Fil Rouge ICO - Groupe ICOnique

Yishan Sun, Simon Kurney, Pablo Aldana, Cédric Jung, Baptiste Deconihout, Zoé Poupardin

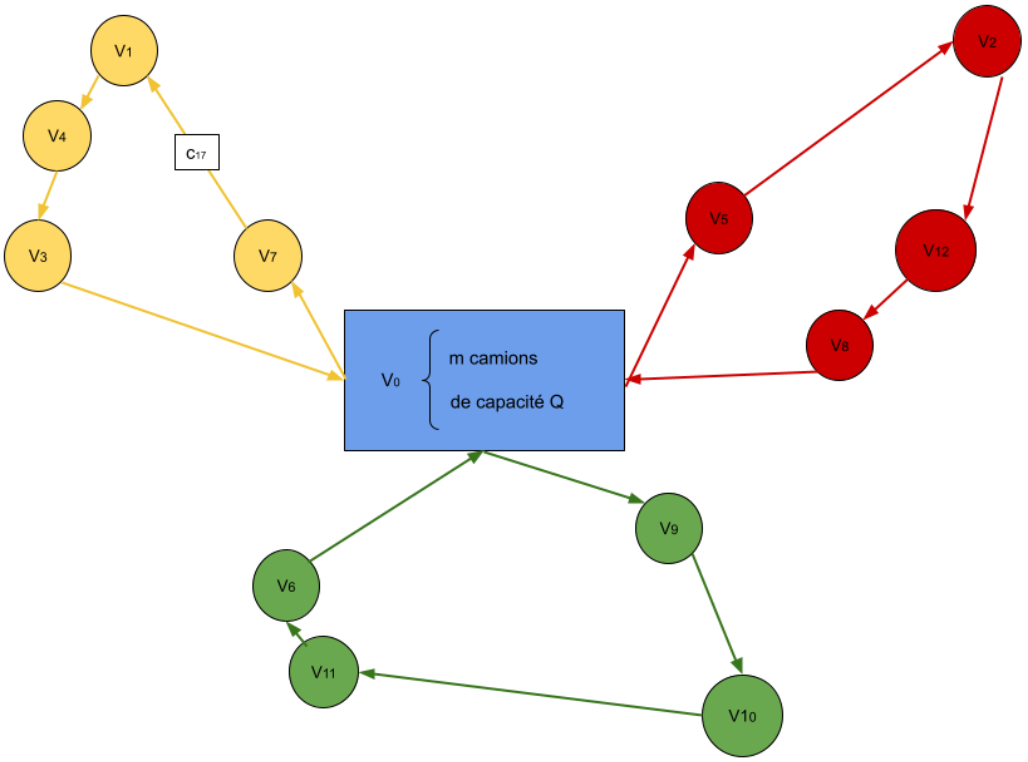
Introduction

I. Les problèmes de PTVFT

I.1. Description

Le Vehicle Routing Problem (VRP), en français problème de tournée de véhicule est un problème algorithmique d'optimisation NP-complet. Il s'agit d'une analogie généralisée du célèbre problème du voyageur de commerce. L'objectif du VRP est de déterminer l'ensemble des routes, recouvrant la totalité des nœuds clients à livrer, minimisant le coût total de la tournée , tout en respectant les contrainte de capacité des véhicules (le coût prend en compte le nombre de camions affrétés à la tourner, ainsi que le nombre de routes empruntées).

Plus concrètement, on peut considérer le problème en dessinant un graphe complet comportant (N + 1) nœuds ; un nœud central V_0 représente le dépôt où stationnent K véhicules identiques de capacité Q. Les autres nœuds V_i, i ∈ {1,2,…,N} représentent les clients i caractérisés par une demande de produit q_i, et attendant leur colis dans un intervalle de temps . Les arcs (i, j) représentent la possibilité d'un trajet direct du client i au client j avec un coût de transport de C_{ij}.



I.2. Complexité

Comme pour l'ensemble des problèmes NP-complet, obtenir de manière optimale une solution complète est quasi impossible sur un grand nombre de données. L'approche de résolution d'un tel problème est de trouver une solution qui soit la plus proche possible de la solution optimale. Pour parvenir à une telle solution, on fait appel à des méthodes approchées, par exemple des métaheuristiques.

Le VRP étant un problème d'optimisation combinatoire, la fonction de calcul du coût d'une solution prend de nombreux paramètres en considération. Nous avons décidé d'introduire progressivement les différents paramètres aux calculs du coût.
Nous avons pris en compte les paramètre dans l'ordre suivant :

1. Nombres de camions et coût des routes
2. Volumes des camions
3. Temps et fenêtre de livraison

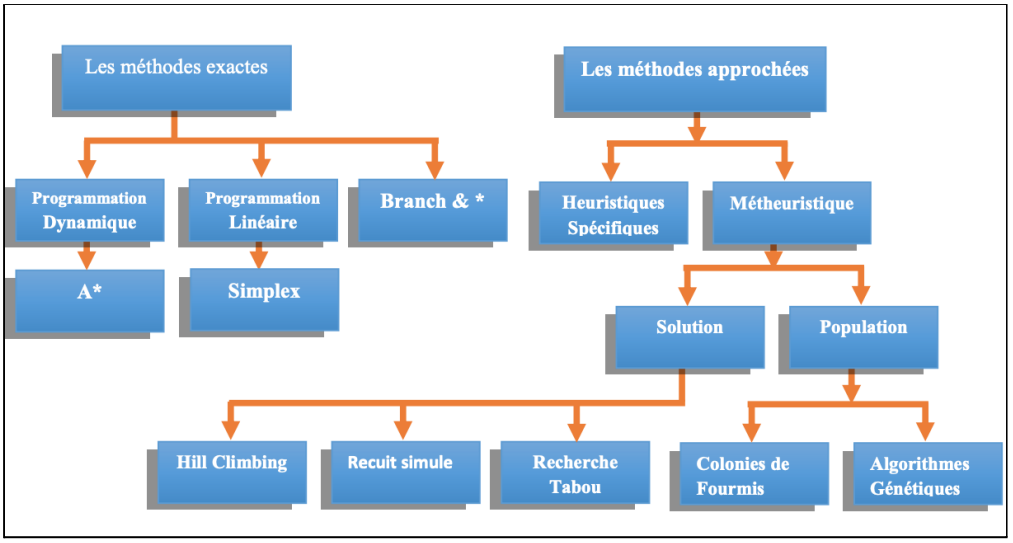
Dans cette première partie de projet, nous allons programmer puis confronter trois métaheuristiques :

1. Méthode de recherche tabou
2. Recuit simulé
3. Algorithmes génétiques : méthode d'optimisation stochastique basés sur le mécanisme de la sélection naturelle

I.3. État de l'art

On peut résumer la plupart des méthodes d'optimisations existantes en deux grandes familles de méthodes :

- Les méthodes exactes
- Les méthodes approchées



Nous ne nous attarderons pas sur les méthodes exactes énumérées dans ce graphe puisqu'elles ne sont pas présentes dans notre projet.
Les méthodes approchées aussi appelées méthode heuristiques sont des méthodes qui ont pour objectif de trouver une ou des solutions "optimales" et réalisable dans un temps raisonnables à des problèmes complexes comme celui du *VRP*
Les métaheuristiques sont des méthodes inspirées de phénomènes naturels, elles ont été adaptées à la résolution des problèmes et plus particulièrement aux problèmes d'optimisation, qui visent d'atteindre un optimum global parmi de nombreux optima locaux
Nous allons en étudier trois d'entre elles comme dit précédemment mais il est intéressant de savoir qu'il en existe d'autres, avec parfois mêmes des approches différentes du problème.

II. Optimisation par Métaheuristiques

II.1. Tabou

II.1.1. Algorithmes spécifique PTVFT

On retrouve l'algorithme dans le fichier *taboue.ipynb*.

L'objectif est d'obtenir le meilleur ordonnancement pour desservir tous les clients dans le temps imparti. Un ordonnancement est meilleur qu'un autre si son coût total est plus faible que ce dernier.

Nous abordons ce problème en considérant une liste de camions, chacun défini par une liste de chiffres correspondant aux clients qui vont être livrés. Au début de l'algorithme, nous commençons avec une solution que nous cherchons à améliorer en modifiant l'ordre des clients livrés et le nombre de camions. Pour cela, nous avons recours à différentes fonctions de voisinage (exchange, inverse, exchange_inside) permettant d'inter-changer cet ordre.
Dans un premier temps, il est nécessaire de savoir si les listes trouvées peuvent correspondre à des solutions. Il s'agit du rôle de la fonction **acceptable** qui vérifie si le mouvement pour obtenir la liste n'a pas déjà été fait (appartenance à la liste tabou) et si la liste est meilleure que la précédente selon le critère d'aspiration.
Dans un second temps, le critère pour déterminer si la solution obtenue est meilleure que la celle déterminée jusqu'à présent est le critère du coût, obtenu par la fonction **coût,** est calculé selon le trajet parcouru par l'ensemble des camions.

Les fonctions de voisinage permettent la création de nouvelles solutions. Elles se basent sur des opérations élémentaires telles que des inversions d'ordre de livraisons de clients, de nouveaux camions en divisant des camions préexistants, des fusions de camions et enfin des créations ou des suppressions de camions.

Ainsi, la fonction **exchange** permet de modifier l'ordre de livraison de clients livrés en ne modifiant pas le nombre de clients par camion. Par exemple, la solution [[4,2,5],[1,3]] peut

devenir [[3,2,5],[4,1]]. De la même façon, la fonction **relocate** permet d'attribuer les clients à d'autres camions (existant ou non) sans tenir compte du nombre de clients par camion, en constituant des paires de clients. Enfin, la fonction **exchange_inside** permet la modification de l'ordre de livraison des clients au sein d'un même camion. Dans les trois cas, les fonctions vérifient si les solutions proposées sont acceptables (en appelant la fonction du même nom).

La structure de l'algorithme tabou est le suivant :

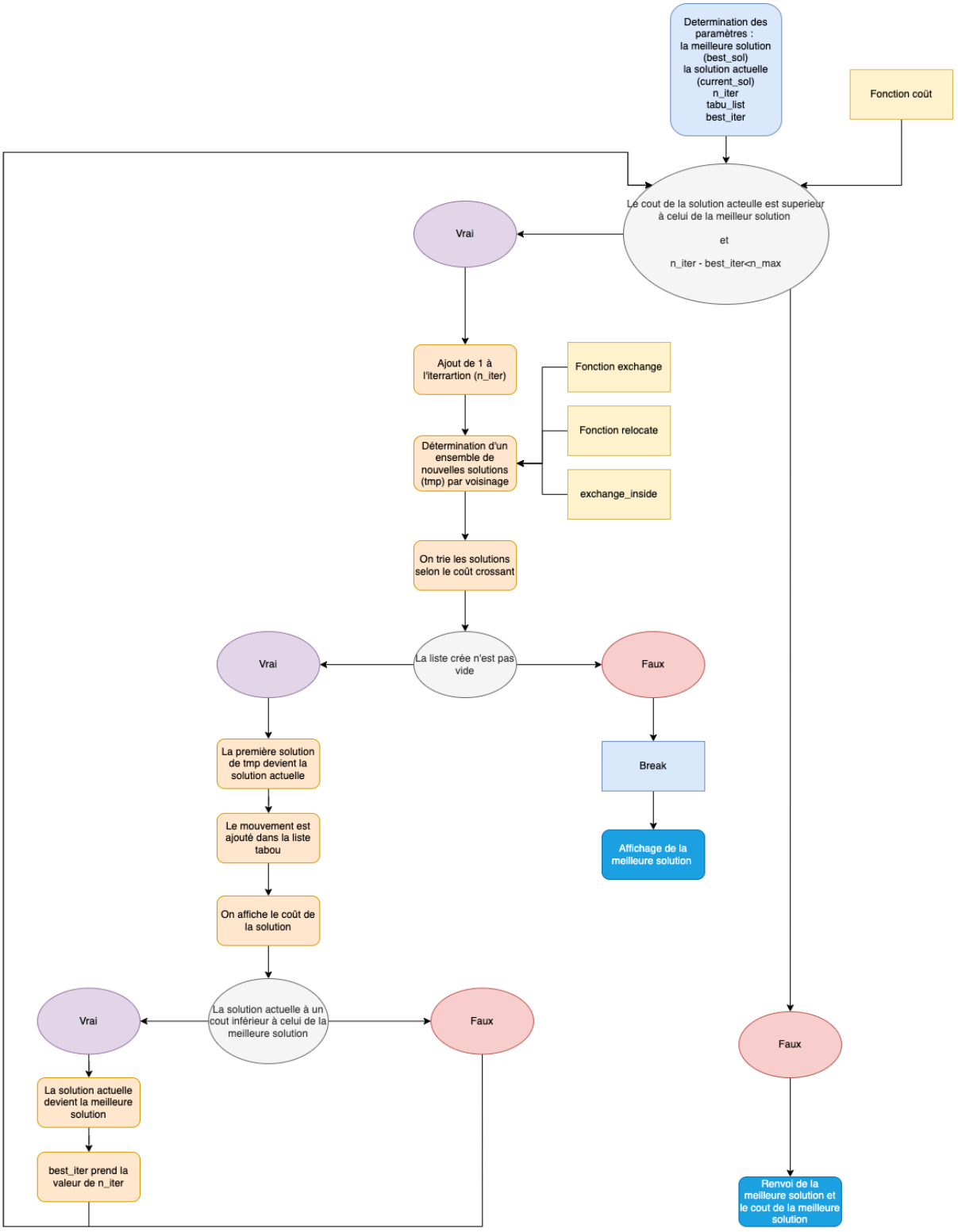
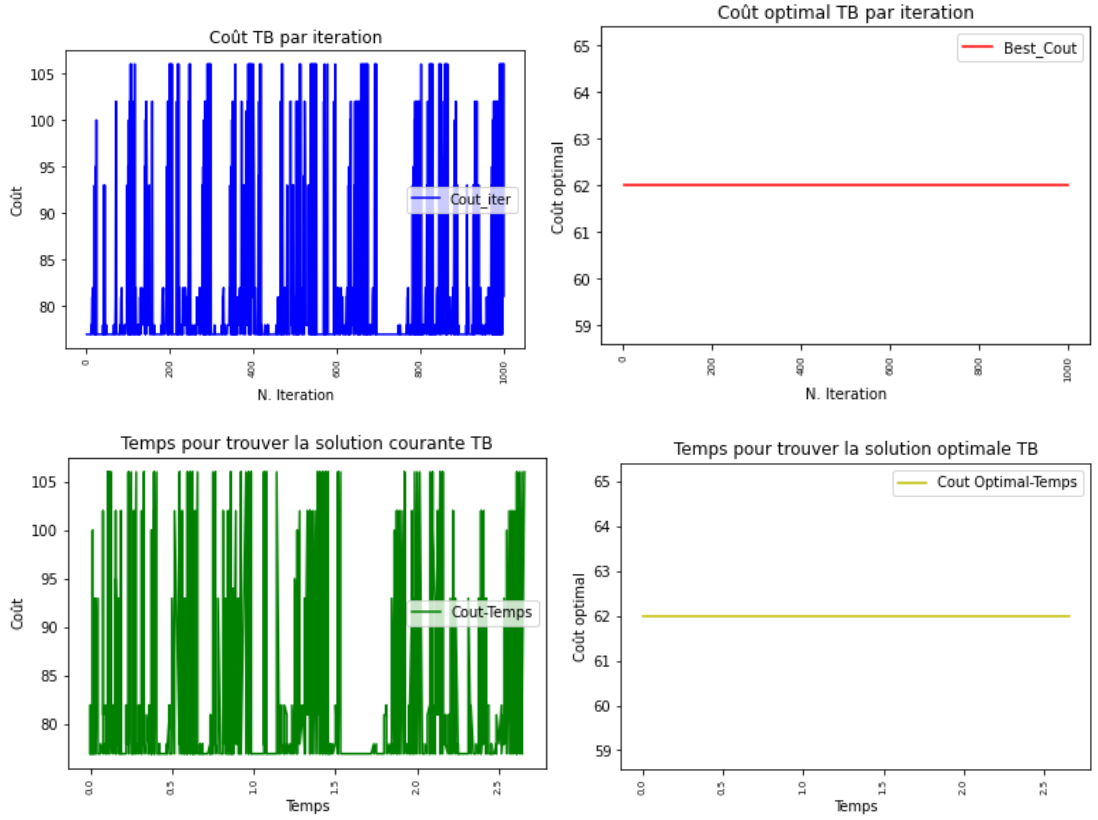


