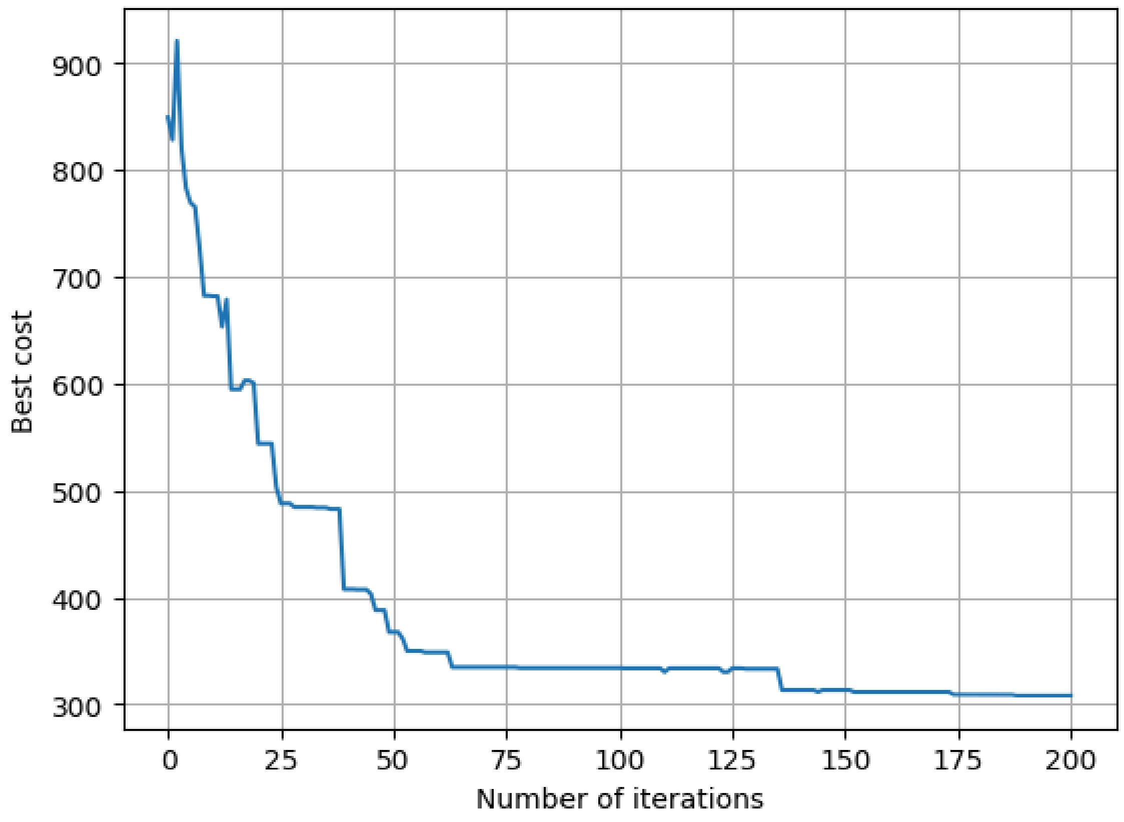


figure 16 : Results with nb_gen=50

Secondly, nb_iter is the parameter that determines the number of iterations of the 3 stages described above. Each iteration offers the possibility to explore new solutions and thus to find better ones.

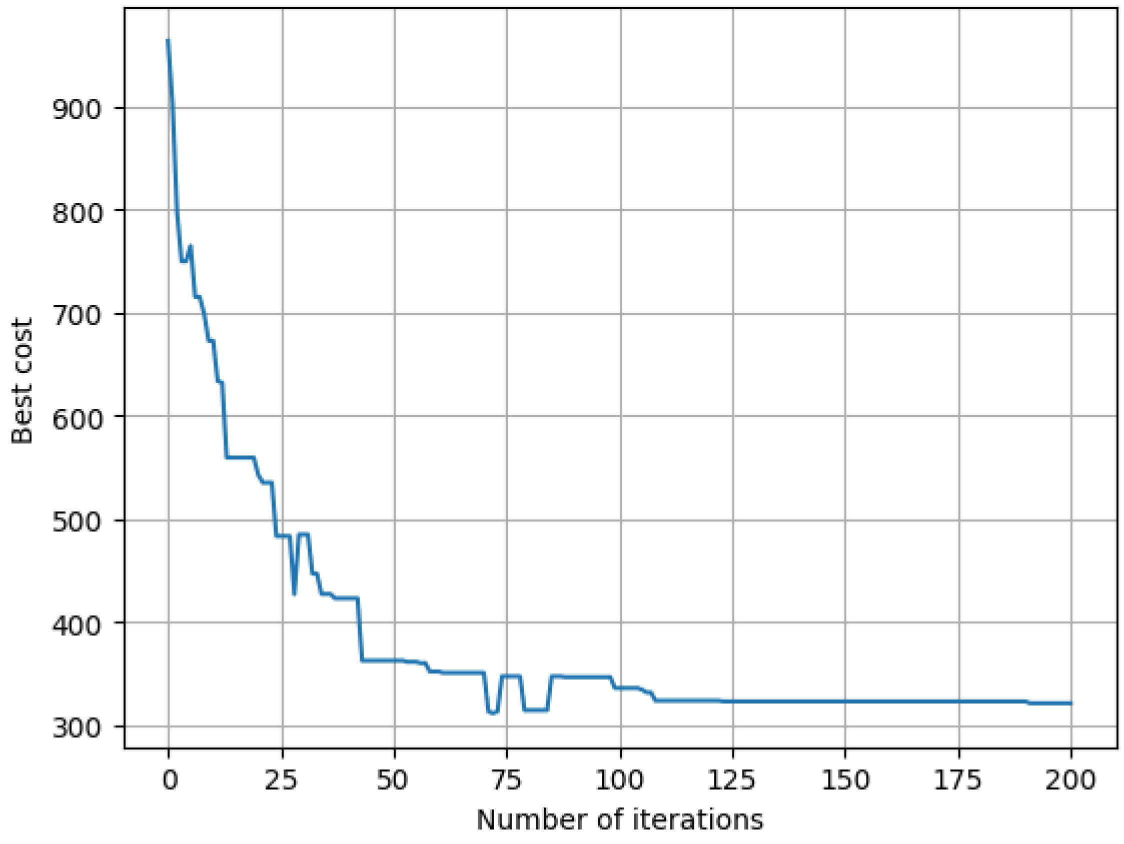


figure 17 : Results with nb_iter=200

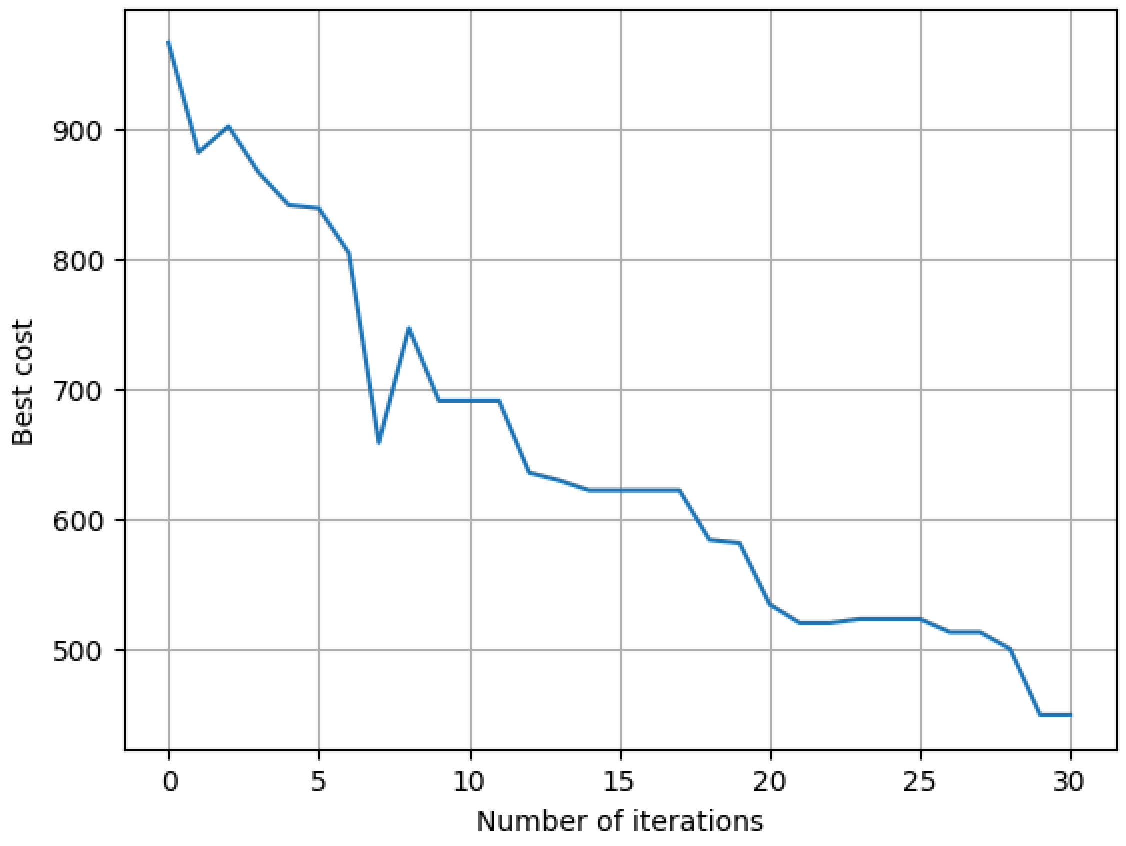


figure 18 : Results with nb_iter=30

Thirdly, P_cross is the probability of triggering the crossover function. As the crossover function is the base of the genetic algorithm, one can expect that the higher P_cross, the better the solution found.