Figure 1 : Schéma du fonctionnement général de l'algorithme Tabou

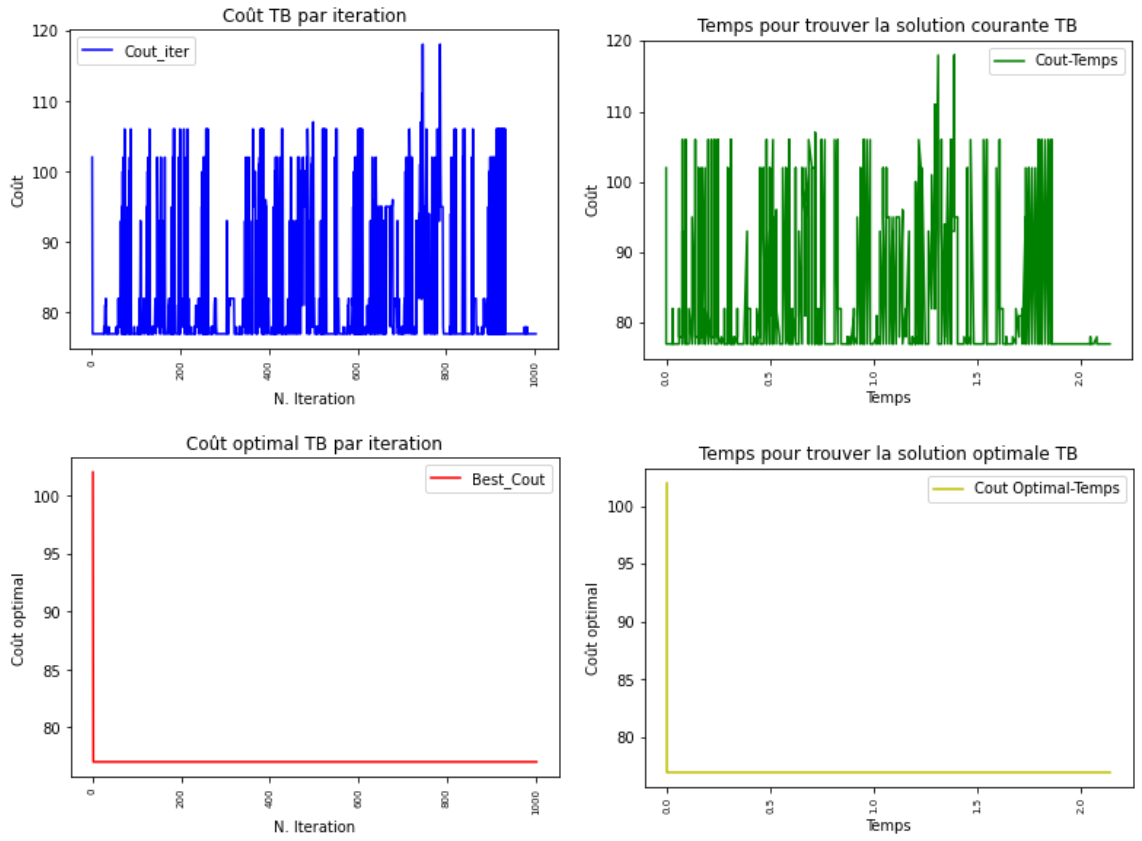
II.1.2. BDD de petite et de grande tailles

II.1.2.1. Les courbes et tableaux

On a dessiné des graphiques pour afficher la variation de coût actuel et de coût optimal en fonction du nombre d'itération et de temps.
On est parti d'abord de notre exemple simple avec 5 clients notés [1, 2, 3, 4, 5] avec un poids ω=5 pour le calcul du coût, un nombre d'itération maximum de 100 et un facteur d'aspiration de 100.
En partant de la solution initiale [[1, 2, 5], [4, 3]] on obtient les courbes suivante et la solution [[1, 2, 5], [4, 3]] d'un coût de 62



En partant de la solution initiale [[1, 2, 5], [4, 3]] on obtient les courbes suivante et la solution [[1, 5], [4], [2, 3], []] d'un coût de 77



II.1.2.2. Analyse des résultats

On retrouve cela dans le fichier *taboue-with-capacities.ipynb*.

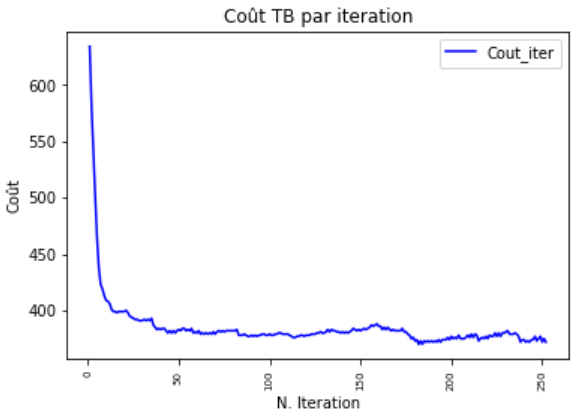
Nous avons décidé d'implémenter la notion de capacité dans une deuxième version de l'algorithme tabou, en implémentant une condition dans la recherche de voisinage. C'est-à-dire dans les fonctions, **relocate**, **exchange** et **exchange_inside**.

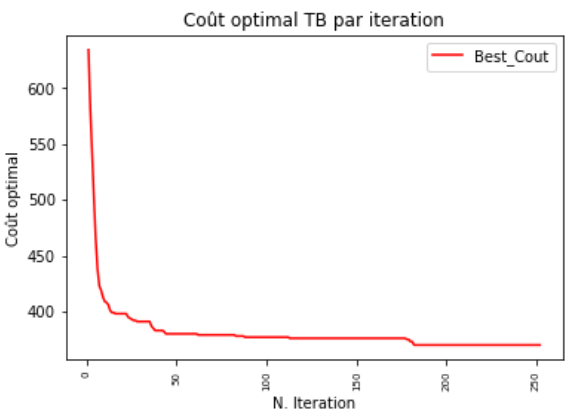
Nous avons implémenté la fonction **filter_on_capacities** qui permet de ne garder que les solutions qui répondent à une capacité maximale de chaque camion pour chaque route.

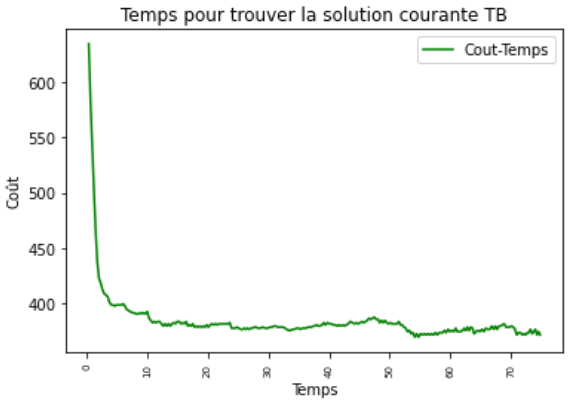
Dans le cas la fonction **relocate**, si la fonction **filter_on_capacities** retourne une liste de solution vide, on va tenter de rajouter une route à la solution donnée en paramètre à **relocate** et d'y déplacer un client afin de faire en sorte que d'avoir un voisinage. Cette façon de faire nécessite d'être amélioré.

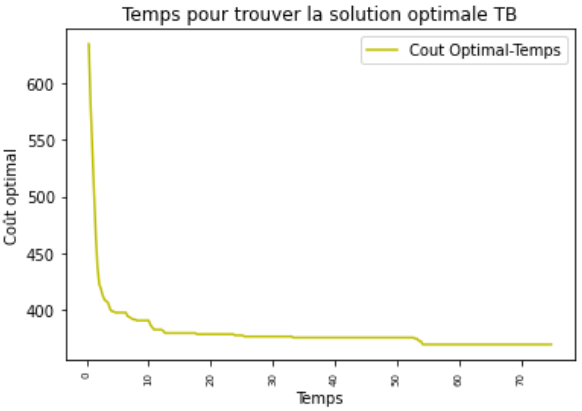
On est parti d'abord de notre exemple simple avec 51 clients noté avec un poids ω=5 pour le calcul du coût, un nombre d'itération maximum de 100 et un facteur d'aspiration de 3.

Avec exemple avec 51 clients, avec capacité maximale des camions de 100, et chaque client a une demande d'une taille 1, on obtient un coût de 370:

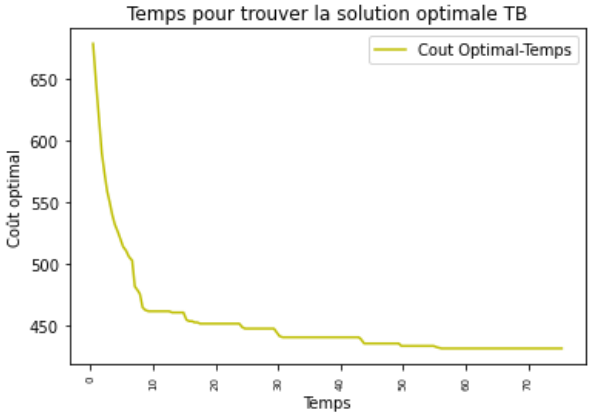


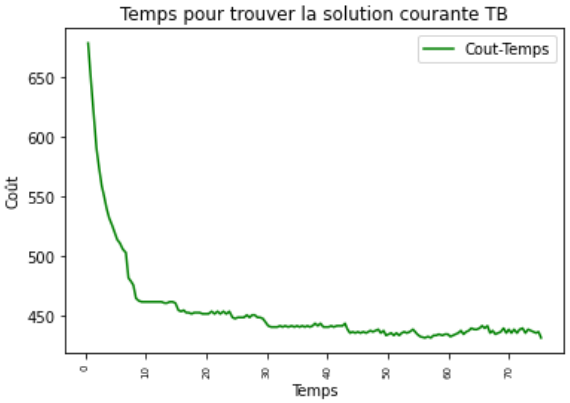


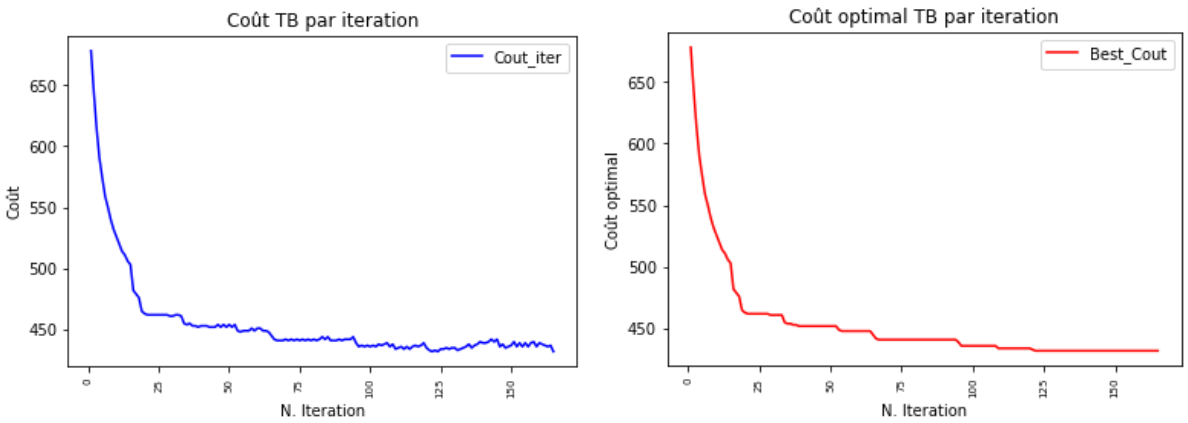




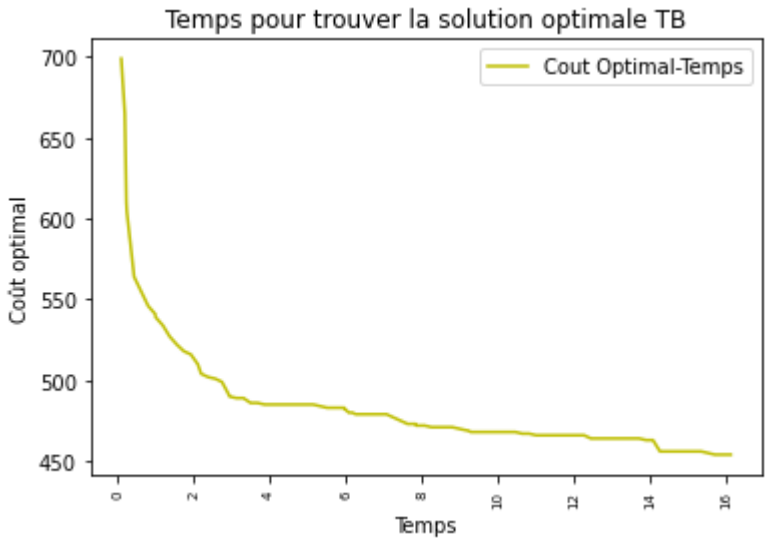
Avec exemple avec 51 clients, avec capacité maximale des camions de 100, et chaque client a une demande d'une taille 3, on obtient un coût de 432:







On peut enfin comparer l'apport du Q-Learning dans la recherche de la solution optimale. En effet, avec le Q Learning, la solution optimale est trouvée en quelques secondes (ici 16s) alors qu'il s'agit de l'ordre des minutes sans le Q learning.