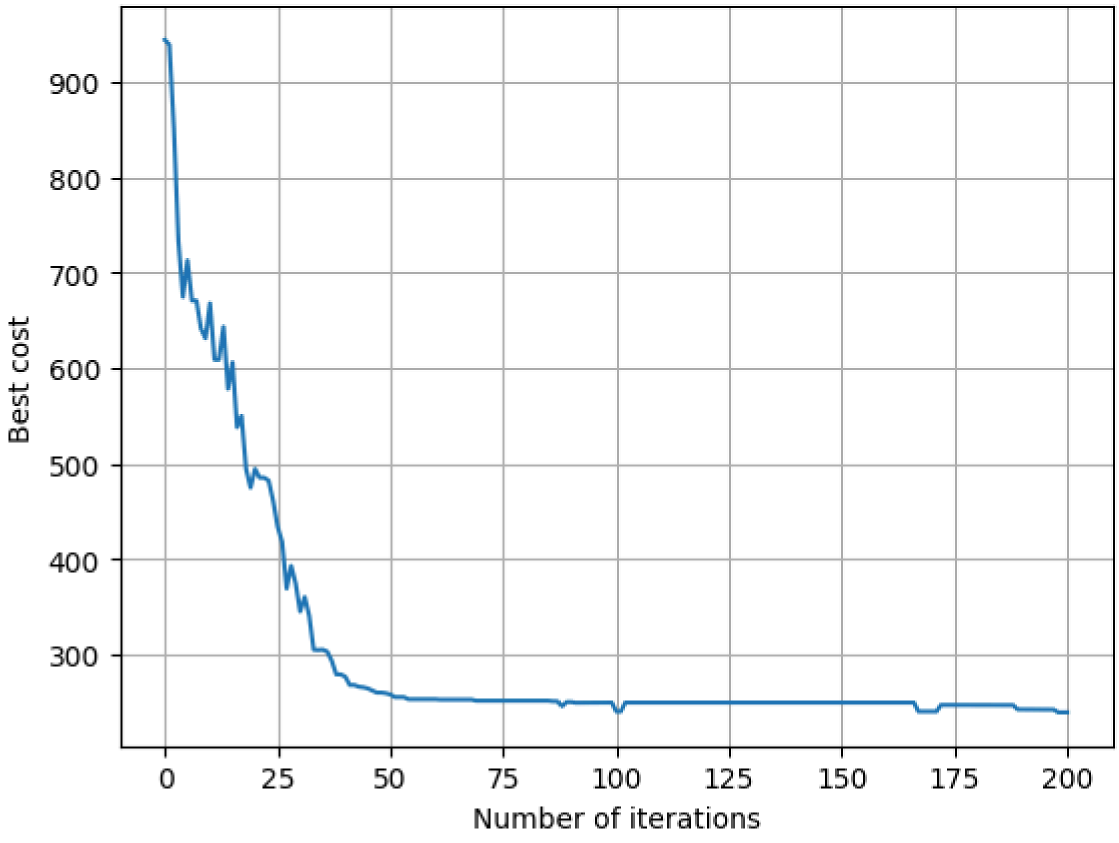


figure 19 : Results with P_cross=1

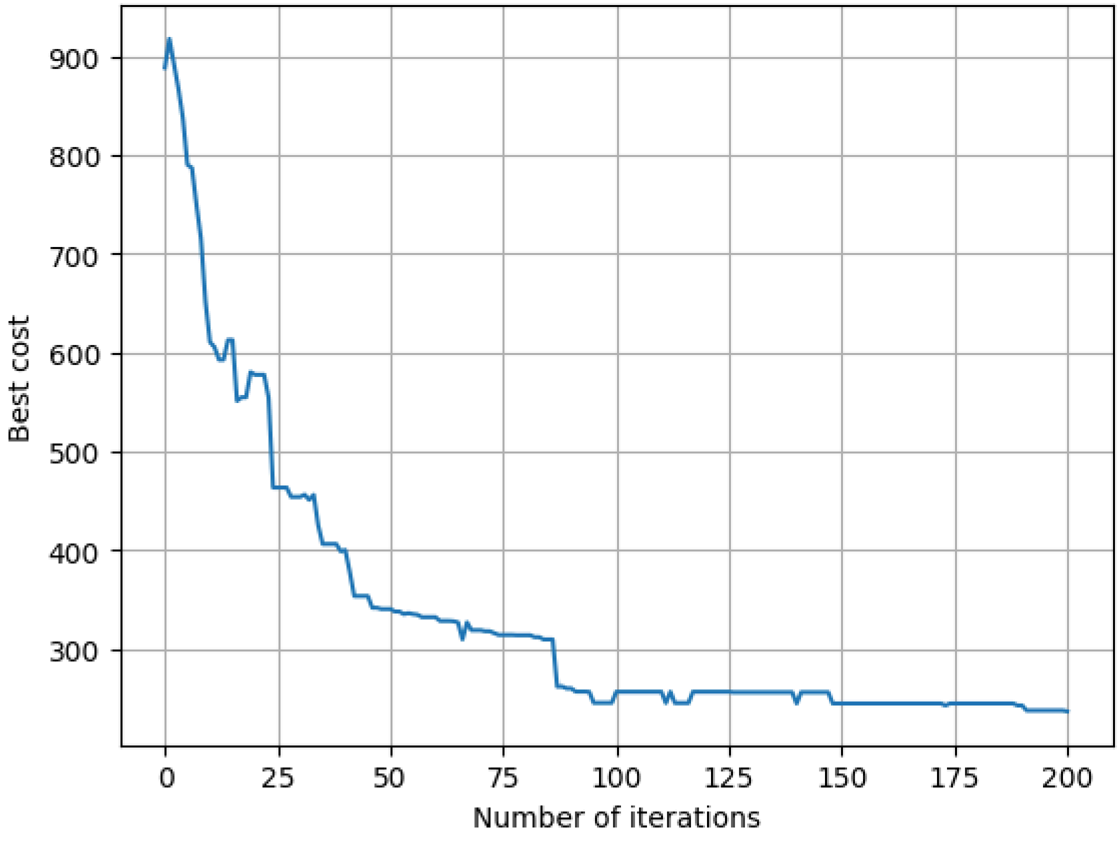


figure 20 : Results with P_cross=0.7

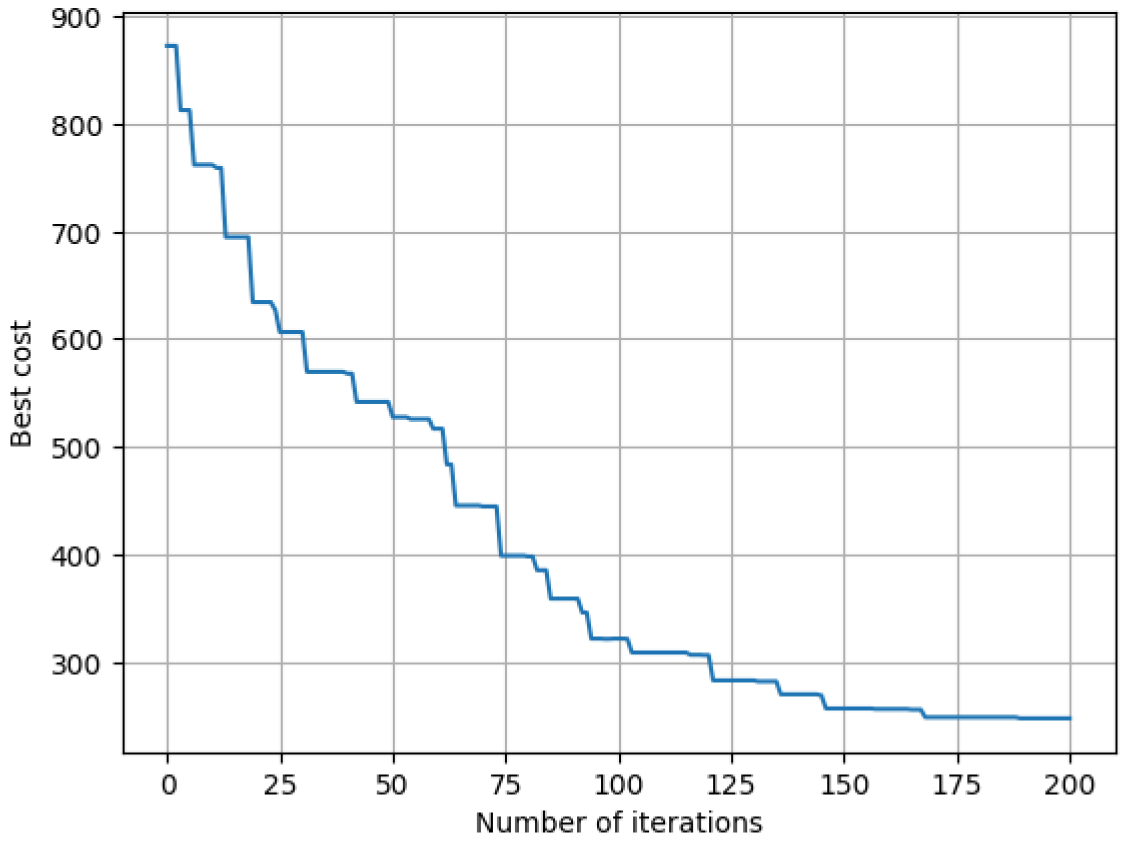


figure 21 : Results with P_cross=0

Finally, P_mut is the probability to trigger the mutation function which guarantees the diversity of the solutions. Consequently, a low probability should be the best choice because a higher value could prevent the algorithm from converging and removing the mutation function could lead to converge to a local minimum instead of the global one.

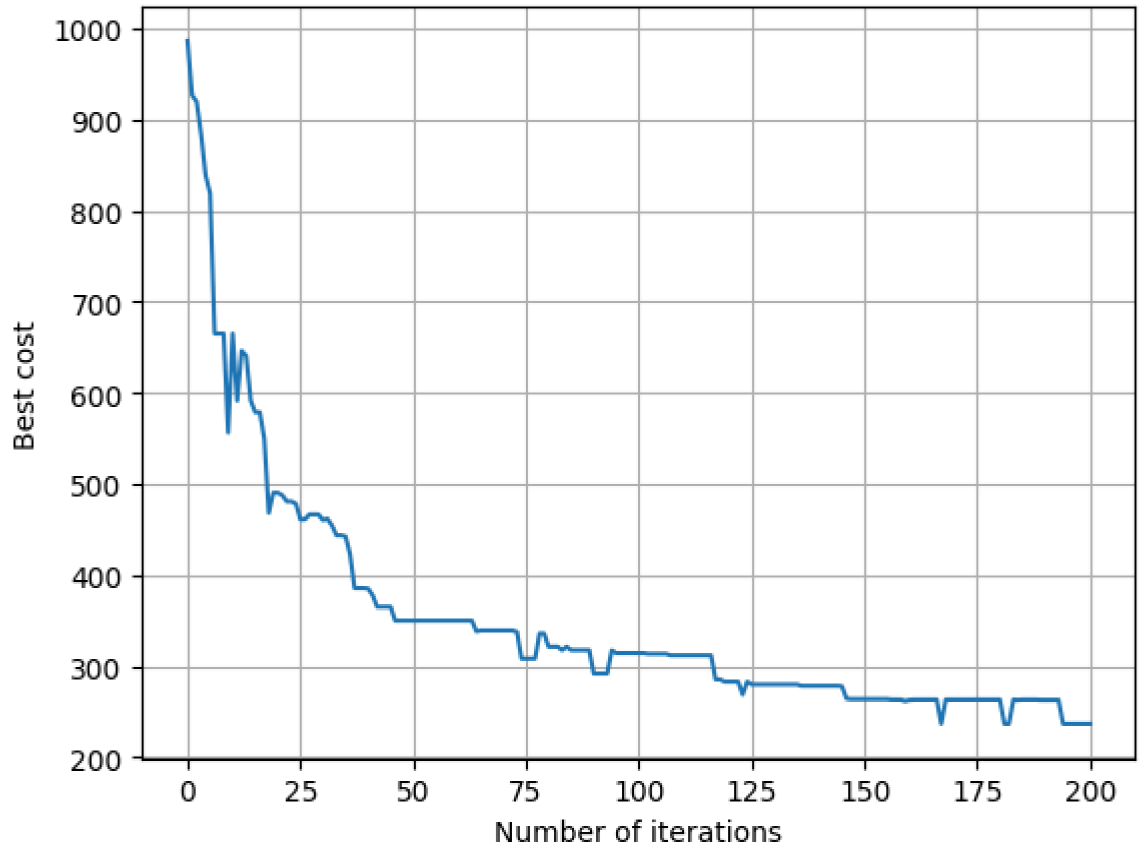


figure 22 : Results with P_mut=0.1

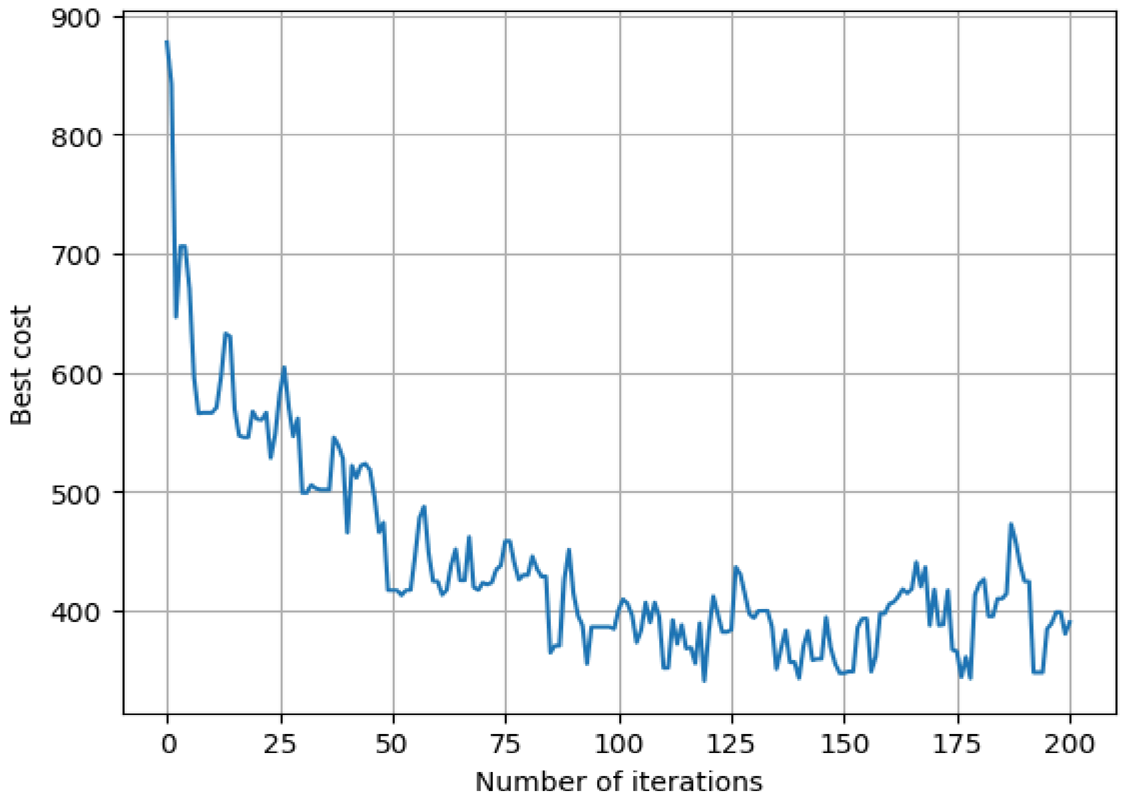


figure 23 : Results with P_mut=1

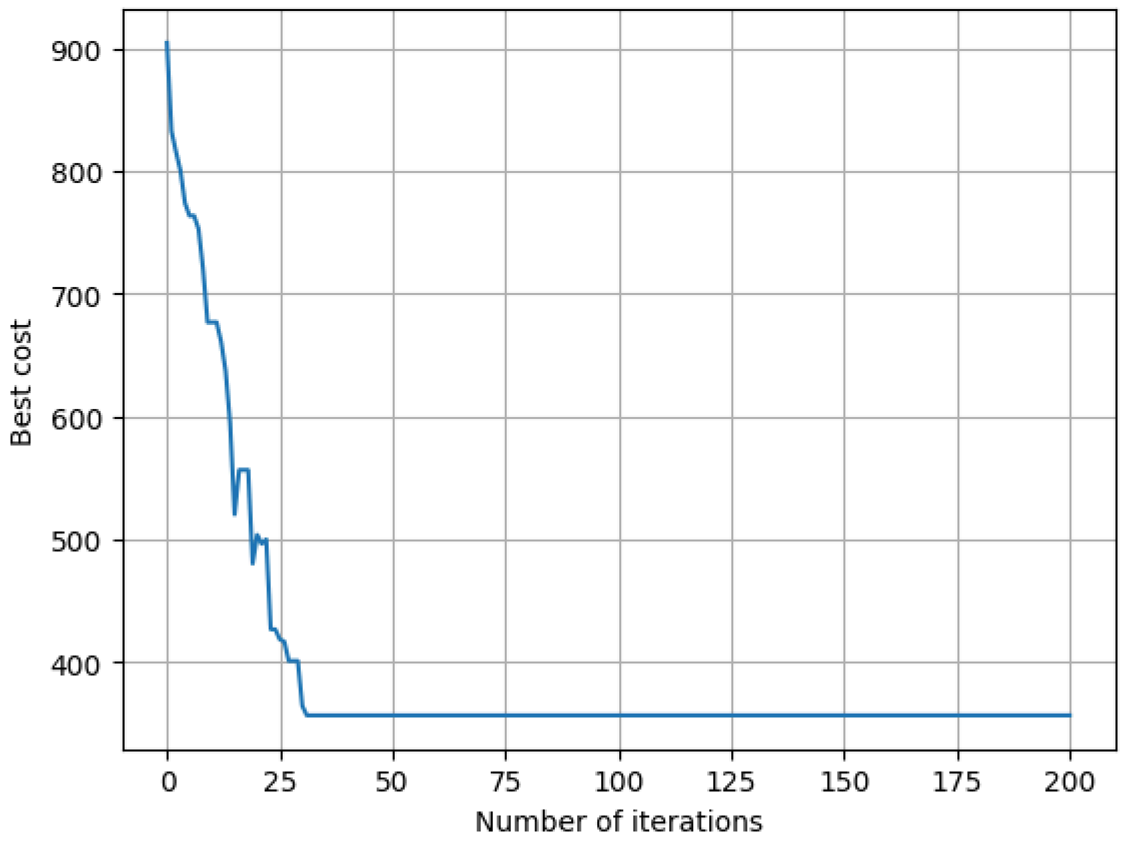


figure 24 : Results with P_mut=0

II.3.2.2. Results analysis

The previous tests gave a few indications about the influence of each parameter. One can see that having a higher initial population gives better results as it enables the algorithm to explore more possibilities. Moreover, if the number of iterations of the 3 stages is not enough, the algorithm cannot converge and thus cannot provide an optimized scheduling.
Then, the value of P_cross has an influence over the convergence speed of the algorithm but surprisingly does not change the best score found that much. Finally, P_cross is necessary to guarantee the diversity of the solutions explored. As seen in the last test, if its value is 0 the algorithm converges quickly but finds an unoptimized solution and cannot improve it due to the lack of diversity in its population. Besides, choosing a value of P_mut too high will result in the impossibility to converge.

II.3.2.3. Relevance of the study

This first approach allows us to have a better understanding of how the genetic algorithm works and what is the influence of each of its parameters. Indeed, after these tests, It is possible to set the values of these parameters by default. For the next part of this study, which consists in carrying the same kind of tests with a larger Database, we set by default the following values :

- nb_gen = 100
- nb_iter = 200
- P_cross = 1
- P_mut = 0.1

II.3.3. Large Database

We are going to carry on the same tests on a larger Database in order to check whether the influence of the parameters are still the same or not on a larger scale.
The new Database consists of the clients from route 1 including 107 clients and a maximum of 8 lorries.

II.3.3.1. Results

Unlike the previous section, we are going to begin with nb_iter because it is essential to ensure the convergence of the algorithm before looking at the other parameters.

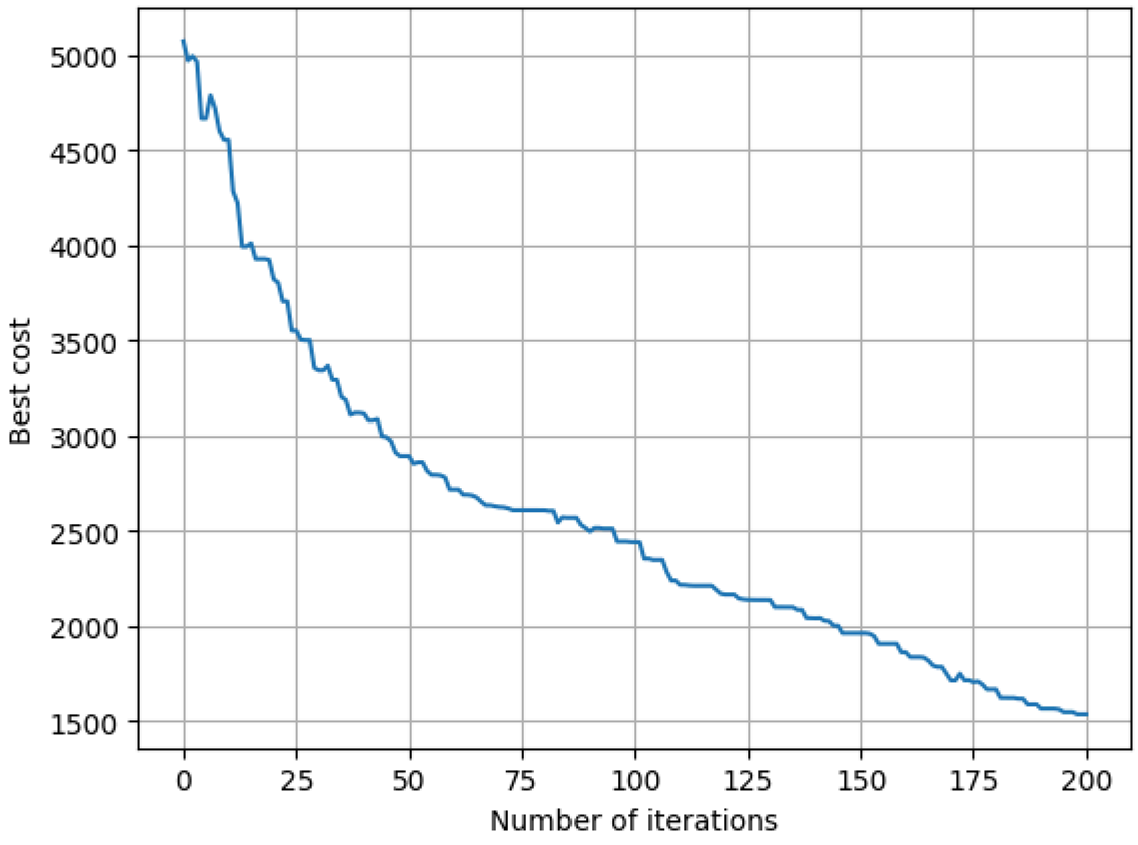


figure 25 : Results with nb_iter=200

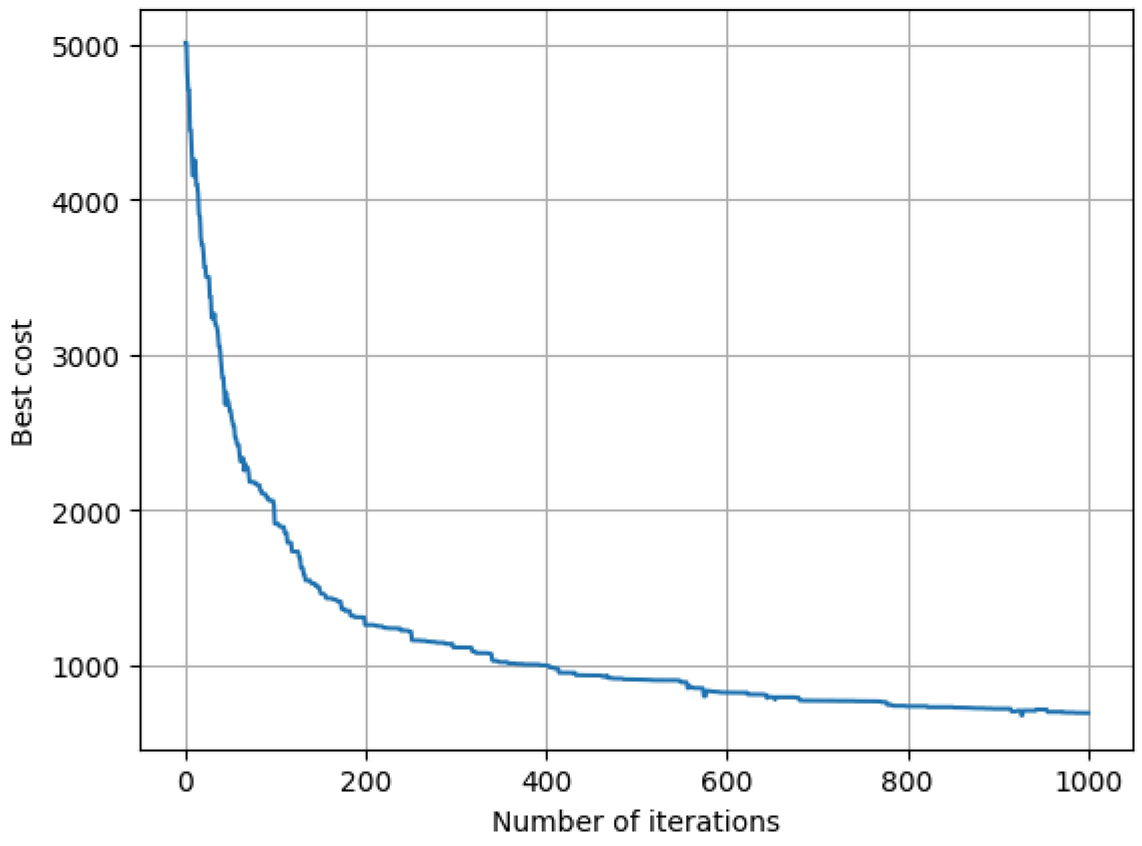


figure 26 : Results with nb_iter=1000

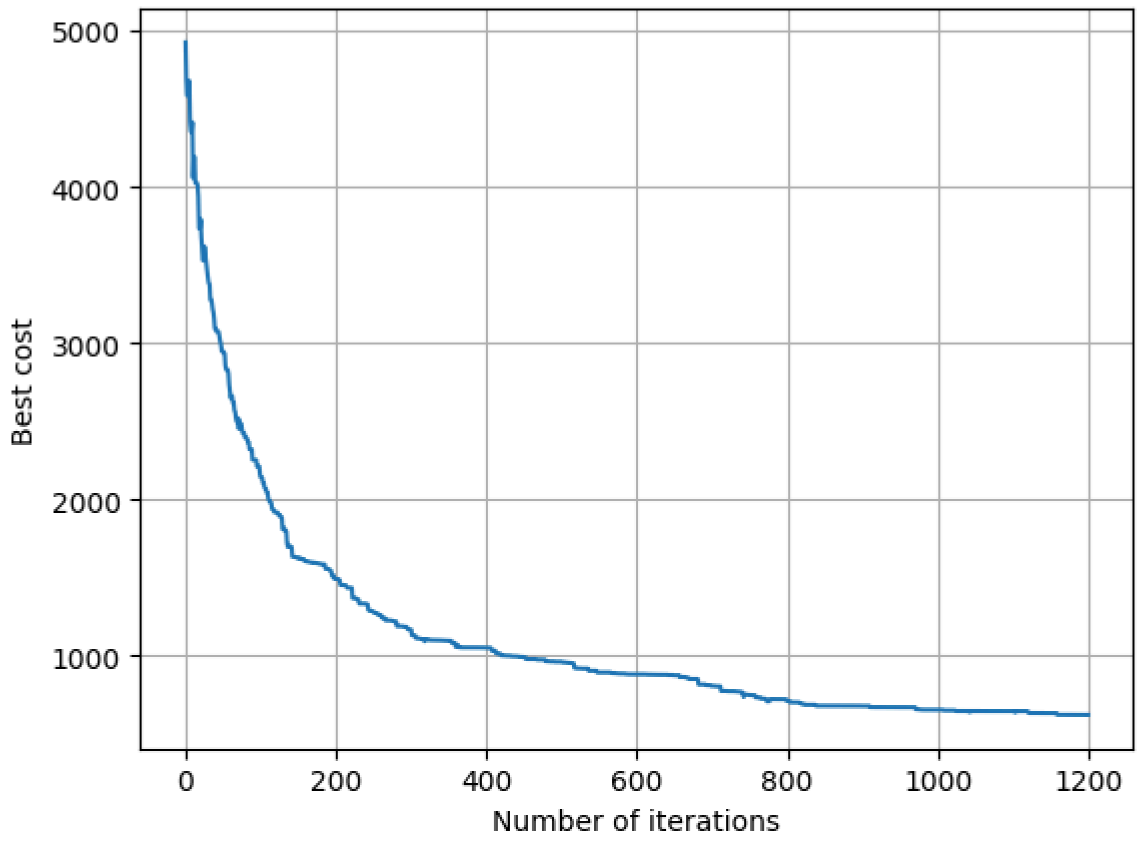


figure 27 : Results with nb_iter=1200

To ensure the convergence of the algorithm, one can clearly see that it needs at least 1000 iterations. Consequently, we will choose 1200 iterations for the next tests.

Now we can go back to the previous order to test each parameter and nb_gen is the next in the list.