II.1.2.3. Intérêt de l'étude

II.2. Recuit Simulé

II.2.1. Algorithmes spécifique PTVFT

On retrouve l'algorithme dans le fichier *rs.ipynb.*

Une seconde approche peut être celle du recuit simulé. Bien que cette méthode se base elle aussi sur la modification d'une solution acceptable par l'utilisation de fonctions de voisinage (décrites ci-dessus), celle-ci diffère en acceptant de garder une solution qui n'est pas meilleure que la précédente en supposant que cette dernière correspond à maximum locale. Ainsi, cette méthode fait appel à une probabilité liée au temps de recherche de la solution (voir figure 2).

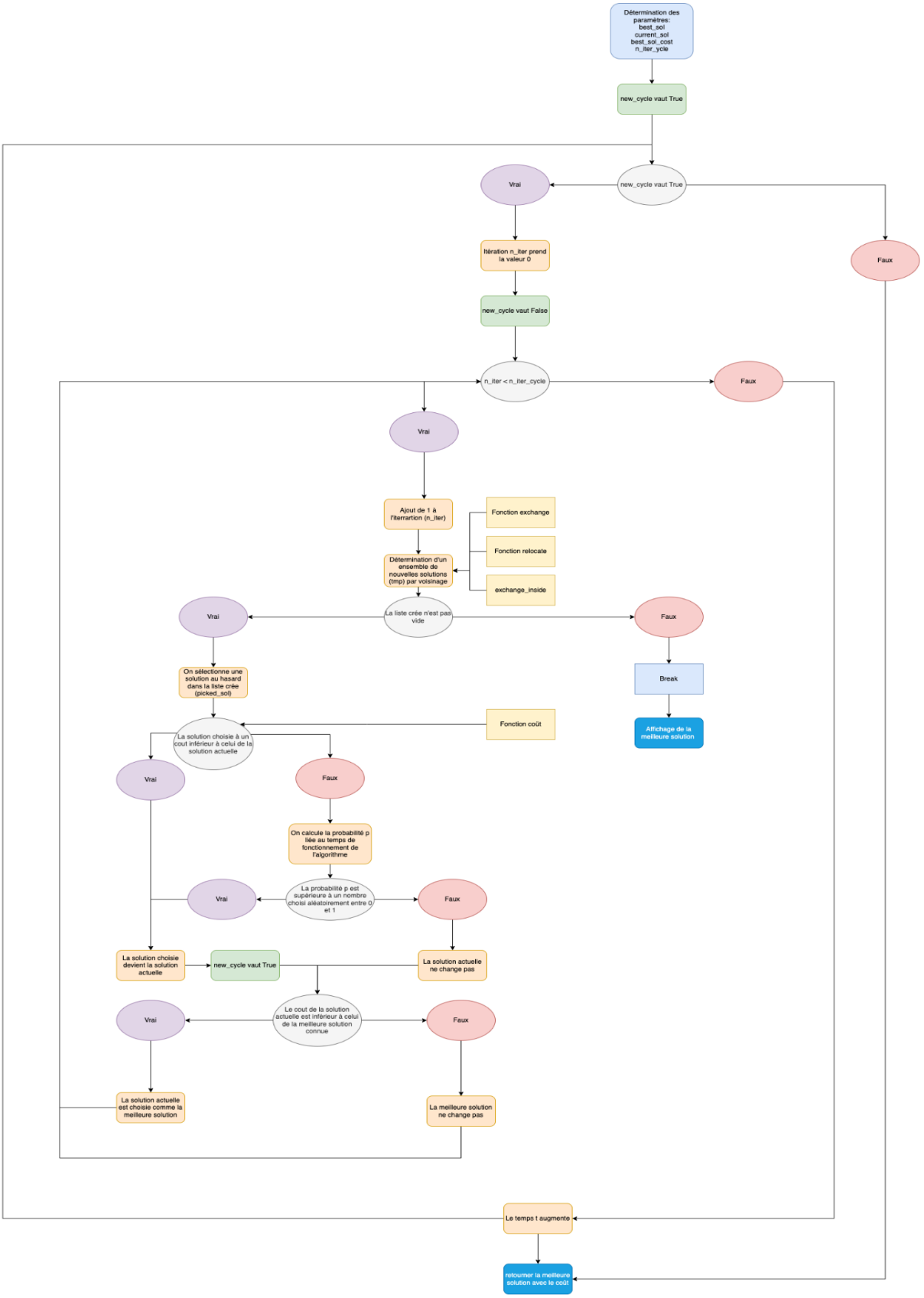


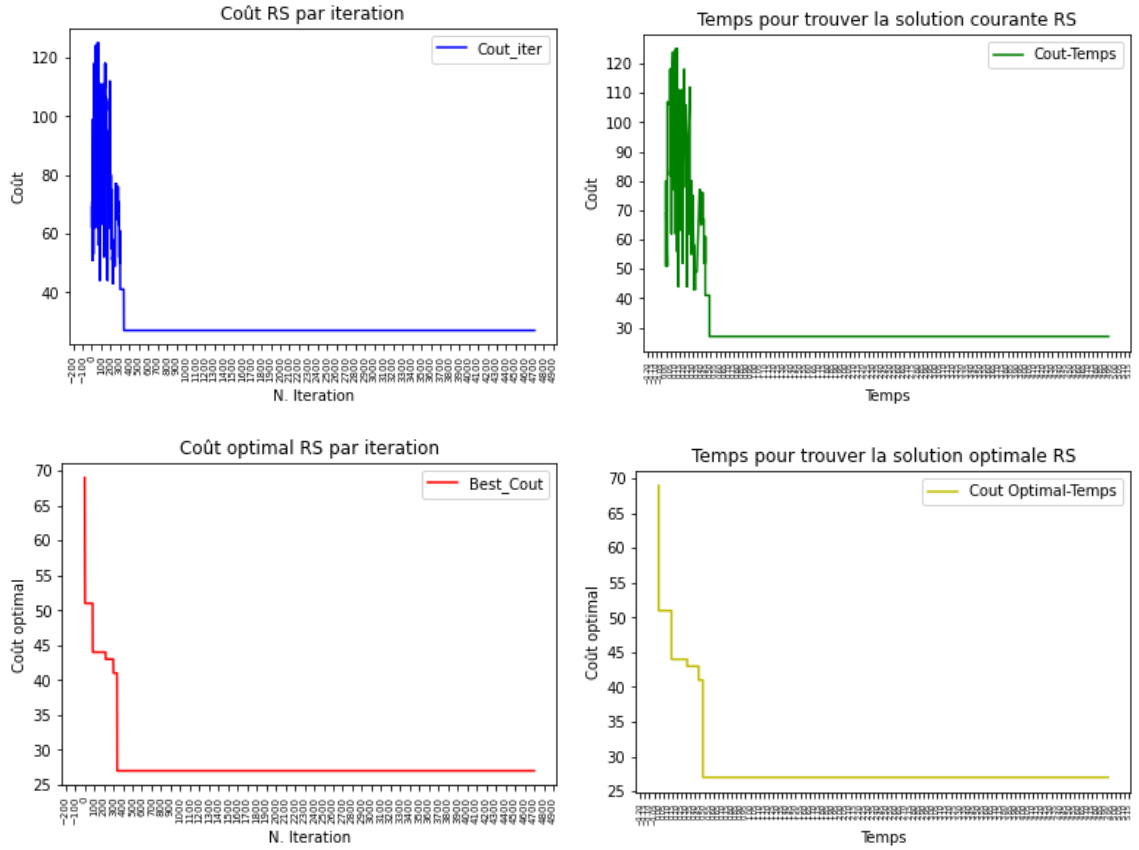
Figure 2 : Schéma du fonctionnement général de l'algorithme recuit simulé

On retrouve l'algorithme avec le même raffinement que celui présenté en II)c) dans le fichier *rs-with-capacities.ipynb*.

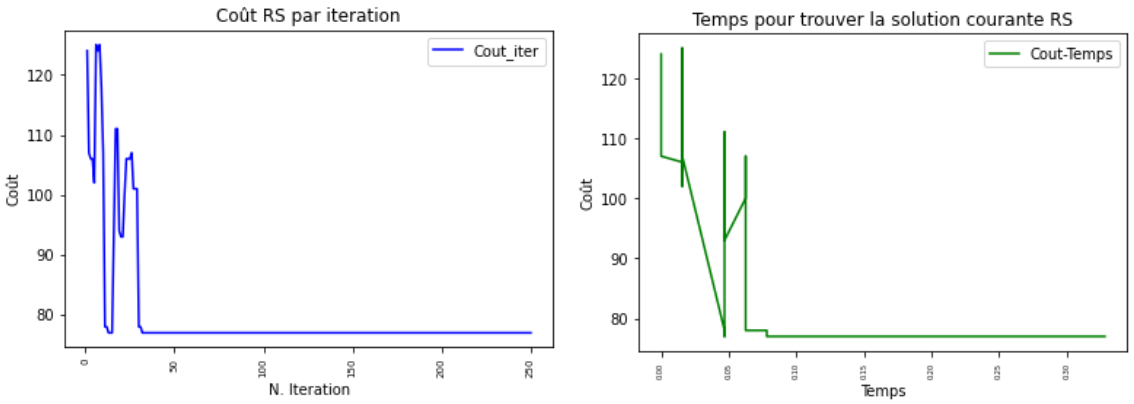
II.2.2. BDD de petite et de grande tailles

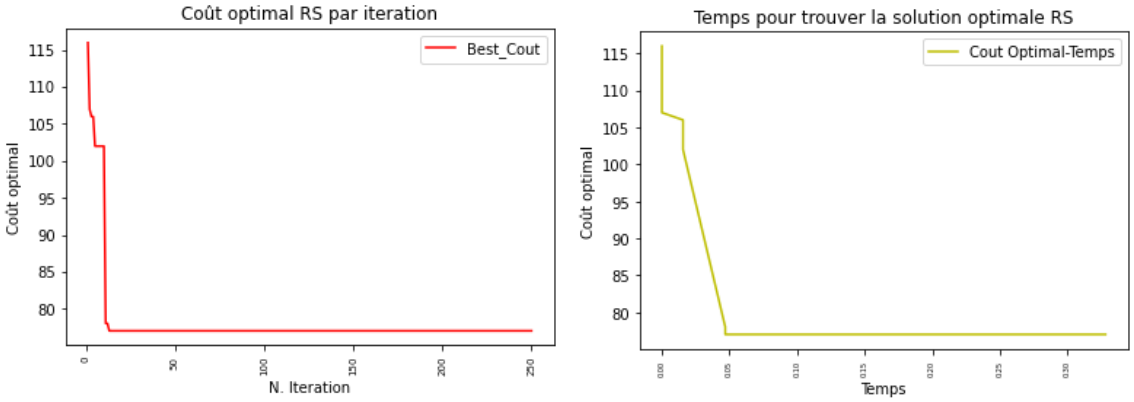
II.2.2.1. Les courbes et tableaux

On considère toujours un poids ω=5 pour le calcul du coût.
A partir de même exemple que tabou, on a aussi dessiné les graphes dans ce cas. On peut constater qu'il y a déjà beaucoup d'itérations.