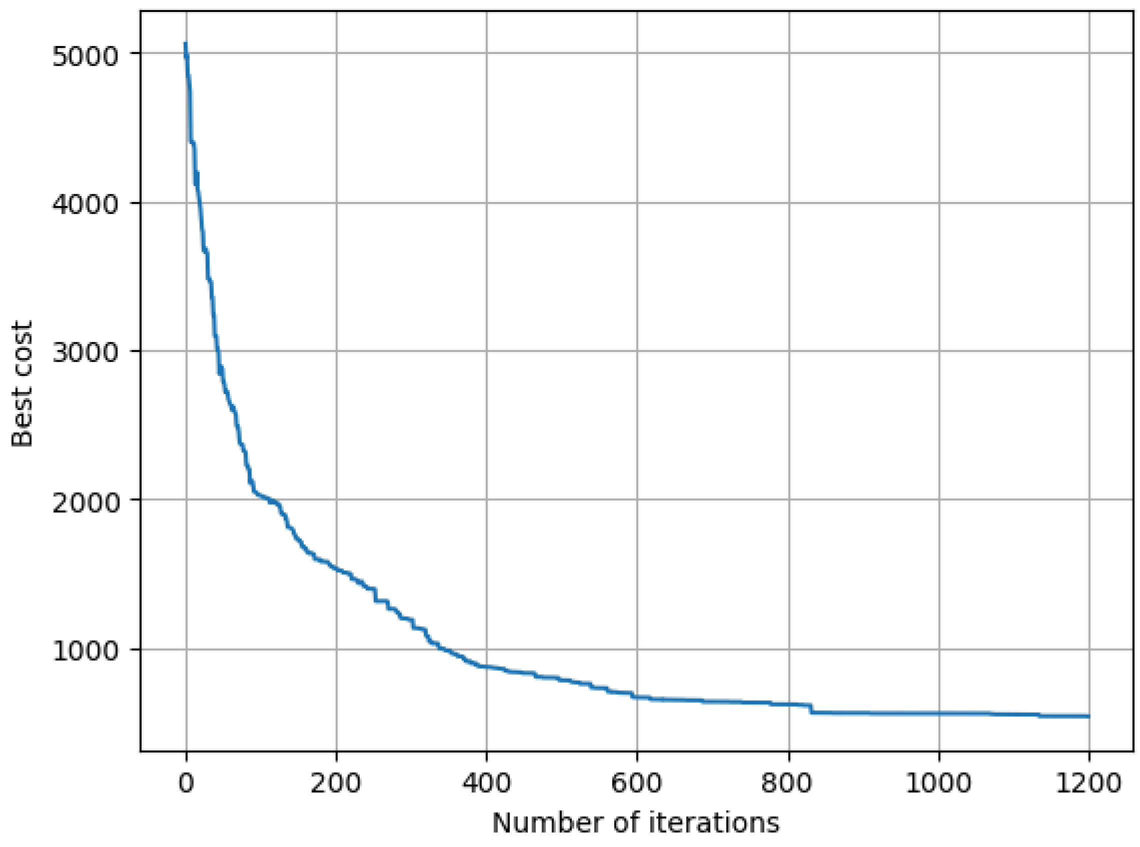


figure 28 : Results with nb_gen=100

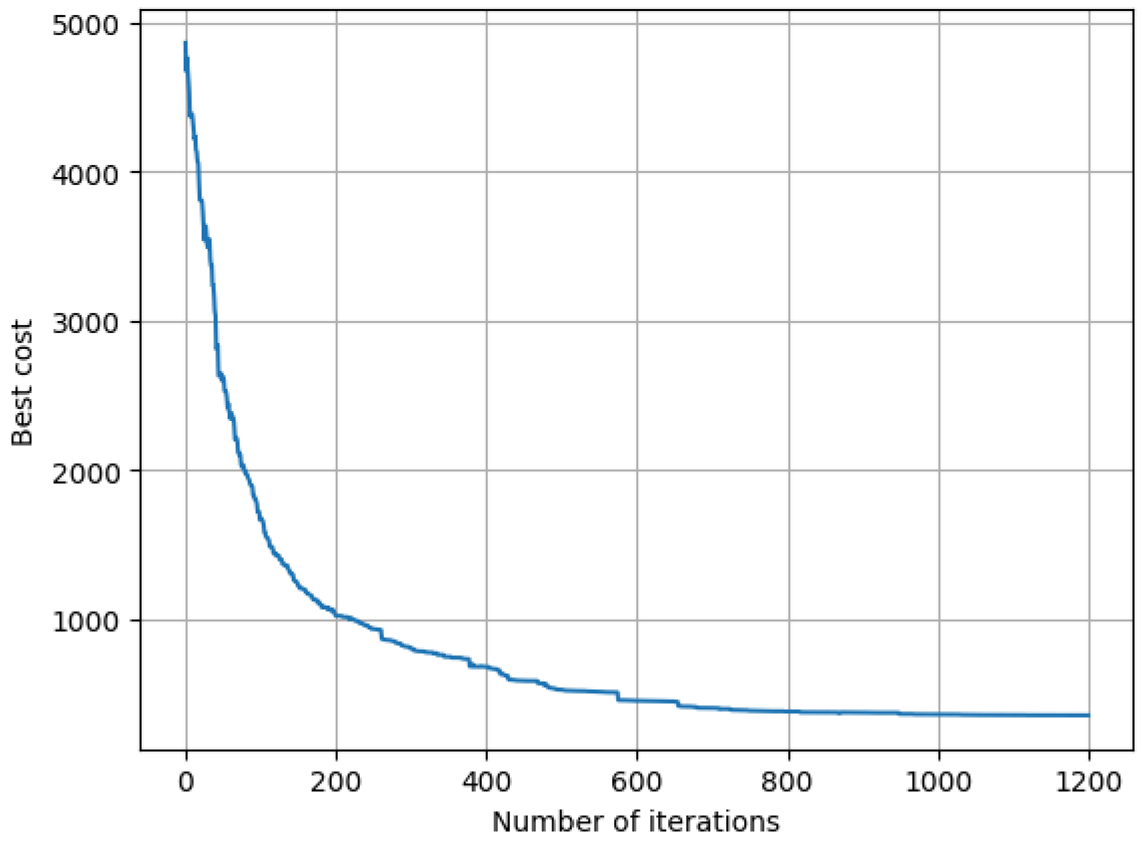


figure 29 : Results with nb_gen=400

Next is the tests of P_cross.

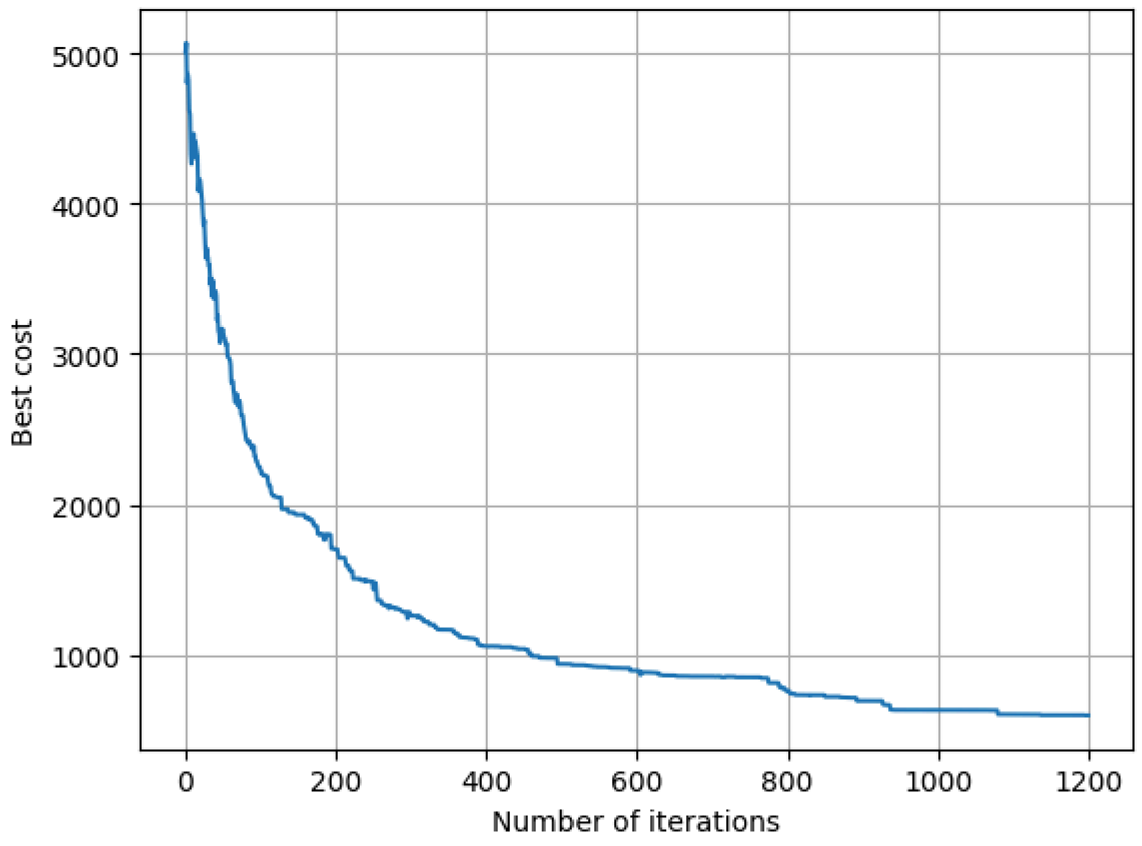


figure 30 : Results with P_cross=1

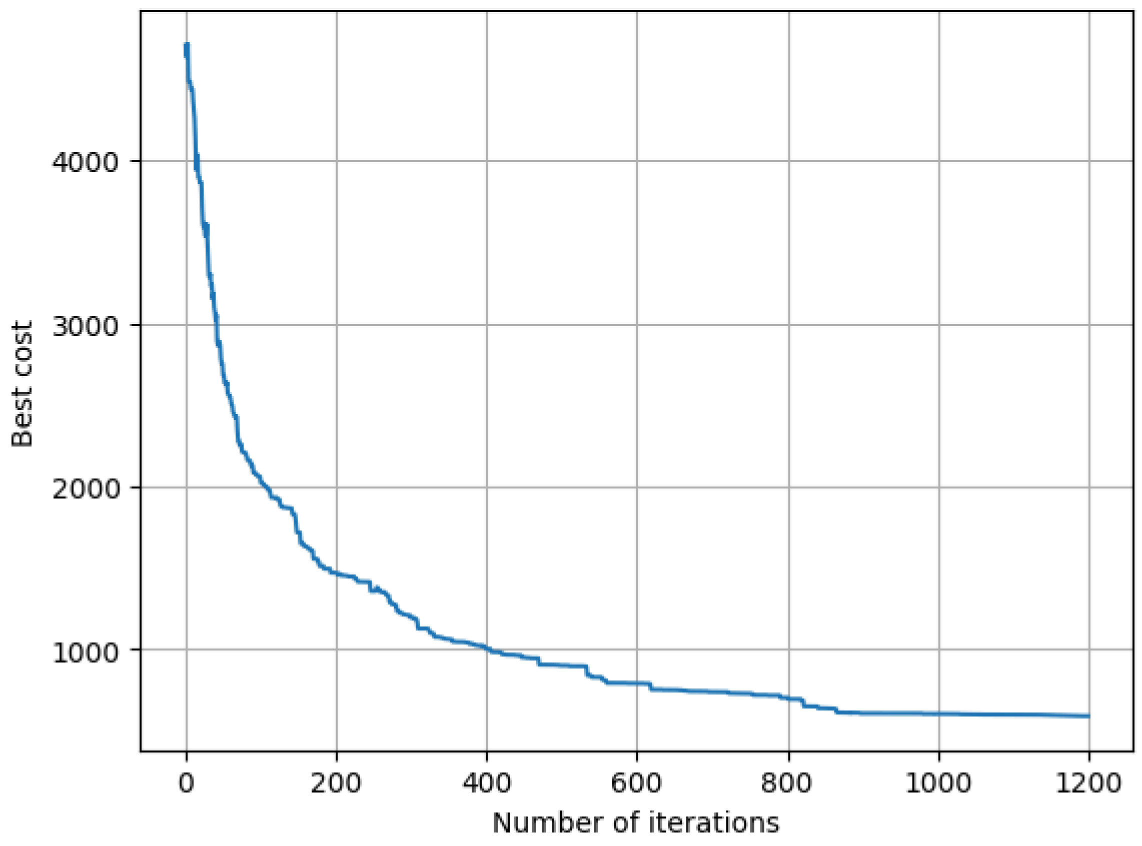


figure 31 : Results with P_cross=0.7

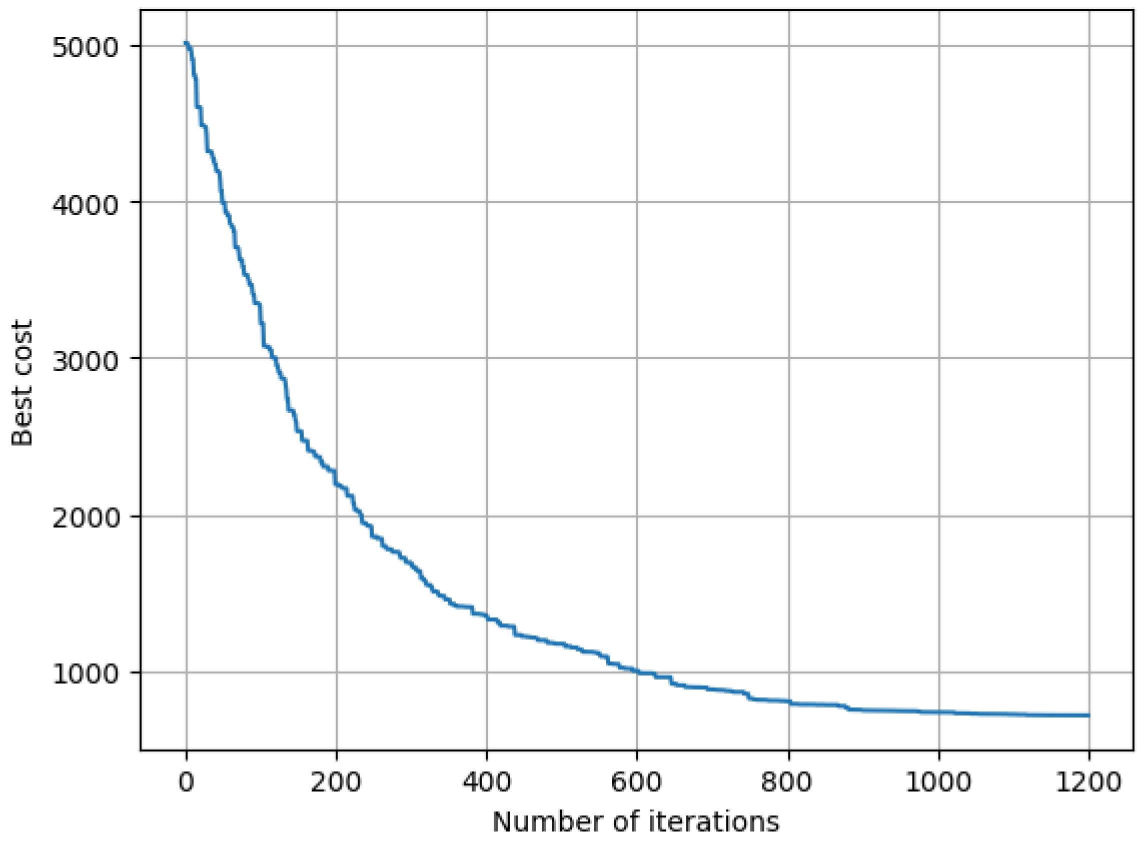


figure 32 : Results with P_cross=0

The last one is P_mut.

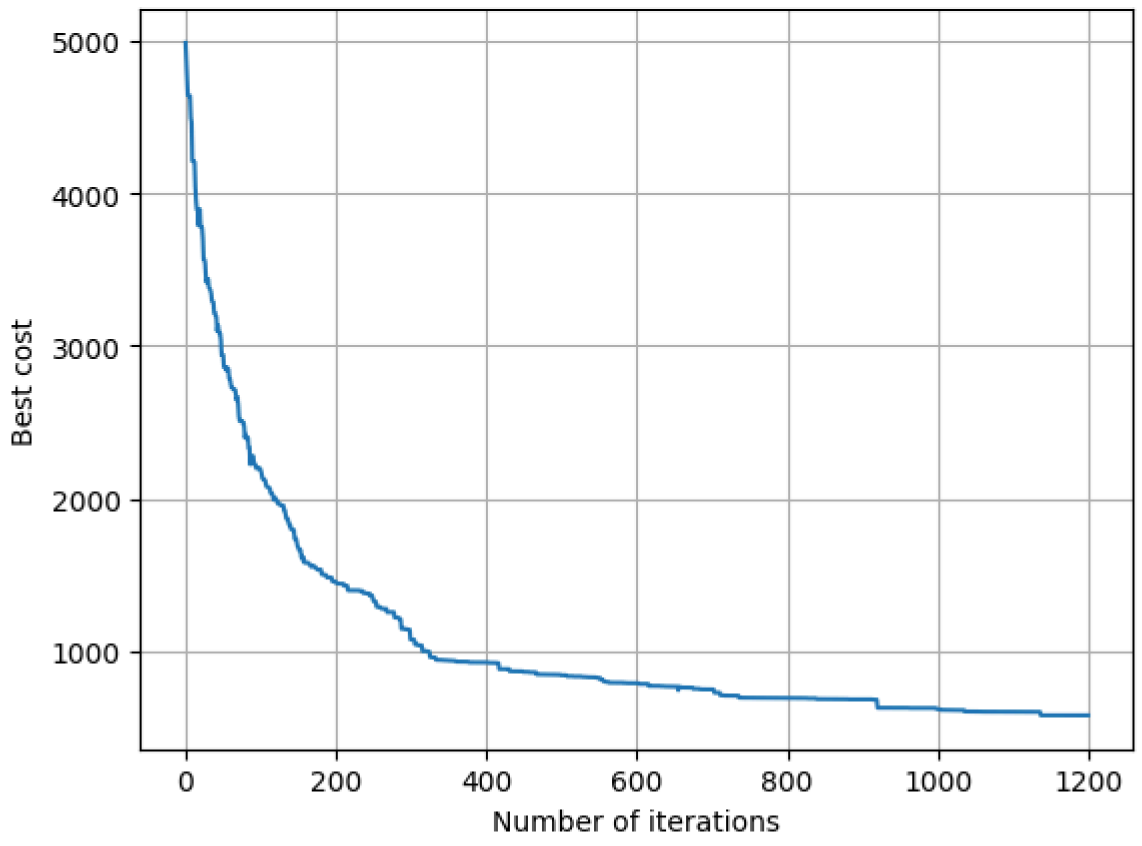


figure 33 : Results with P_mut=0.1

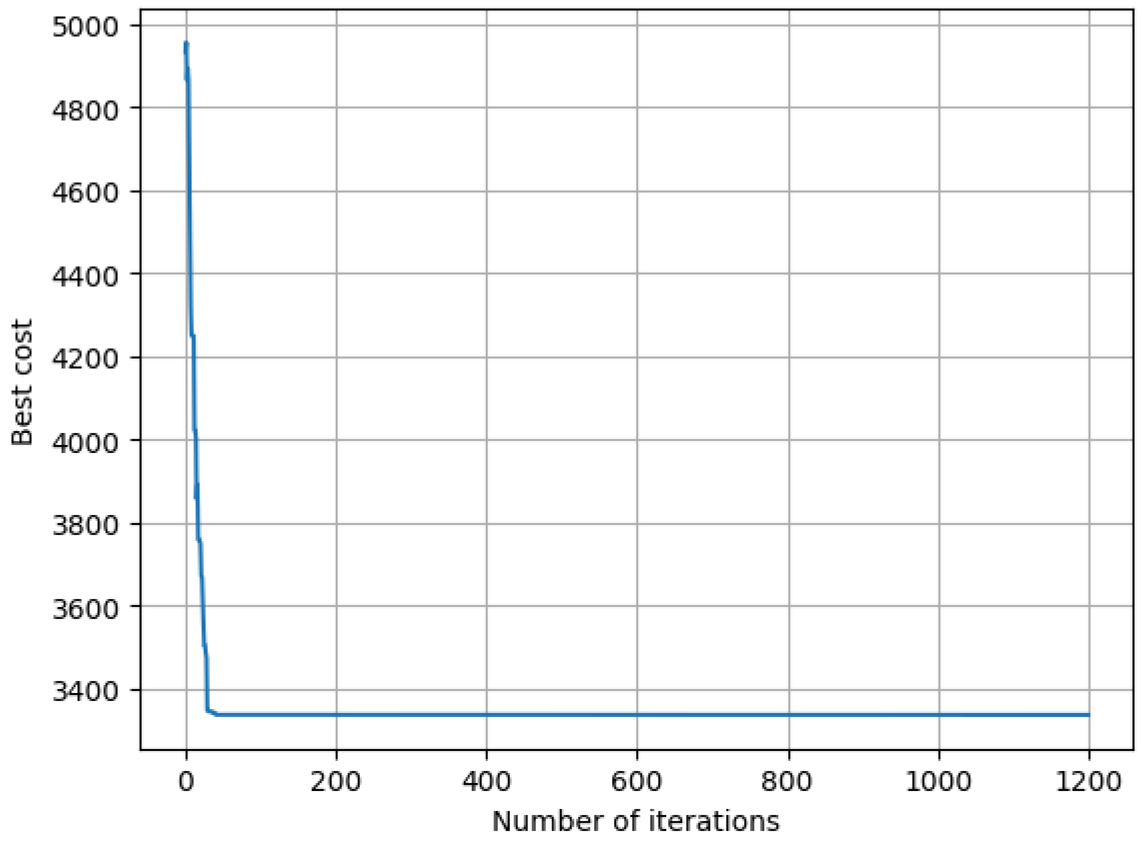


figure 34 : Results with P_mut=0

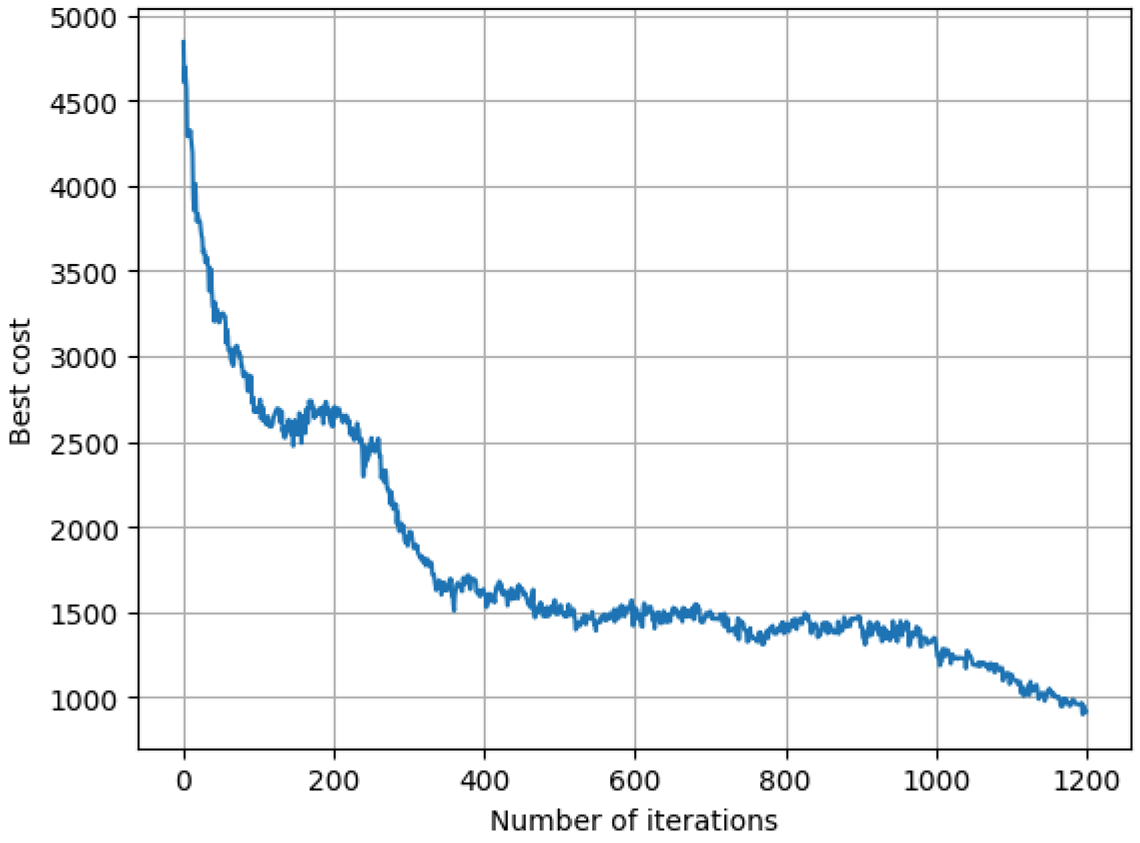


figure 35 : Results with P_mut=1

II.3.3.2. Analysis of the results

Again, we can make almost the same analysis as for the study of a small Database : P_mut must be low but not equal to zero, nb_iter must be large enough to enable the algorithm to converge, the higher nb_gen the best solution may be found.
Nonetheless, some new commentaries can be made.

- The minimum number of iterations required for the algorithm to converge depends on the number of clients to visit
- A higher value of nb_gen enables the algorithm to converge faster
- Different values of P_cross=1 may give better solutions if the number of clients increases.

II.3.3.3. Relevance of the study

We were able to discover some differences in the results of the genetic algorithm depending on the size of the problem. This will result in new default settings that are more in line with the problem we want to solve, which is the one of route1.

- nb_gen=400
- nb_iter=1200
- P_cross=0.7
- P_mut=0.1

II.4. Comparison of meta-heuristics

We have studied all the meta-heuristic algorithms. Now we will compare their performance. Figure 2 shows that the algorithms converge in different ways. The simulated annealing algorithm is slow at start-up, but converges rapidly after one second of execution. The taboo algorithm is the most efficient over this time window, but we can see from Figure 2, which studies the 3 algorithms over a slightly longer time, that the genetic algorithm converges to a more optimal solution.

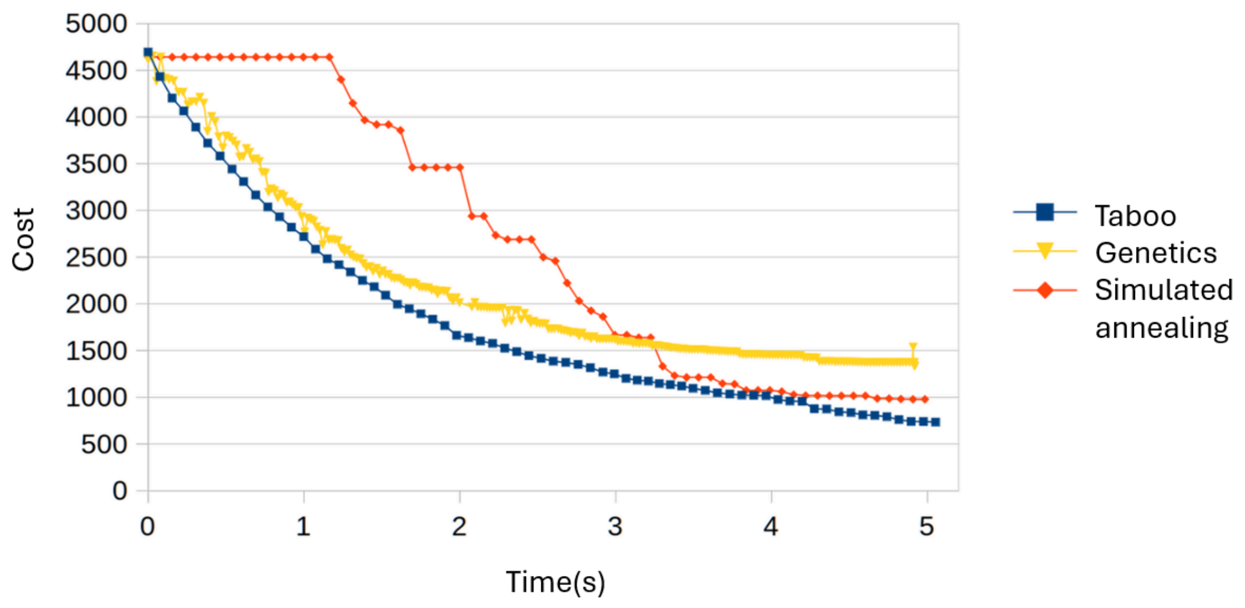


Figure 36 : Comparison of the meta-heuristics cost with their execution time

Figure 2 shows that these algorithms can divide the cost of a solution by 4 compared with a random solution. The most efficient over a 20-second period is the genetic algorithm, followed by the taboo algorithm and finally the simulated annealing algorithm.
However, no algorithm should be abandoned. In the next section, we will look at how to exploit their respective advantages to obtain better solutions.