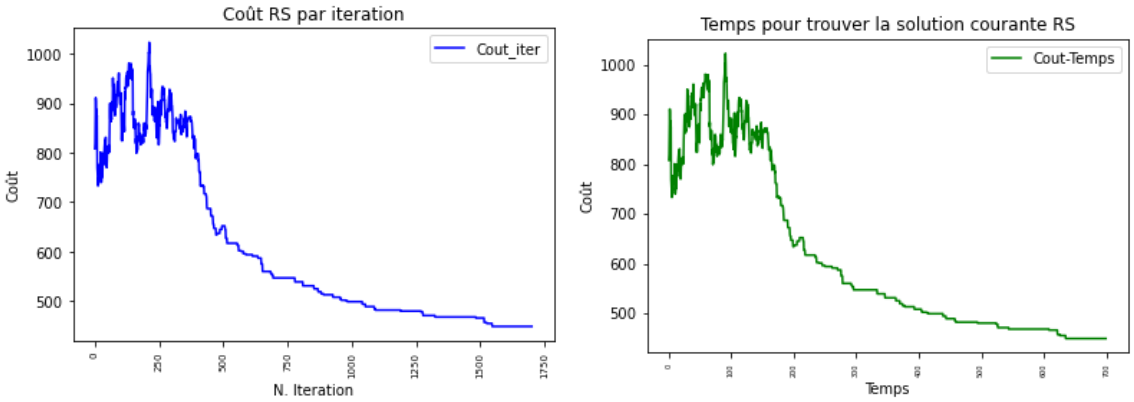


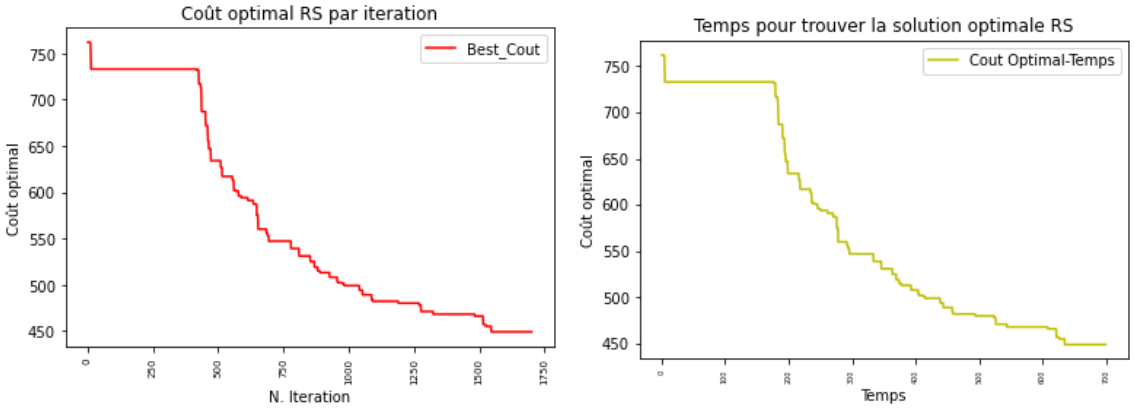
Après avoir ajouté la notion de capacité et avoir modifié nombre de l'itération, on a obtenu les simulations :





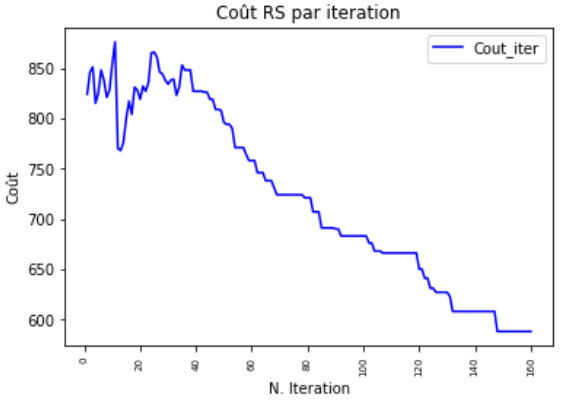
Avec le même exemple complexe, on a obtenu les simulations suivantes. Nous avons été forcé d'interrompre le calcul car trop d' itérations, l'algorithme a pris 11 minutes 45 secondes pour trouver la solution.

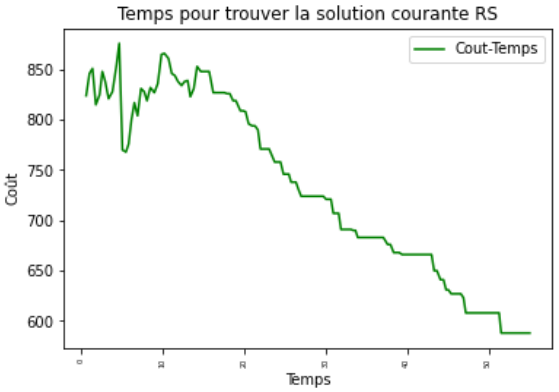


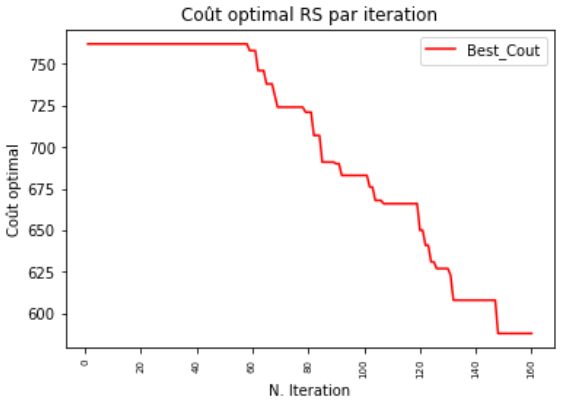


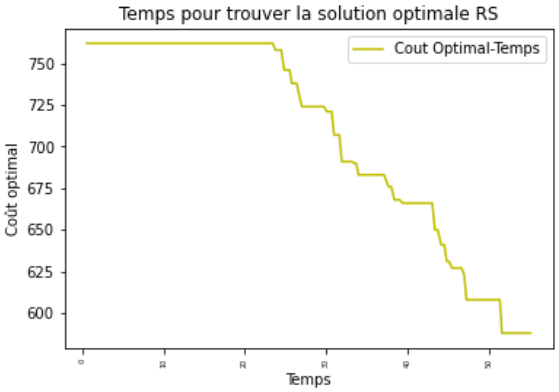
Après avoir ajouté la notion de capacité et avoir modifié nombre de l'itération, on a obtenu les simulations :

Avec la capacité 51 clients ayant chacun une demande de volume 1:

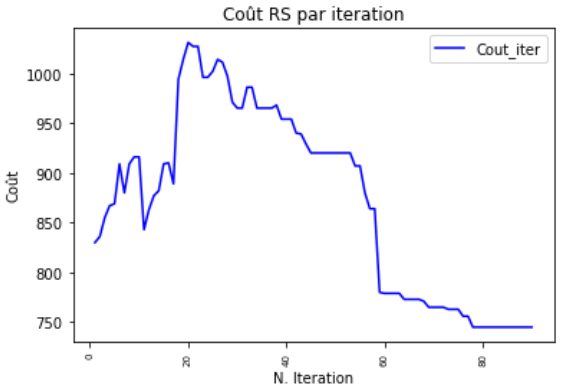


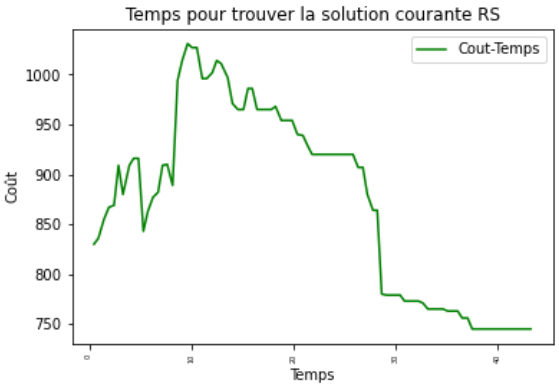


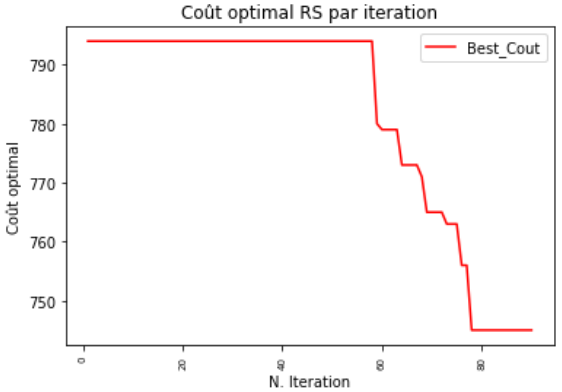


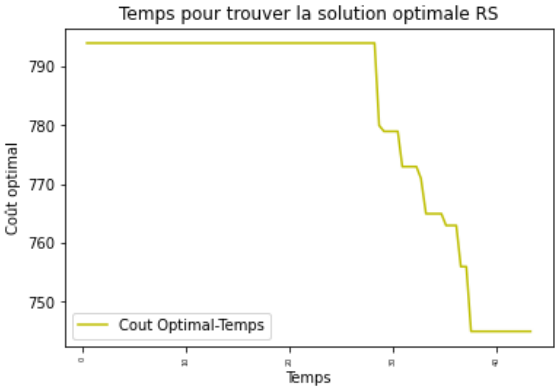


Avec la capacité 51 clients ayant chacun une demande de volume 3:



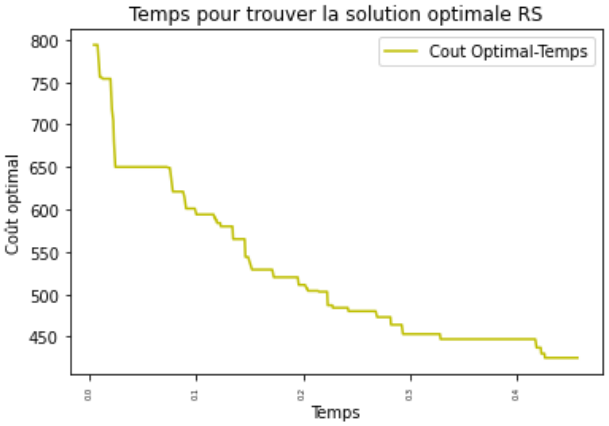






II.2.2.2. Analyse des résultats

On cherche ici à comparer les résultats obtenus avec ceux obtenus grâce au Q learning. On observe que pour 3 de capacité et une capacité maximale de 100, le temps pour obtenir la solution optimale est de moins d'une seconde (ici moins de 0,5s) alors que sans le Q-learning l'ordre de grandeur est celui des minutes. On en conclut que le Q learning est plus rapide pour la recherche de la solution optimale.



II.2.2.3. Intérêt de l'étude

II.1. Algorithmes génétiques

II.1.1. Algorithmes spécifique PTVFT

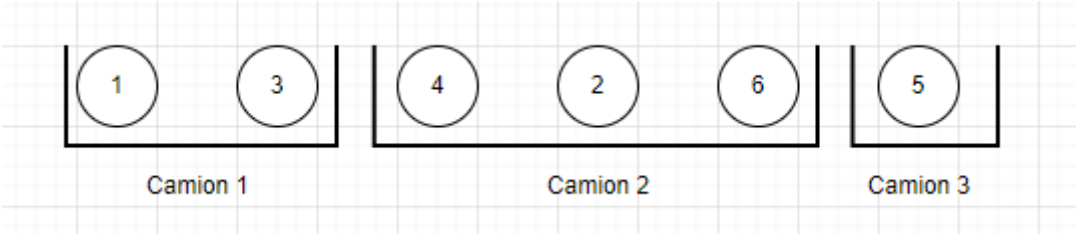
Dans ce cas, on applique l'algorithme génétique vu après le cahier de charge suivant:

- Etape 0 : Définir un codage du problème
- Étape 1 : t:=0, créer une population initiale de N individus P(0) = x1,x2,...,xN
- Etape 2 : Evaluation
    - Calculer la force F(xi) de chaque individu xi, i=1…N
- Etape 3 : Sélection
    - Sélectionner N individus de P(t) et les ranger dans un ensemble S(t). Un même individu de P(t) peut apparaître plusieurs fois dans S(t)
- Etape 4 : Recombinaison Grouper les individus de S(t) par paire, puis, pour chaque paire d'individus :
    - avec la probabilité Pcross, appliquer le croisement à la paire et recopier la progéniture dans S(t+1) (la paire d'individus est éliminée, elle est remplacée par sa progéniture),
    - avec la probabilité 1-Pcross, recopier la paire d'individus dans S(t+1)

  Pour chaque individu de S(t+1) :

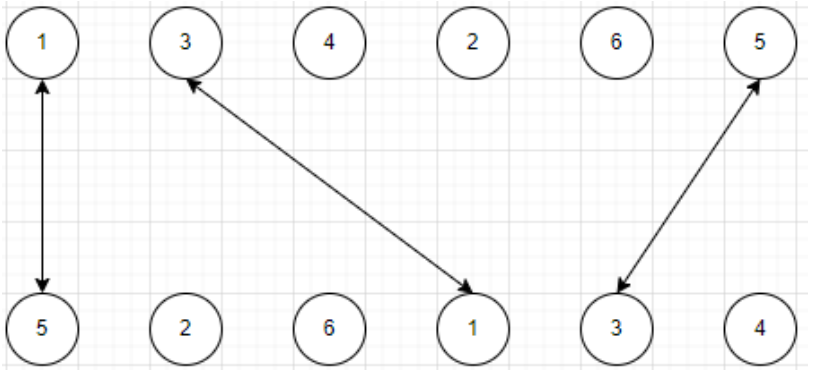
    - avec la probabilité Pmut, appliquer la mutation à l'individu le recopier dans P(t+1)
    - avec la probabilité 1-Pmut, recopier l'individu dans P(t+1)
- Etape 5 : Incrémenter t et reprendre à l'étape 2 jusqu'à un critère d'arrêt

On démarre avec l'ordonnancement, qu'on a utilisé pour les clients. On a choisi d' utiliser une chaîne de clients, qui représente la répartition dans le camion. Principalement, la chaîne sera lue à la fin du algorithmique et on va remplir chaque camion avec la capacité maximale c'est-à-dire lorsque le paquet du client suivant ne passe plus dans le camion, on le met dans le prochain. Pour éviter des trajets qui ne sont pas logiques, on donne en plus la possibilité pour l'algorithme de choisir un nouveau camion, si ce soit moins cher.