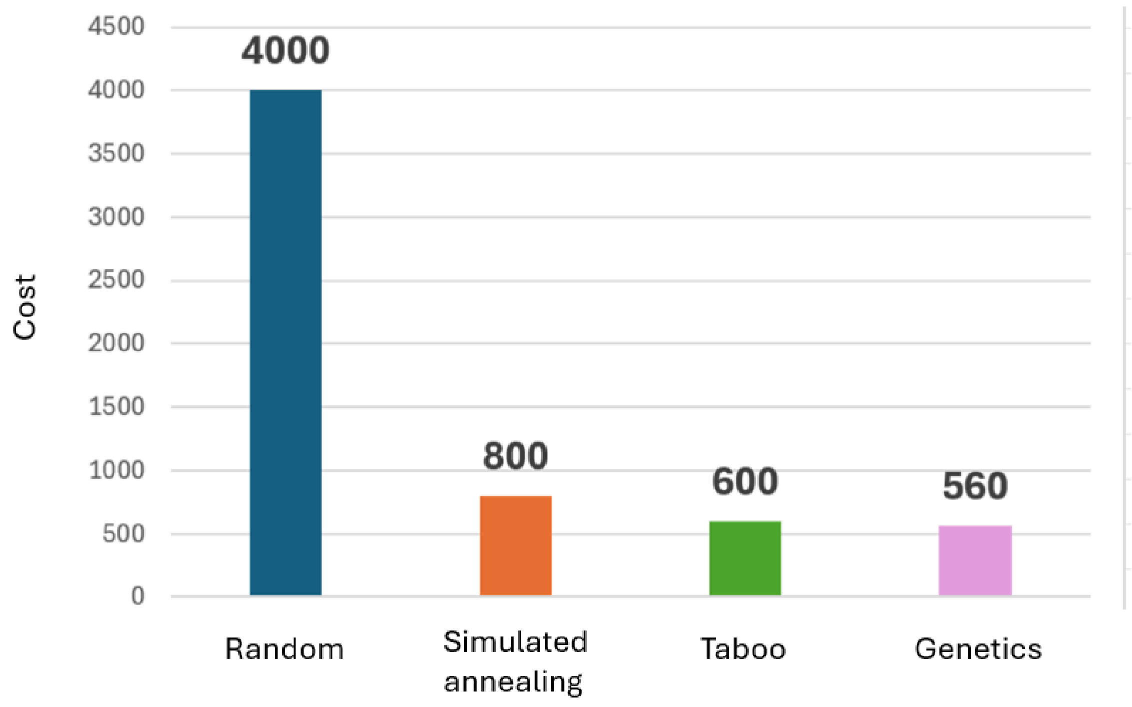


Figure 37 : Comparison of the meta-heuristics results

III. Collaborative Optimisation : SMA+Metaheuristics

III.1. SMA Ami Protocol:

II.2.1. Specific SMA Ami algorithm

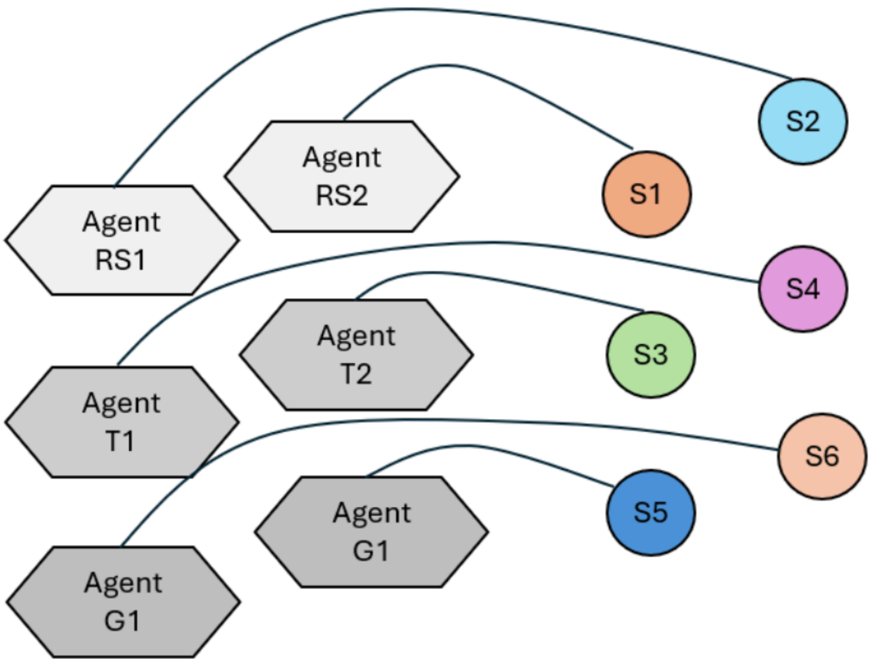
Principals:

The Ami protocol is based on the sharing of solutions and solution scores between agents through a Solution Pool. This Solution Pool contains progressively better solutions.

First, three agent classes are created, one for each meta-heuristic: a simulated annealing agent class, a taboo agent class and a simulated annealing agent class.

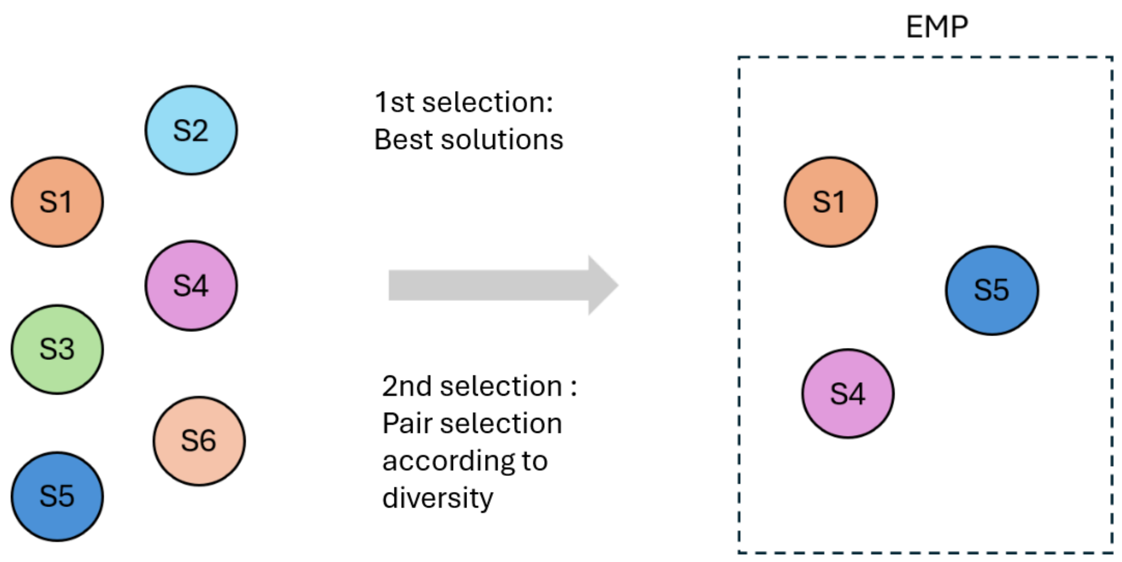
The steps in the optimization model are as follows:

1) Agent parameters are initialized: initial state, number of iterations and agent-specific parameters.
2) Each agent will then provide an optimized solution with its associated score. These solutions and scores will be stored in a "results" list.



3) The Pool of Solutions will then be formed. Not all solutions will be included in the Pool of Solutions. There are two selections before the Pool is integrated:
   - The first selection consists of choosing the best half of the solutions found.
   - The second selection consists of choosing the pairs of solutions with the lowest diversity. The diversity calculation between two solutions corresponds to the number of arcs they have in common. (Note: Tests have been carried out between the choice of pairs with the lowest and highest diversity. The stronger choice of diversity pair enables a wider exploration of the environment, while the weaker choice of diversity pair enables concentration on promising areas).

   The selected solutions are then integrated into the Pool of Solutions.



4) Finally, the simulated annealing and taboo agents will randomly select a solution from the Pool of Solutions. The genetic agent will take the solutions from the Pool to generate its initial population and will complete it with randomly generated solutions.
5) Steps 2,3,4 are iterated for a given number of steps.

II.2.2.1. Results

The evolution of the best solution found by the agents corresponds to the graph below, using a dozen agents for 5 iterations for the big data base :

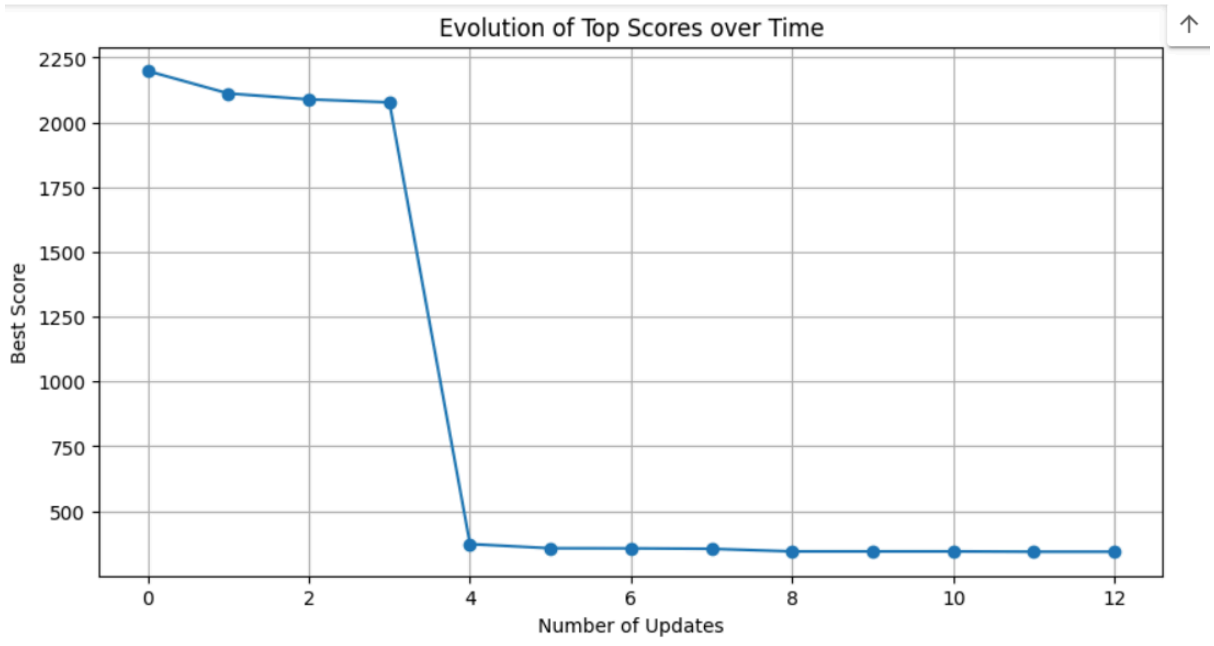


figure 38 : Evolution of best score over time

II.3.3.2. Analysis of the results and interest of the study

We obtain a new best solution around 380. This improves the meta-heuristic result by 180 points. The development of a friendly protocol to improve truck scheduling is interesting.

III.2. SMA Enemy protocol

III.2.1 Enemy protocol

In the enemy protocol, metaheuristic agents act as rivals, sharing only the current best value of their optimization criterion, rather than their complete solutions. Each agent works independently to find a viable solution, without exchanging with others. When an agent discovers a criterion value that is better than the one it previously received, it sends a message to the sender to update its solution. If the value received is higher, the receiving agent adjusts its parameters to improve its own solution. Genetic, tabu and simulated annealing approaches explore the search space globally and locally, respectively. Therefore, it is not always wise to have them compete directly.

Our protocol is based on two rival teams, the first composed of a genetic agent and a taboo agent, and the second of a genetic agent and a simulated annealing agent. Agents from each team work together to outperform the opposing team. Each genetic agent on each team receives an initial population that it improves to provide an initial solution space for the other agent on its team. If one of the teams achieves a better criterion than the other, it shares this information with its opponent. If the team receiving this information finds that it is better, it adjusts its parameters to seek an even better solution.