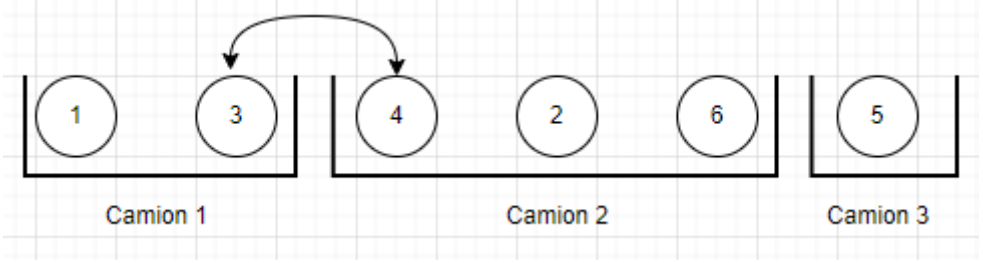
i. Le croisement

Vu que le découpage de la manière de faire le croisement présentée dans le cours n'est pas réalisable avec la structure appliquée, on choisit trois clients aléatoires et différents, dont l'ordonnancement sera changé, dans l'exemple ci-dessous les clients 1,3 et 5. Alors on obtient les ordonnances 135 pour la première et 513 pour la deuxième solution. Maintenant, pour effectuer un croisement, on change le premier client avec le premier de l'autre solution, le deuxième avec le deuxième etc. des deux clients. C'est-à-dire qu'on obtient l'ordonnance 513 pour la première solution et 135 pour la deuxième. Cette structure nous permet d'avoir une solution valable en tout cas, vu qu'un changement vertical comme présenté dans le cours est presque jamais valable avec la structure choisie (sauf si les clients choisis occupent les mêmes emplacements pour les deux clients). On garde cependant la logique du croisement et peut adapter des structures de l'autre solution



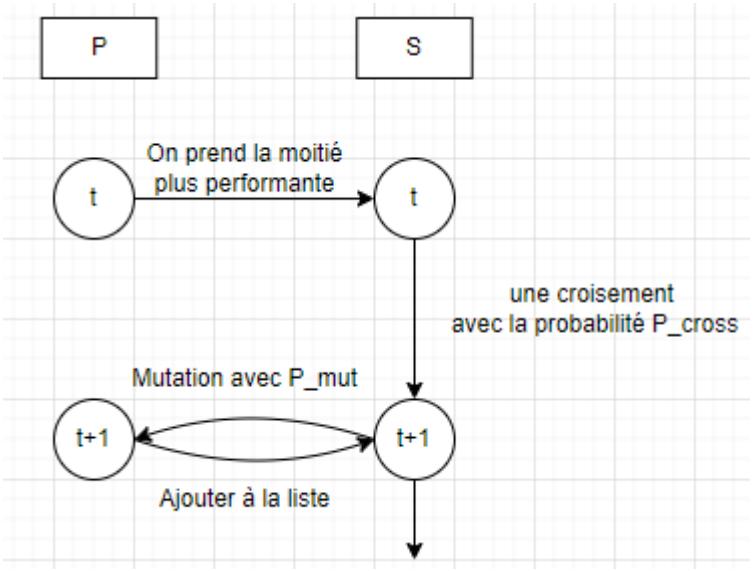
ii. La mutation

Pour la mutation on change l'ordonnance de deux clients côté à côté. On prend un client aléatoire de 1 à N-1 et change sa position avec son voisin à droite.



iii. Organisation

Alors on organise la structure de notre algorithme. On prend d'abord la moitié de la population P(t) qui performe le mieux comme S(t). Puis, on les met en couple et on effectue un croisement avec la probabilité P_cross et on garde tout dans S(t+1). Maintenant les gènes mutent avec la probabilité P_mut et seront stockés en P(t+1). Enfin, on les rejoint avec les gènes en S(t+1) et on peut recommencer le processus pour t+2. Cela nous permet de garder la taille de la population constant et éviter des évolutions imprévisibles (extinction ou population en croissance illimité), vu qu'on prend la moitié au début et double la population en S(t+1).

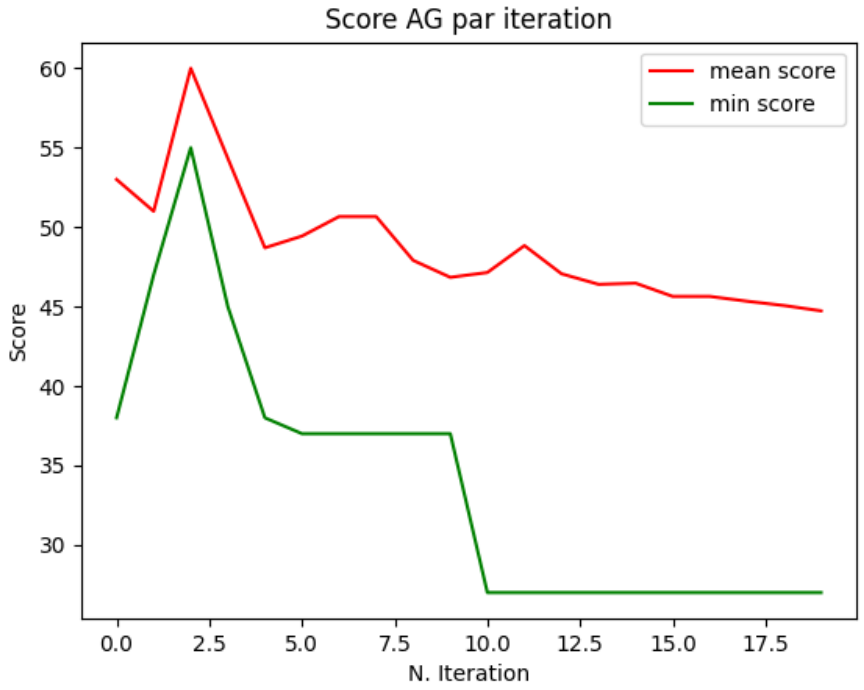


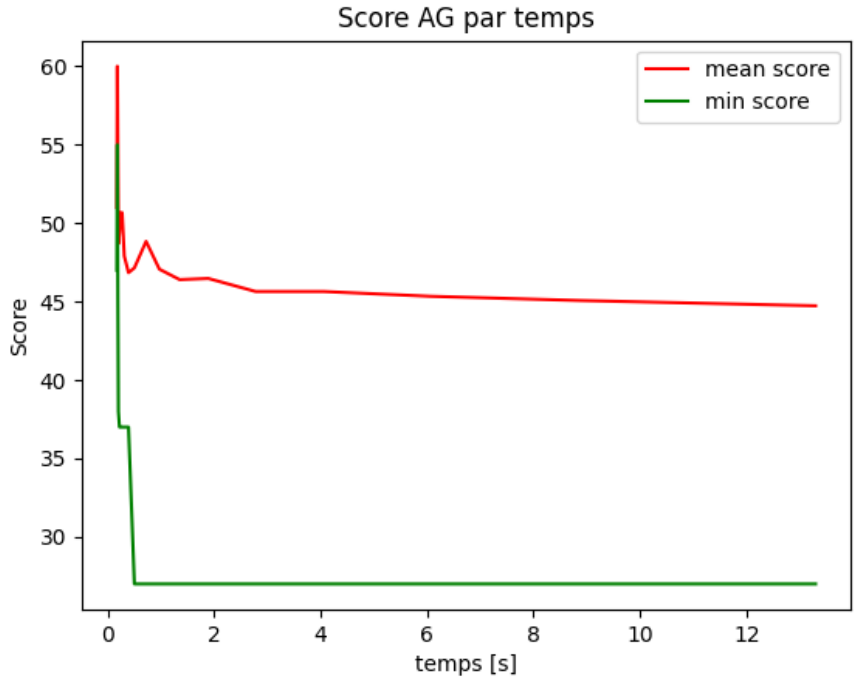
II.1.2. BDD de petite taille

II.1.2.1. Les courbes et tableaux

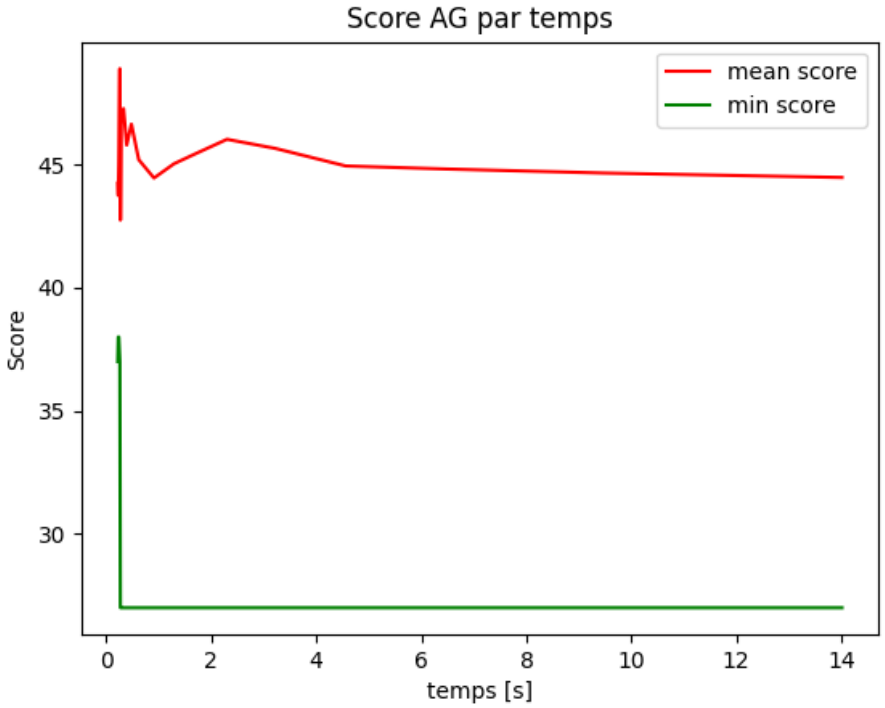
On applique l'algorithme génétique sur un système de 5 clients.La pénalité d'utiliser un nouveau camion est de 5. Nous lançons une simulation de 20 générations avec une population initiale de 4 chromosomes.

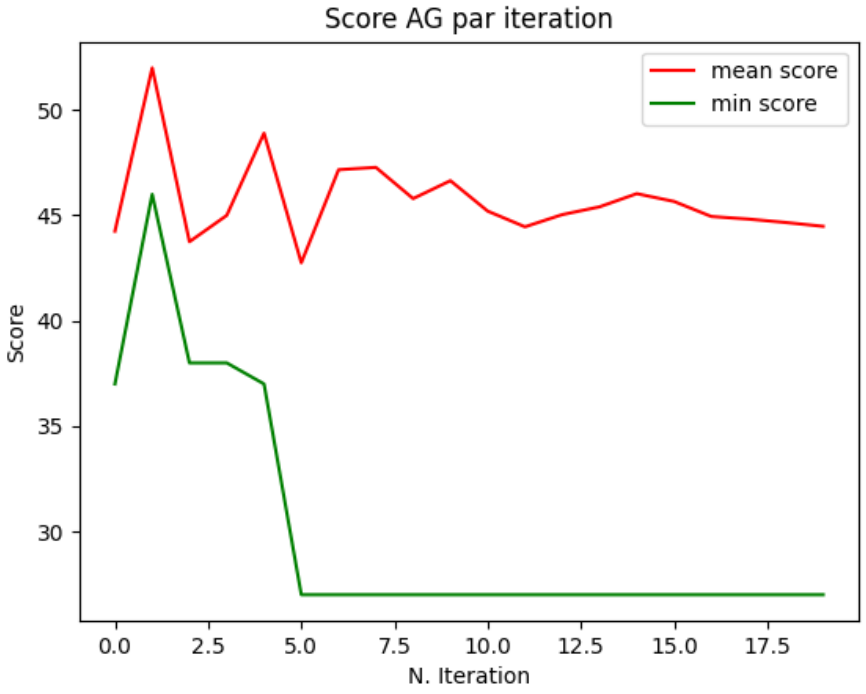
En premier temps on utilise comme **P_mut = 30%, P_cross = 30%**





Une deuxième simulation avec **P_mut = 60%, P_cross = 60%**





II.1.2.2. Analyse des résultats

On voit que l'algorithme fonctionne bien. On arrive à baisser le score minimal, même si la performance diminue pendant certaines itérations. Cet effet est dû au fait que le croisement et la mutation sont aléatoires (avec une probabilité d'application), alors parfois la meilleure solution est changé pour une pire. Globalement, on voit que la moyenne de la population se stabilise, même si le changement n'est pas très fort. Cela est lié au fait que avec un système si petit on n'arrive pas à trop de solutions différentes.

II.1.2.3. Intérêt de l'étude

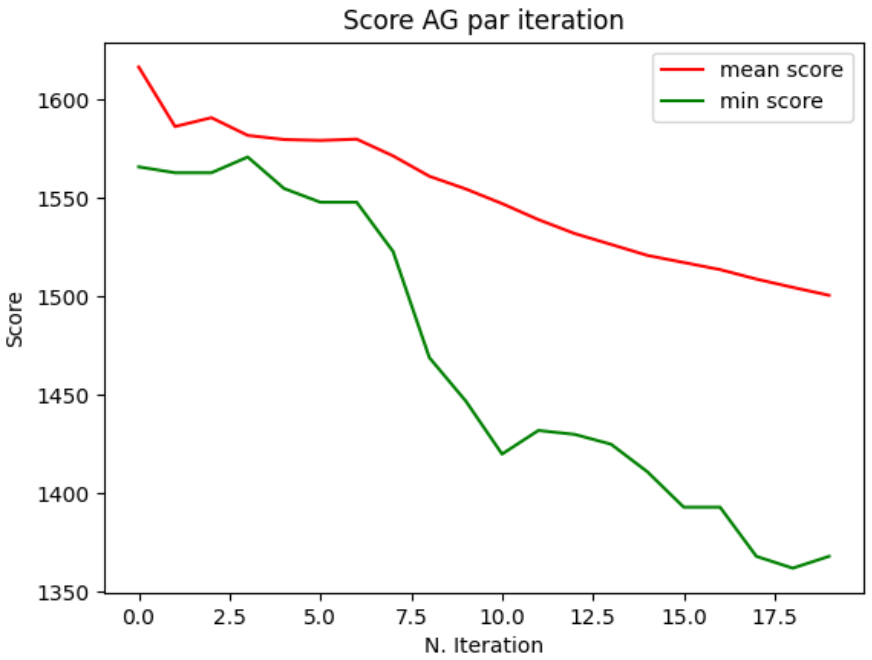
L'étude permet de vérifier la faisabilité de notre algorithme. Vu que le concept de l'algorithme génétique était modifié pour répondre aux exigences du PTVFT, on a pu observer que l'algorithme réalisé arrive à suivre les concepts de la théorie. Ainsi on peut réaliser une optimisation de grande taille ensuite.

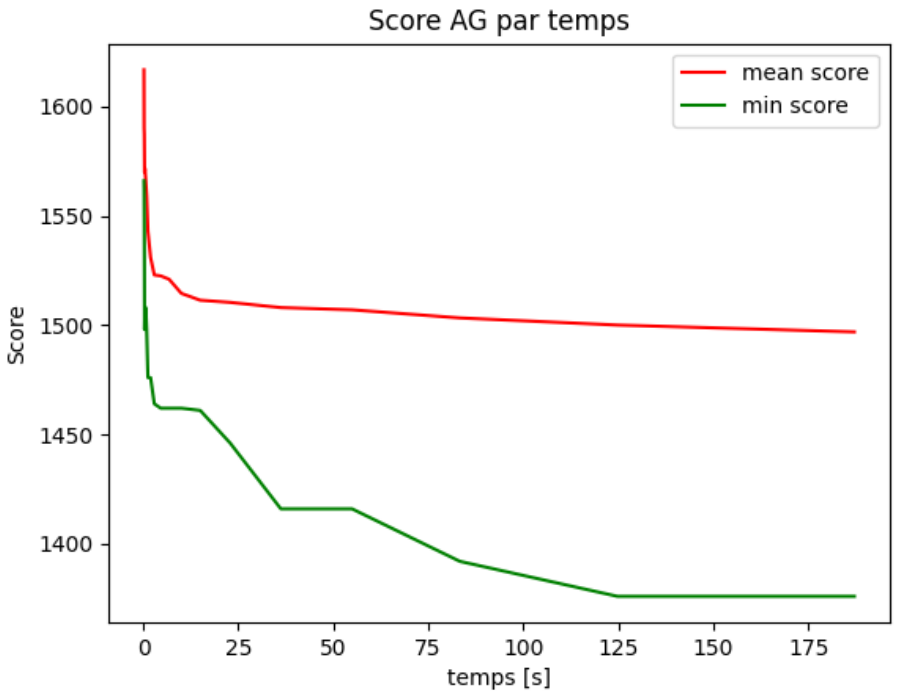
II.1.3. BDD de grande taille

II.1.3.1. Les courbes et tableaux

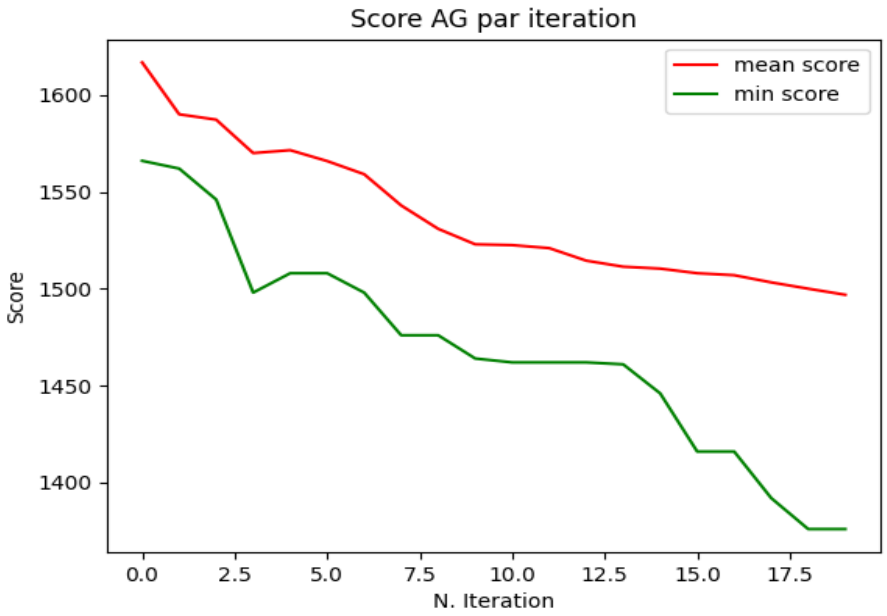
Simulation avec la matrice créée comportant 52 clients. Chaque client a une demande aléatoire entre 1 et 100. Et les camions ont une capacité totale de 100. La pénalité d'utiliser un nouveau camion est de 5. Nous lançons une simulation de 20 générations avec une population initiale de 4 chromosomes.

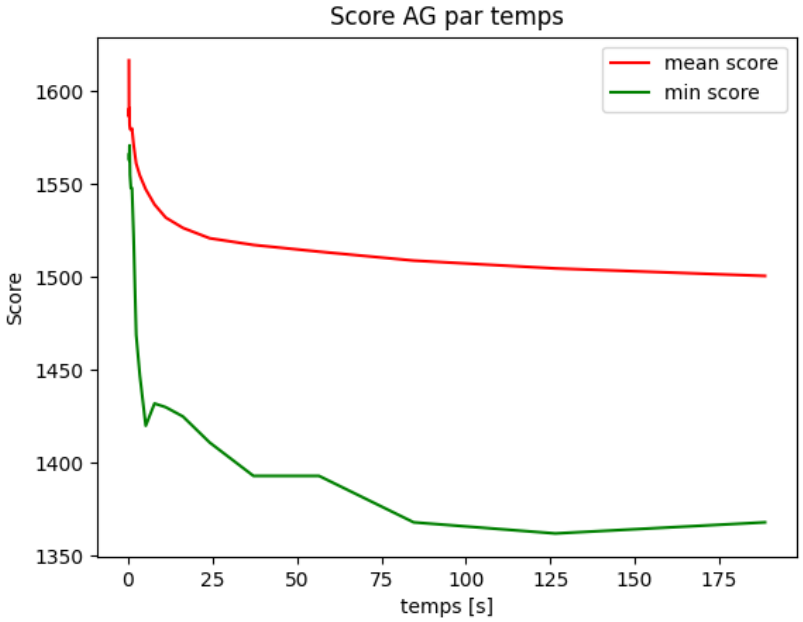
En premier temps on utilise comme **P_mut = 30%, P_cross = 30%**





Une deuxième simulation avec **P_mut = 60%, P_cross = 60%**





II.1.3.2. Analyse des résultats

Après simulation de 20 itérations, on observe que les scores des dernières populations ont diminué en moyenne. Et il y a une diminution pour l'individu qui comporte la solution avec le score minimal. Avec une probabilité plus grande de mutation, nous observons une chute plus grande dans le minimum du score qu'il trouve. On va étudier la dépendance des probabilité dans la partie de Q-Learning.

II.1.3.3. Intérêt de l'étude.

Cette étude de grande taille permet d'examiner la vraie performance par rapport du BDD de petite taille. Même si la performance par rapport à l'algorithme tabou est moins forte, l'algorithme permet de découvrir des nouvelles solutions avec sa manière aléatoire.

III. Optimisation Collaborative : SMA+Métaheuristiques

III. 1 SMA

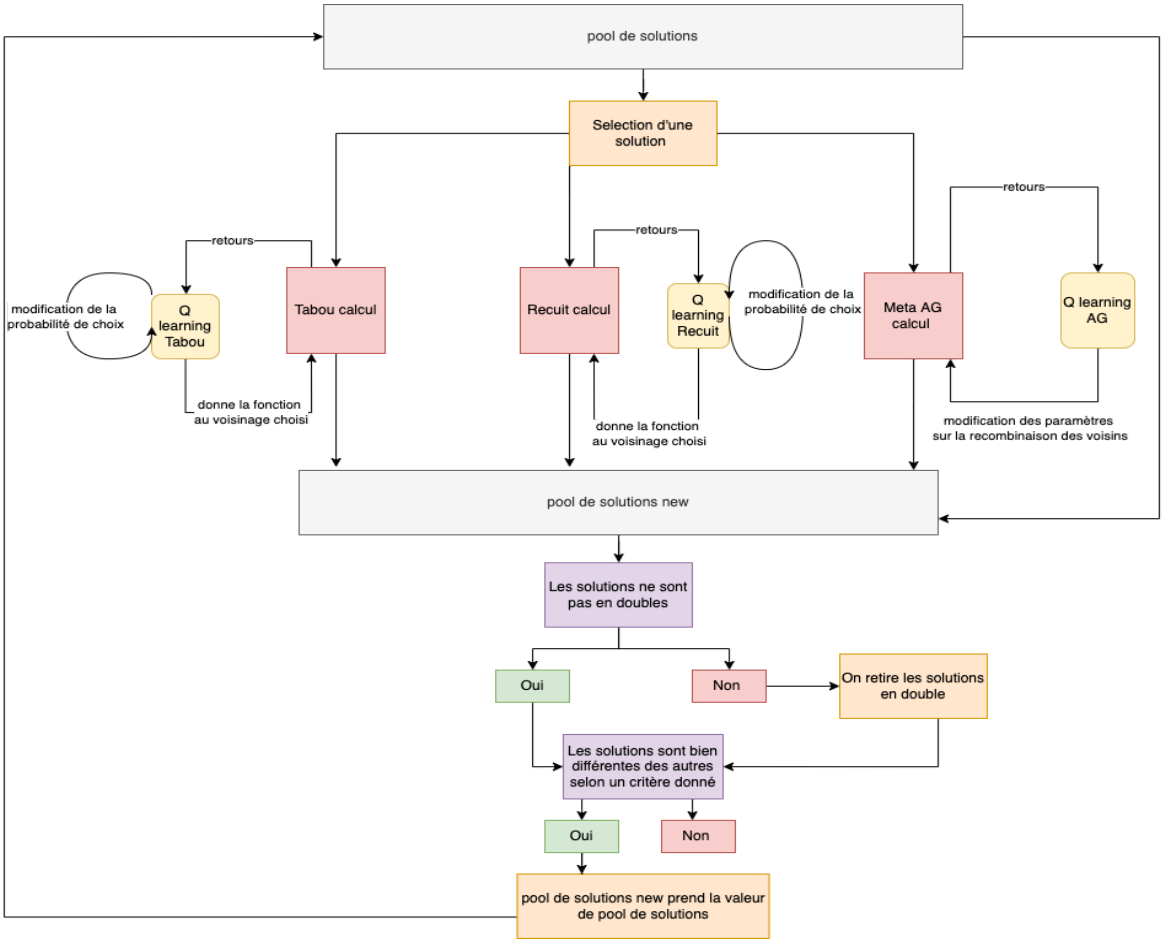
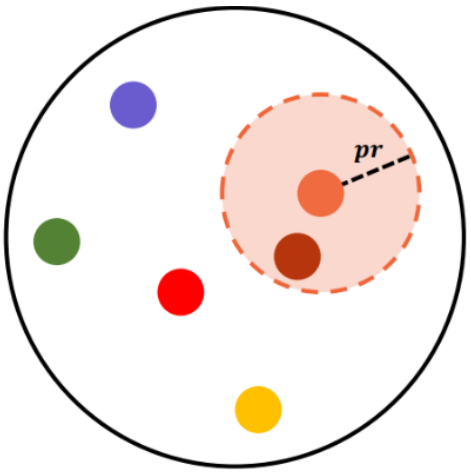


Schéma final de l'approche avec SMA

Sur le schéma ci-dessus, on a représenté le schéma global de notre algorithme, avec notamment le QLearning que nous allons voir dans IV.

Nous avons intégré notamment une prise de décision à la sortie de chaque algorithme:

$$g(\phi_i) = \sum_{j=1}^{P} \phi(\lambda_{ij}) \quad \text{où} \quad \phi(\lambda_{ij}) = \begin{cases} 1 - \frac{\lambda_{ij}}{pr} & \text{si } \lambda_{ij} \leq pr \\ 0 & \text{si } \lambda_{ij} > pr \end{cases}$$



Nous avons implémenté une méthode pour choisir si on garde ou pas une solution, on va choisir des solutions suffisamment différentes (comparaison entre deux solutions: nombre d'arêtes en commun).
Cela nous permet de garder une grande diversité et ainsi d'explorer diverses solutions.