To come up with this system we looked for research and publication in the field such as: Self-Reconfigurable Manufacturing Control based on Ontology-Driven Automation Agents, by Wilfried Lepuschitz
https://www.researchgate.net/figure/An-example-of-a-Multi-Agent-System-for-decision-support-6_fig1_277727183

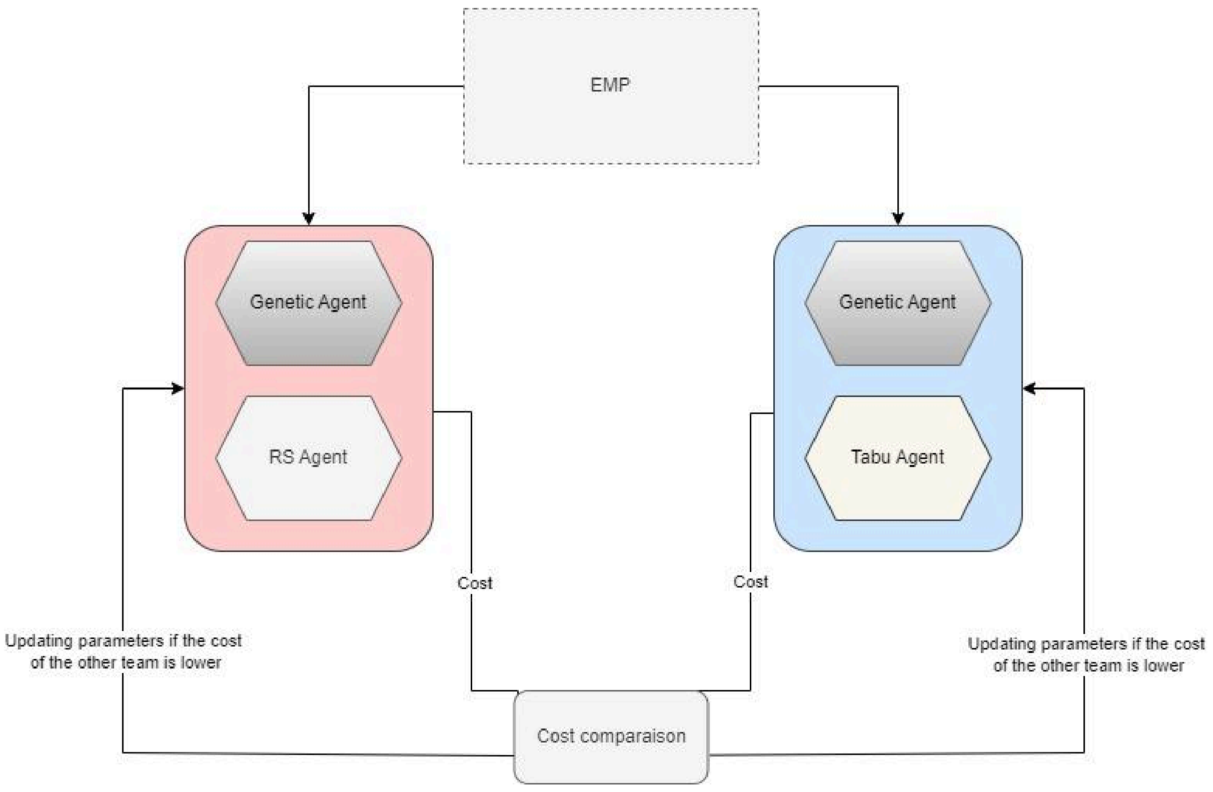

EMP
Genetic Agent
RS Agent
Genetic Agent
Tabu Agent
Cost
Cost
Updating parameters if the cost of the other team is lower
Updating parameters if the cost of the other team is lower
Cost comparaison

Figure 39 : Enemy collaboration scheme

To update the parameters of the two teams we follow this scheme: When one team learns that the other has discovered a solution superior to theirs, they adjust their parameters and restart their search. When one team receives information that it is outmatched by the other, it modifies its parameters and begins the search again as follows:

In our procedure we only update the parameters of the genetic and tabu agent, not the genetic ones as it becomes really costly in terms of execution times if we also update the genetic agents. So for the team with simulated annealing(RS), the parameters of simulated annealing are adjusted. We do this by updating the initial temperature and cooling rate that are increased by an amount proportional to the difference between the received criterion and the best criterion already found.
For the team combining the genetic algorithm and the tabu search, the number of neighbors of the tabu algorithm is increased by an amount proportional to the difference between the criterion received and the best criterion already found.

III.2.1 Analysis of the results

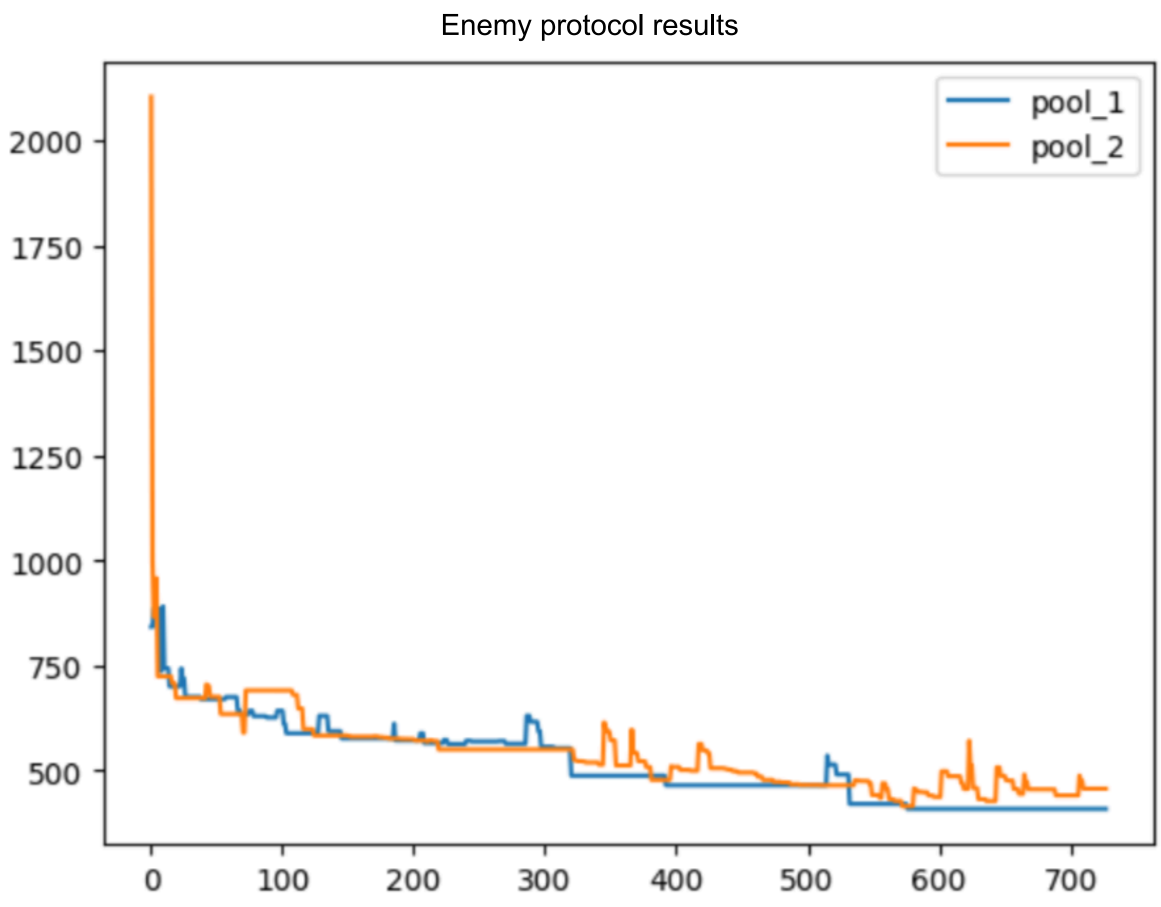


Figure 40 : Results of the enemy protocol

Observations :

We observe that the two pools of each competitor evolve and alternate, which is coherent with the functioning of the protocol, each team strives to find a solution superior to that of the other. After a certain number of iterations, the two teams converge towards an optimal solution where each can no longer make any significant improvement so we have chosen to stop the algorithm at a given step to help with the computational cost.

III.3. Comparison of the two protocols

After analyzing the results of each protocol we now have to compare them: we have done so by changing parameters and sizes of pools in each protocol. We executed the codes and ran them through multiple iterations to observe the results that gave us the following curve representing the evolution of the results according to the number of Clients taken into account.

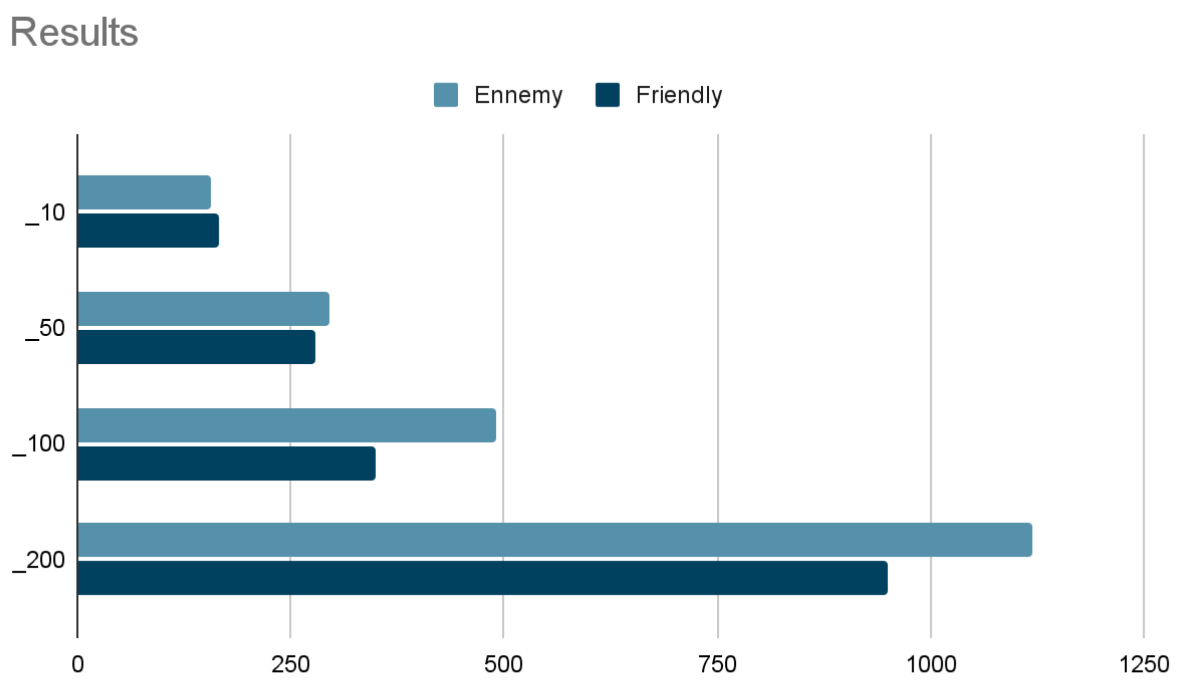


Figure 41 : Comparison of the results

Observations:

It is remarkable to see that for a small number of clients, both protocols succeed in identifying almost the same optimal solutions. However, as the number of clients increases, the "friends" protocol becomes significantly superior, whether in terms of solution quality or execution time. This superiority was anticipated, given that the “friends” protocol benefits from full collaboration between agents.

IV. Collaborative Optimisation : SMA + Metaheuristics + RL

I Principles

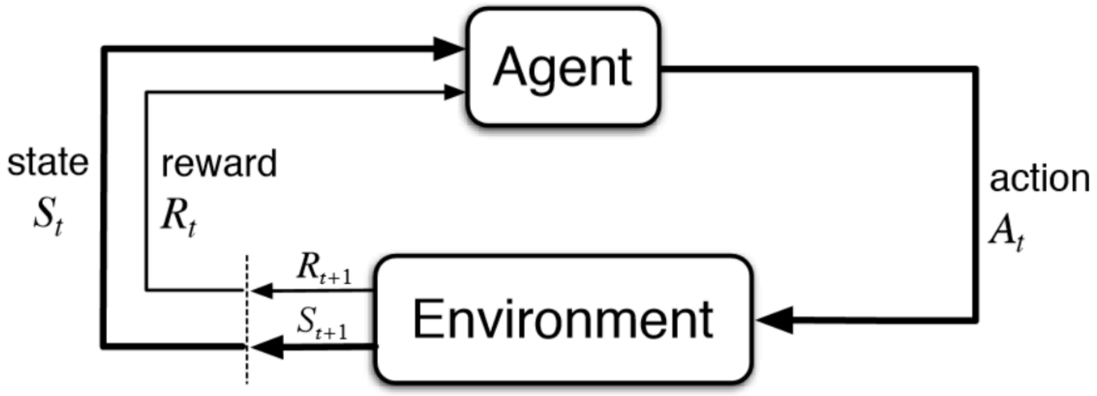
I-Reinforcement method with Q-learning

Reinforcement learning is an approach in artificial intelligence that enables a system to learn how to make optimal decisions by interacting with its environment. The objective is to maximize the total long-term reward (the gain in kilometers here) by making appropriate decisions. This involves modeling the environment as a series of possible states and actions, they are the neighbors of the current state and the changes in a road in our problem.

The goal of Q-learning is to learn a policy that maps states to actions in a way that maximizes the cumulative reward over time. Here, we must, however, minimize the number of kilometers but it is the same logic. This is achieved by estimating the quality of neighbors of the current state, known as the Q-value. The Q-value represents the expected cumulative reward of taking an action from a given state and following the optimal policy thereafter.

During the learning process, the agent explores the environment by taking actions and updating its Q-values based on the observed rewards. The more we improve the road, the more we have rewards.