III. 2 Comparaison Agents Amis et Ennemie

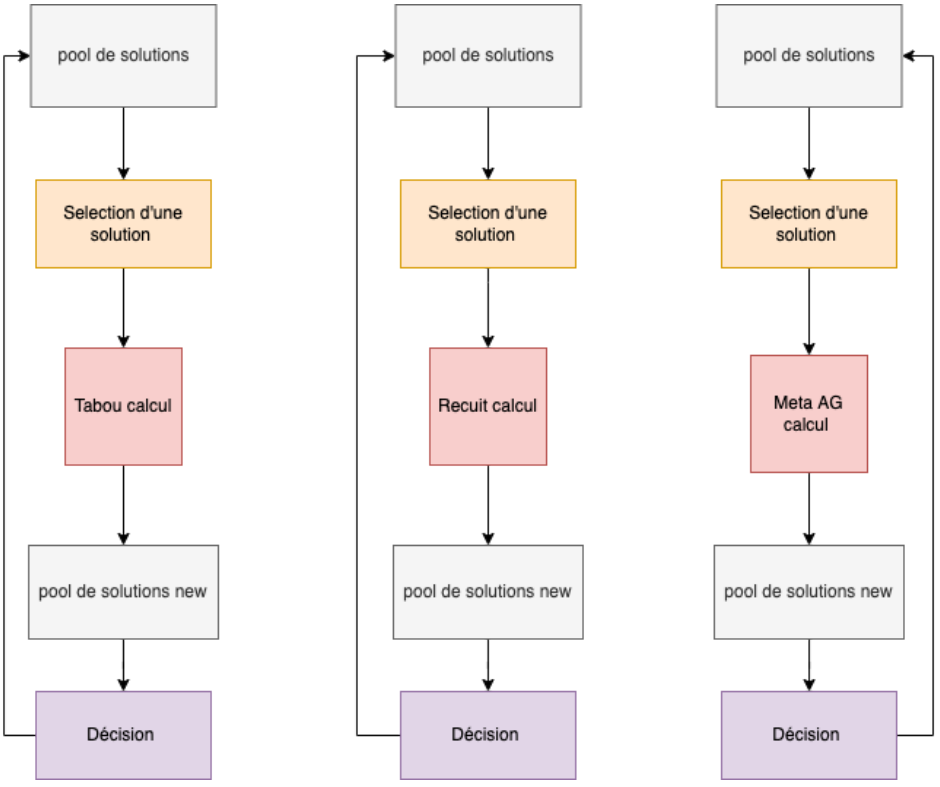
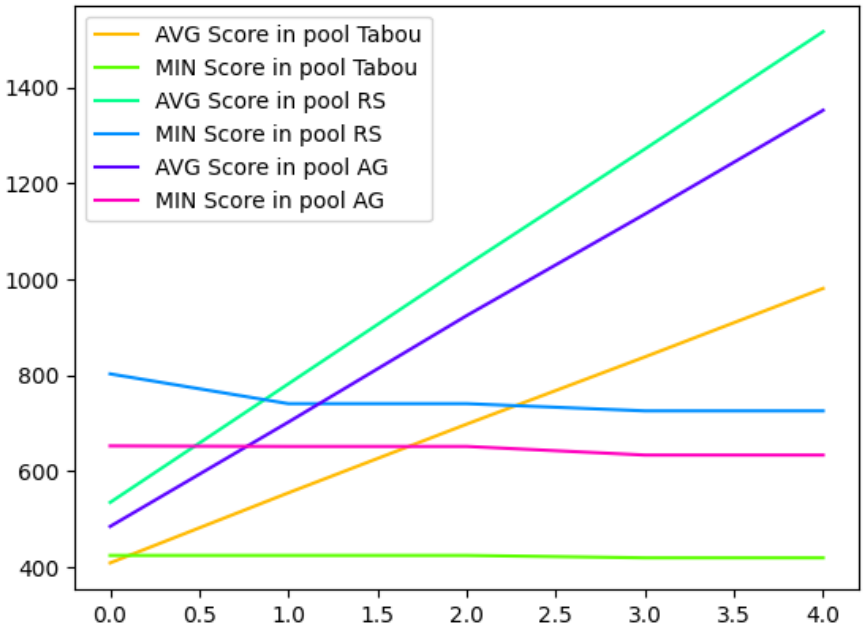


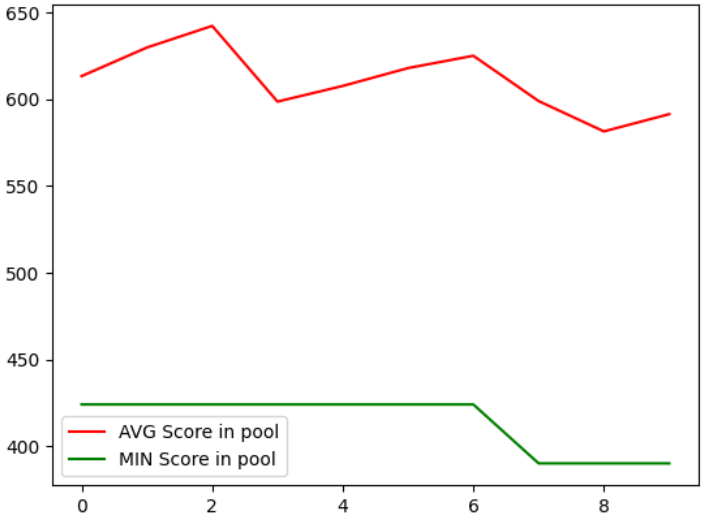
Schéma de l'approche ennemie

Pour l'approche ennemie on a simplement décidé d'implémenter 3 pools différents. Ainsi on a des approches en vase clos.

On obtient les graphiques suivant:



Graphique de l'approche ennemie (coût en fonction du nombre d'itération)



Graphique de l'approche ami (coût en fonction du nombre d'itération)

On voit qu'entre l'approche ami et l'approche ennemie, on a une plus grande rapidité de convergence de l'approche ami, et les score moyen de la pool reste globalement constant dans l'approche ami.

III. 3 Améliorations possibles

On a pu en voir notamment en comparant l'approche ennemi et ami que les algorithmes génétiques et RS sont plus lent pour converger et explorent plus de solutions tandis que Tabou converge rapidement.

Une idée serait de faire une approche semi-ennemie semi-amie avec une alternance de pool tabou et de pool RS/AG. L'idée serait de laisser les algorithmes RS/AG trouver des solutions éloignées puis d'inverser les pools pour que Tabou puisse trouver un minimum à partir des solutions trouvées par RS/AG.

III. 4 Amélioration du Scheduler

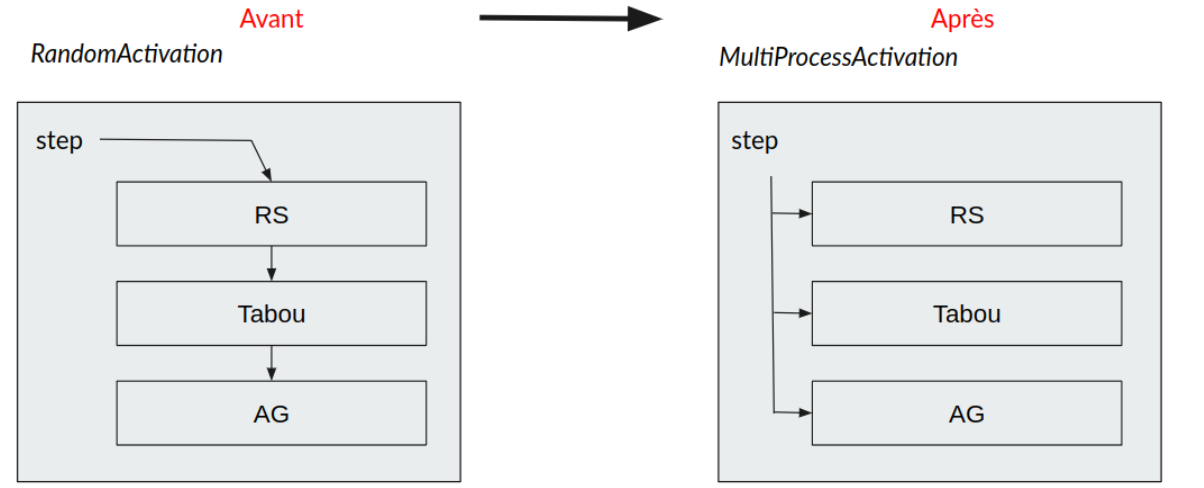
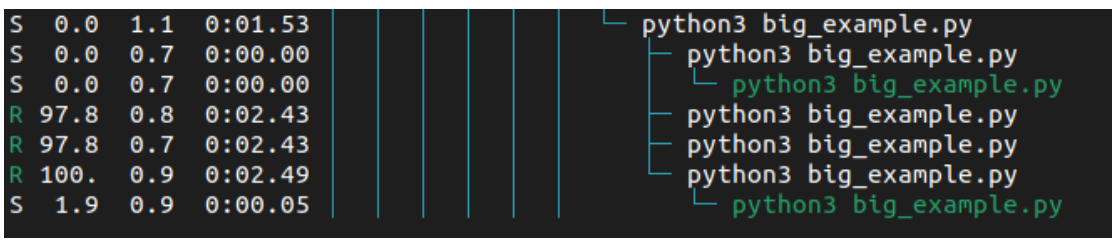


Schéma expliquant le passage à l'activation "MultiProcessActivation"

Nous avons décidé d'améliorer le Scheduler afin de pouvoir tirer parti des processeurs multicoeurs. En effet, par défaut, les calculs se font sur le même coeur, ce qui est beaucoup plus lent que si on répartit l'exécution des 3 agents sur 3 coeurs différents.

On peut voir sur la capture d'écran suivante que le calcul se fait sur 3 processeurs différents.



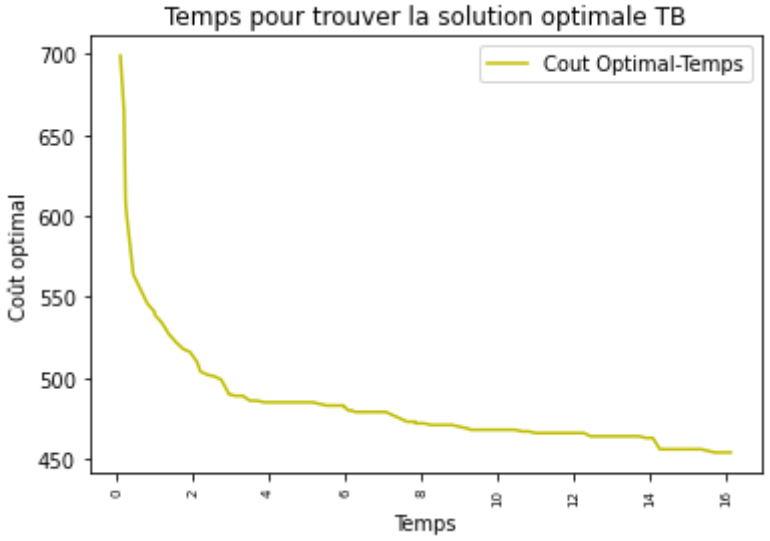
Capture d'écran de la commande "htop" sous Linux

IV. Optimisation Collaborative : SMA + Métaheuristiques + QL

IV. 1 Tabou

IV.1.1 Algorithme spécifique PTVFT

On a pris ici la même structure que pour l'algorithme Tabou dans le cas sans QLearning, cependant ici on a fait intervenir la décision de QLearning dans la recherche du voisinage: ainsi nos états représentent les différentes façons de choisir le voisinage. Ainsi nous avons 3 états, pour le choix du voisinage : relocate, exchange, exchange_inside.

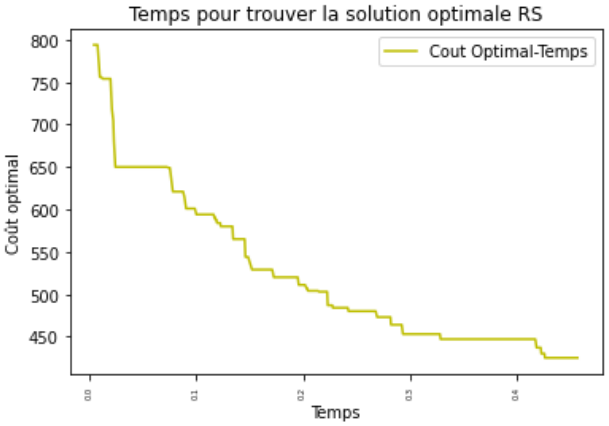


IV. 2 Recuit Simulé

IV.1.1 Algorithme spécifique PTVFT

On a pris ici la même structure que pour l'algorithme Tabou dans le cas sans QLearning, cependant ici on a fait intervenir la décision de QLearning dans la recherche du voisinage: ainsi nos états représentent les différentes façons de choisir le voisinage. Ainsi nous avons 8 états, pour le choix du voisinage : intra_route_swap, inter_route_swap, intra_route_shift, two_intra_route_swap, two_intra_route_shift, del_small_route_w_capacity, del_random_route_w_capacity.

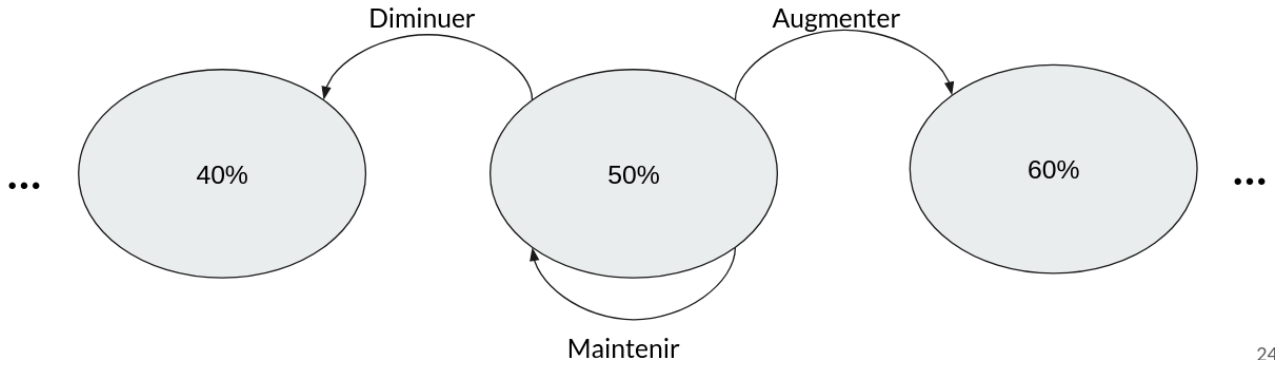
Chacune des méthodes de choix de voisinage est assez explicite. On a veillé à faire attention au fait que les solutions doivent respecter les critères de capacités maximales.