The Q-values are iteratively updated which balances the immediate reward with the expected future rewards. We used the E-greedy in order to balance the algorithm. This update process gradually refines the agent's policy until it converges to the optimal policy that maximizes the total reward over time.



I.2 - SMA and reinforcement

In our SMA program, we have 3 types of agents. Two of them have to choose between multiple neighbors in order to improve the current state : the Tabou algorithm and the SR algorithm.
From the outset, these algorithms took random neighbors to improve scheduling. This method is simple but not very efficient. The idea here is not to modify the way the agents interact with each other, but to improve the 2 agents' ability to find neighbors likely to be promoters. This is the main goal of the reinforcement method in our algorithm.
To do so, we used the Q Learning approach with the E-greedy indicator.

I.3 - creating neighbors

The best way to find better neighbors in those 2 algorithms is to create more neighbors and very diverse ones.
Previously, only a few neighbors were managed between 1 and 4 generation methods.
We implemented 8 ways of creating neighbors :
-**IntraRouteSwap**: Swap randomly selected elements within a single route to explore alternative solutions within the same route.
-**InterRouteSwap**: Swap randomly selected elements between two different routes to explore alternative solutions between different routes.
-**IntraRouteShift**: Shift randomly selected elements within a single route to explore alternative positions within the same route.
-**InterRouteShift**: Transfer randomly selected elements between two different routes to explore alternative positions between different routes.
-**TwoIntraRouteSwap**: Swap two pairs of randomly selected elements within a single route to further explore alternative solutions within the same route.
-**TwoIntraRouteShift**: Shift two pairs of randomly selected elements within a single route to further explore alternative positions within the same route.
-**Eliminateshortestroad**: Eliminate the shortest non-empty route and redistribute its elements among the remaining routes to potentially improve overall efficiency.
-**Eliminaterandomroad**: Eliminate a randomly selected route and redistribute its elements among the remaining routes to explore alternative configurations.

All of these methods are used 100 times or more at each call of the SR or the Tabou algorithm in order to find very good neighbors and also very bad, which is important if we want to explore more. The strength of this method is that we can also improve the quality of exploitation because we can find better solutions quickly. We can also improve the quality of exploration by choosing a very bad neighbor.

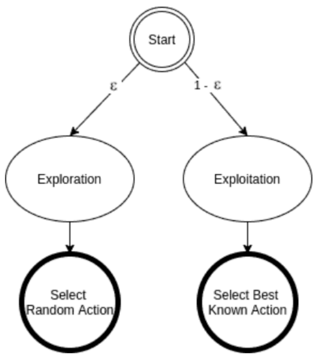
I.4 - Q-learning choice

Now that we have many and diverse neighbors, we have to find a way to evaluate them.
We used the Q Learning method. Our ultimate goal is to find the shortest course. We decided then to minimize the Q value based on the calculation of the length of the current state and the length of the neighbor.

This is the formula for the Q value : $Q(s,a) \leftarrow Q(s,a) - \alpha(r + \gamma \min a' Q(s',a') - Q(s,a))$

The Q value of a solution is his length.
α is the learning rate, γ the learning factor and r the unitary cost for each action.
The more the Q value is low, the better the solution.

I.5 - Choice of the next current state



In order to balance the algorithm, we decided to add à E-greedy criteria in the choice of the next current state. It is very important to balance the algorithm because there are many local optimums and the algorithm can converge too quickly.
To do so, we can allow the function “neighbors_reinforcment” that gives neighbors to the SL and the Tabou algorithm that are worse than the current state in order to escape from the local minimum.

Here is how this function works :

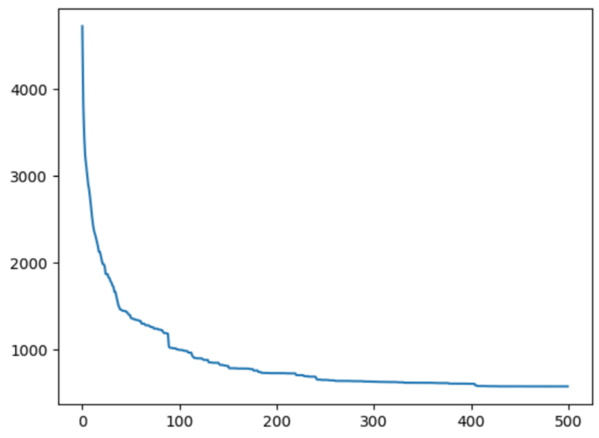
```
def reinforcment_neighbors(road)
    i←E-greedy(e)
    if i negative                                  #we chose exploitation
        L←explore_neighbors(road)          #we create neighbors and put them in a
list

        road←best neighbor in L      #The best is the one who have better Q
        update Qvalue
    end if
    else:                                       # we choose exploration
        road←L[i]                      # i chosen by the E-greedy function
        update Qvalue
    end else



def E-greedy(e):
    p ←random number between 0 and 1
    If e<p:                                         # we choose the exporation
        return random number between 0 and 7      #because there are 8
neighbors
    else :                                          # we choose the exploitation
        return negative number
```

Then we just have to replace the current neighbor strategy (which uses only one of the 8 strategies) by the reinforcement neighbors in the SR and Tabou algorithm in order to give them the reinforcement aptitude.

II Results for Tabou and RS

We found an improvement for the Tabou algorithm with an E-greedy value of 0.2 and below. We can improve the solution from 30 km on average.

With Q learning : Best score : 579 Without Q learning : best score 612



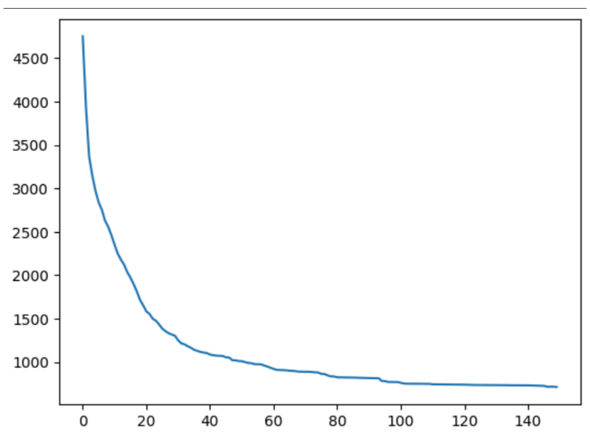


figure 42 : influence of Q learning

This is not always the case for the RS algorithm. It is because the exploration logic is already very present in the algorithm. Even with a very small value of E-greedy, it is very difficult to have better results than without reinforcement. We don't exploit enough.

III Results for SMA and reinforcement

Because it is very long to run many steps on the SMA algorithm with reinforcement, we only put two steps in order to find the best E-greedy value, the best balance between exploitation and exploration.

There are some tests with different values of E-greedy. We can see the improvement of the solution over time. The abscissa represents the number of updates and ordonates the number of kilometers of the solution.(When E-greedy is low, we push more exploitation than exploration.)

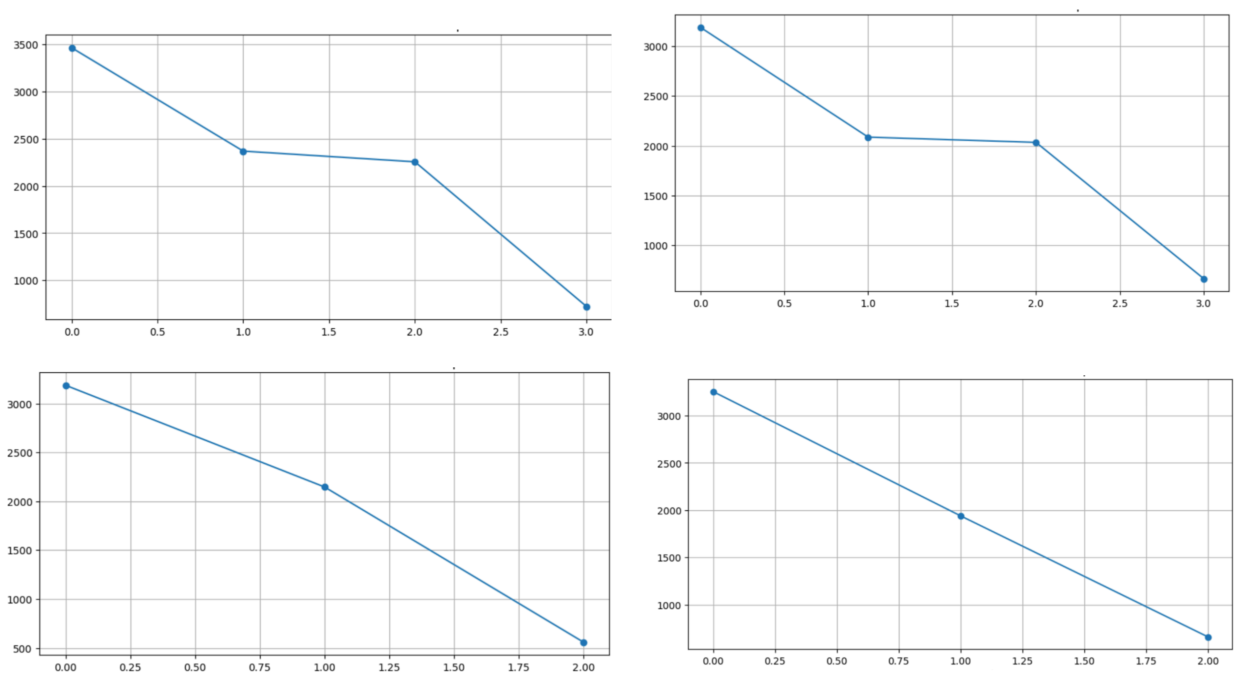


figure 43 : Cost evolution according to E-Greedy value

	E-greedy = 1	E-greedy = 0.7	E-greedy = 0.2	E-greedy = 0.02
Best score(km)	560	546	530	379

We find that the value 0.02 was pretty effective but we can also lower this value and keep good results. The exploration has to be very low in order to exploit enough but we didn't expect how much it would be.

Our long test with many steps has been able to find a solution with only 350 km. It is a small improvement compared to the SMA without Q learning.

V. Comparison tables and global analyses of collaborative optimization performance

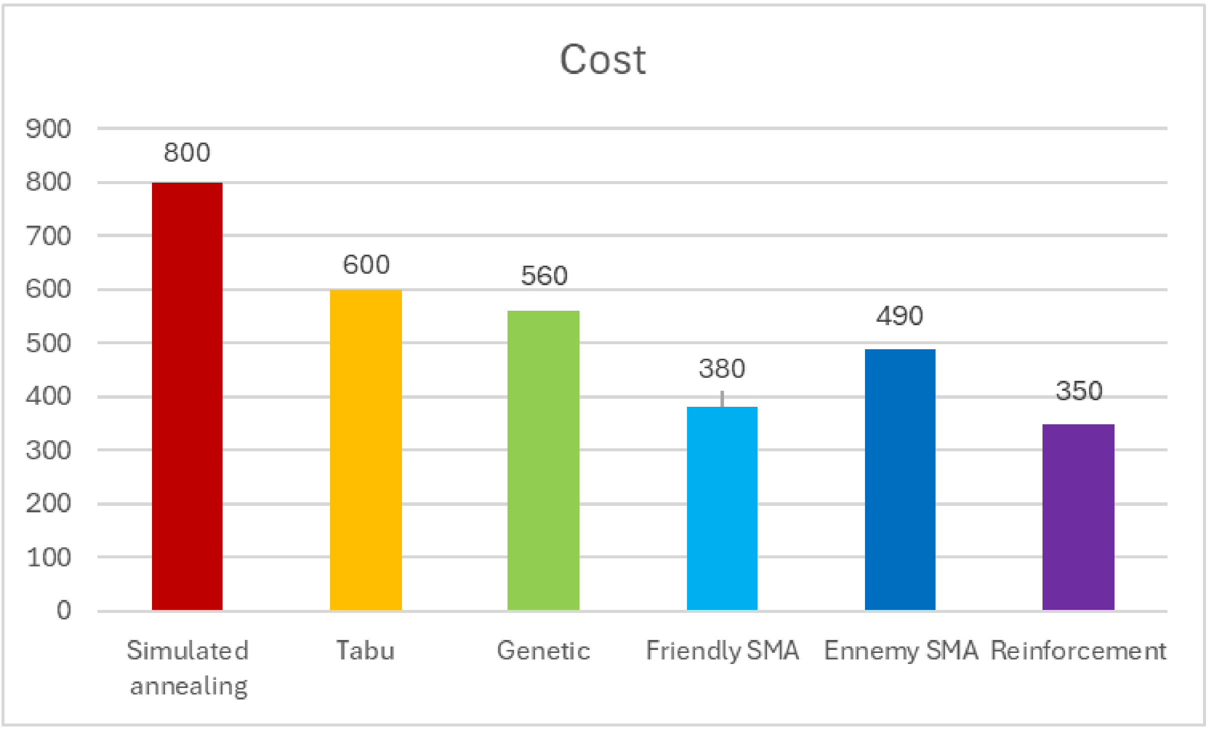


figure 44 : Final results of each algorithms

Conclusion and outlook

This study has enabled us to optimize truck scheduling. For the first route, we went from an average cost of 4,000 for a random solution to a cost of 560 for the metaheuristic part. The SMA part further improved this score to a cost of 380. Finally, reinforcement enabled a final improvement, with a cost of 350.

It could be relevant to have two different E-greedy values for Tabou and RS. A low and good value for Tabou is always too high for RS because this algorithm can already explore a lot.

We could also implement Deep Q learning. Instead of having 8 neighbors, we can create thousands of neighbors. This could be very effective but also very demanding in terms of power and storage.