IV.3 Algorithme génétique

IV.2.1 Algorithme spécifique PTVFT

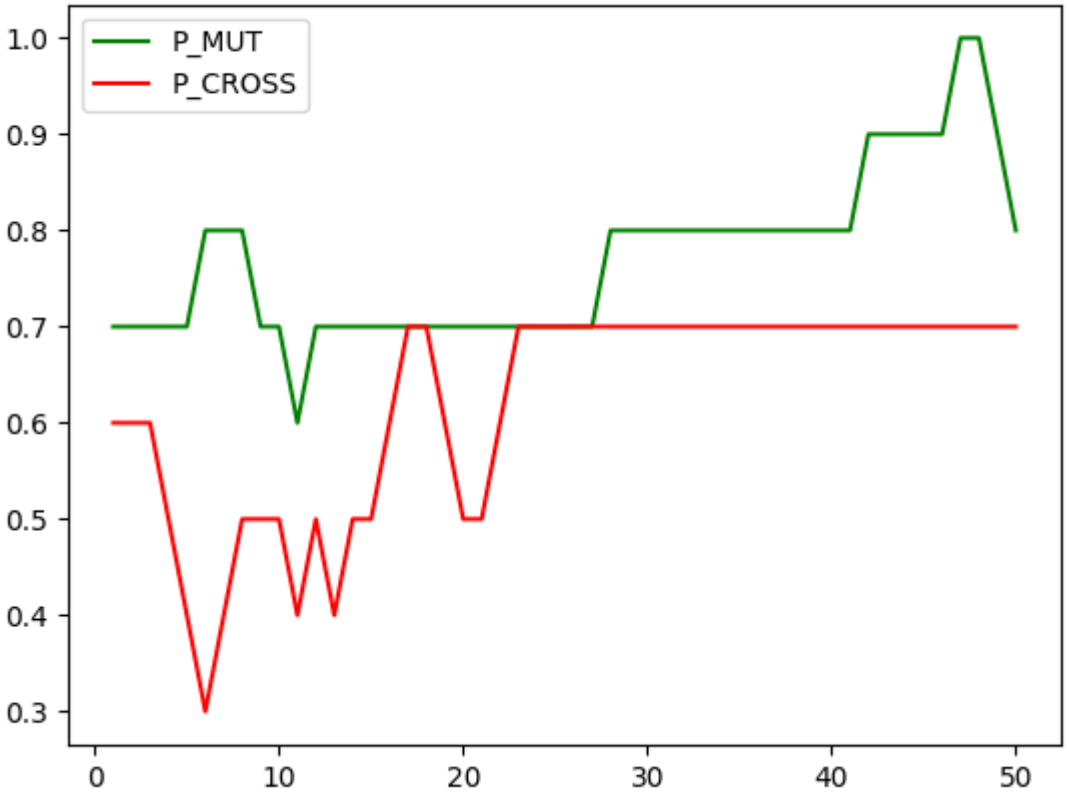
Pour le QLearning dans l'algorithme génétique nous avons décidé de faire une approche par rapport au paramètre propres de l'algorithme. L'algorithme compte avec deux paramètres P_MUT et P_CROSS qui ont été détaillé dans le points II.3.1. Dans un premier temps nous avons fixé c'est parametre soit en 60% soit en 30%, mais c'était une décision arbitraire. Grâce aux QLearning nous voulons avoir une décision basée dans un reward, qui est défini comme le score retrouvé par la meilleur solution obtenue par l'algorithme.
Pour cela nous avons définis 10 états possibles: 10%, 20%, 30%, 40%, 50%, 60%, 70%, 80%, 90%, 100%. Et pour chaque état possible nous avons trois actions possibles que l'Agent peut prendre. Soit maintenir la valeur actuelle, soit diminuer de 10% si possible, soit augmenter en 10%. Ceci est représenté dans le diagramme suivant:

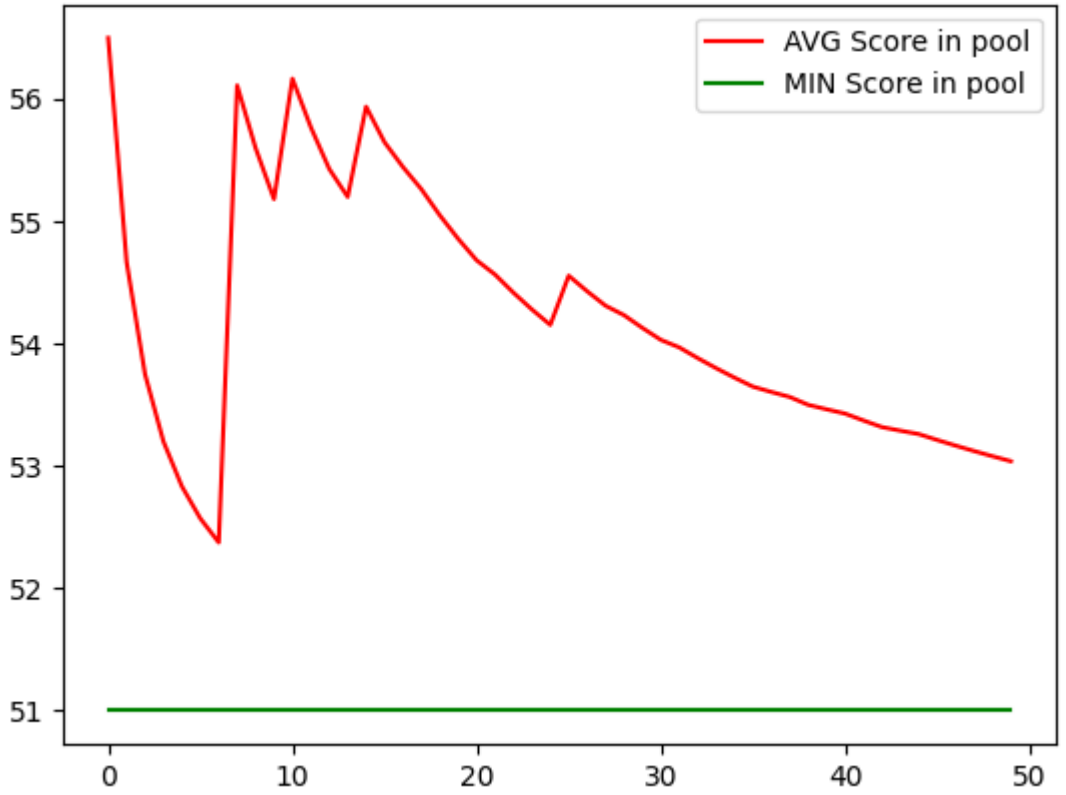


24

IV.2.2 BDD de petite taille

IV.2.2.1 Les courbes et tableaux





IV.2.2.2 Analyse des résultats

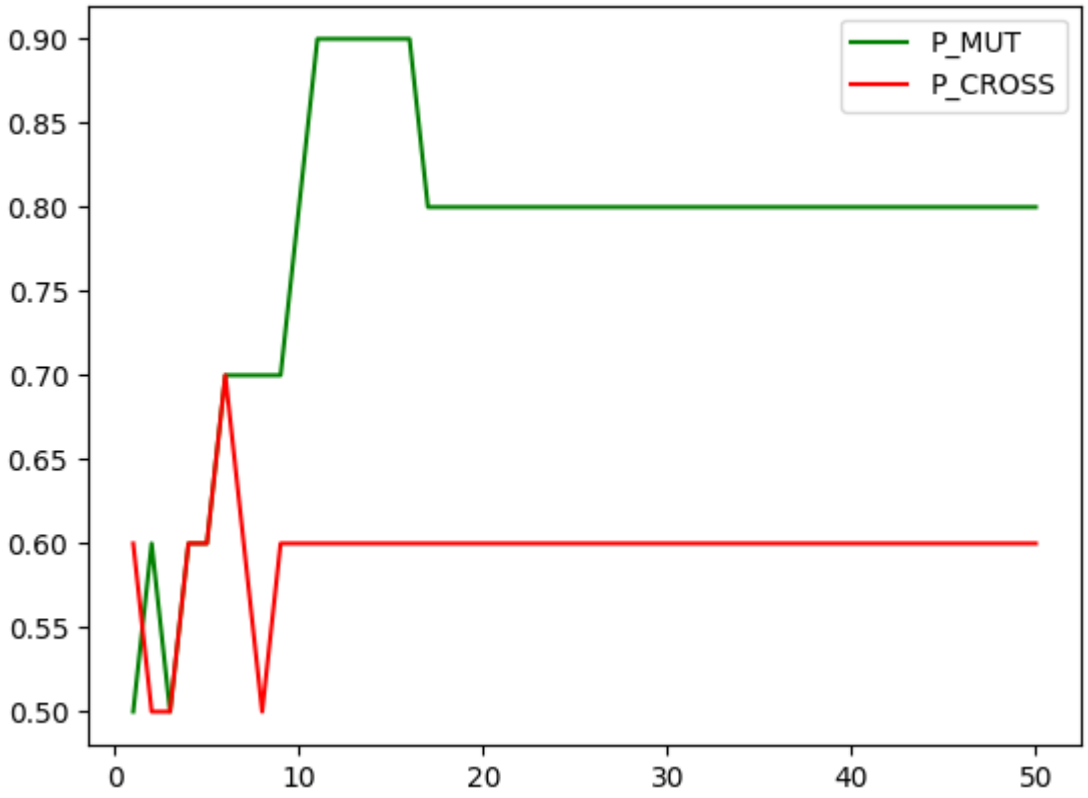
Après une simulation avec la petite base de données, on remarque que l'algorithme trouve la meilleure solution dès la première itération. Mais si on note le score moyen des solutions, proche de la dixième itération on voit comme elle augmente abruptement. Ceci peut être relié au fait que le QLearning permet ainsi d'exploiter mais aussi d'explorer des nouvelles solutions. Et dans ce cas là c'est grâce au changement des paramètres.

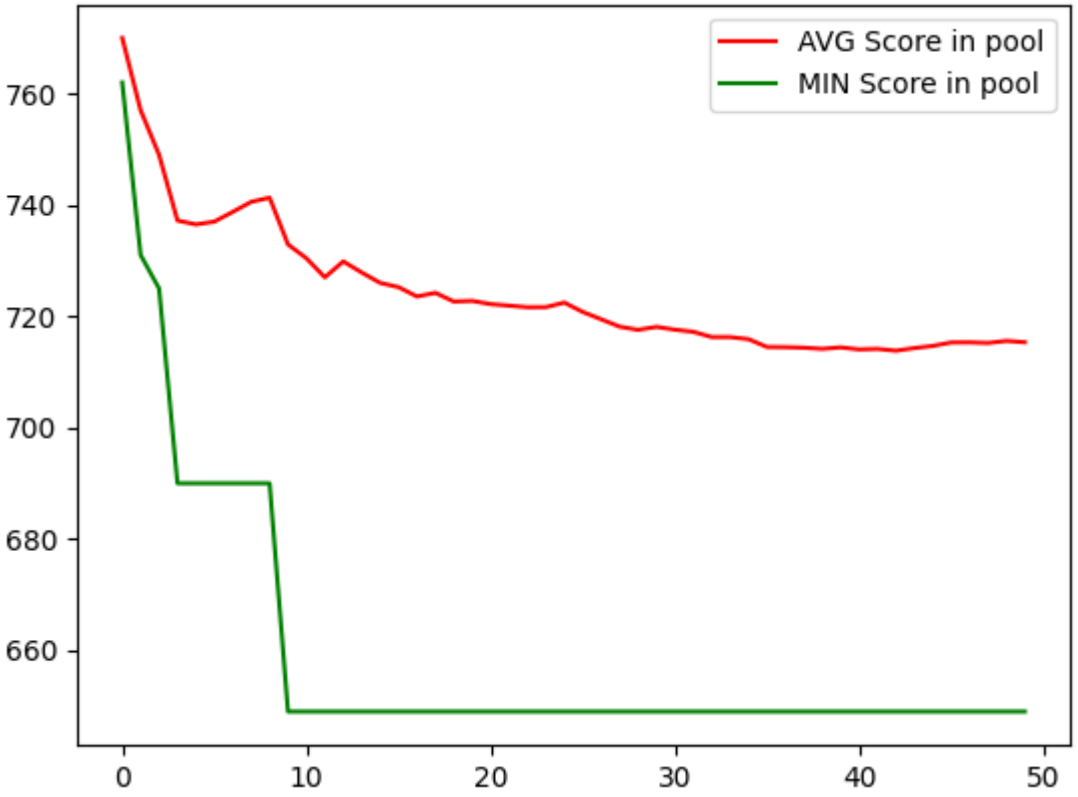
IV.2.2.3 Intérêt de l'étude

Cette simulation nous permet de voir que même si l'algorithme trouve la solution optimale, QLearning réalise d'actions random pour explorer.

IV.2.2 BDD de grande taille

IV.2.2.1 Les courbes et tableaux





IV.2.2.2 Analyse des résultats

Nous pouvons retrouver un changement plus grand dans les paramètres, ceci grâce à la meilleure solution dans un premier temps. Nous observons qu'après quelques itérations, les deux paramètres convergent vers une valeur stable, ce qui permet de trouver une meilleure solution.

IV.2.2.3 Intérêt de l'étude

Cette simulation permet de voir que le QLearning permet d'améliorer l'efficacité de l'algorithme grâce à la prise en décision d'un meilleur paramétrage.

V. Tableaux de comparaison et analyses globales des performances de l'optimisation collaborative

Pour un exemple similaire, avec une cinquantaine de clients, une capacité des camions de 100, une demande des clients variants de 1 à 3 (dans les algorithmes Tabou et Recuit Simulé) et une demande aléatoire (de 1 à 100) des clients dans la méthode AG, et un coût de pénalité ω fixe de 5 par ajout de camion.

<u>Comparaison sans collaboration</u>

Demande client	Nombre de clients	Métaheuristique			Métaheuristique Apprentissage		
		AG	Tabou	RS	AG	Tabou	RS
1	5	coût : 25 nombre d'itérations : 10	coût : +50	coût : 30 nombre d'itérations : + 300	coût : 25 nombre d'itérations : 1		
1	50		coût : 370 nombre d'itérations : 200 temps 70 secondes	coût : 550 nombre d'itérations : + 200 temps 70 secondes			
3	50		coût : 430 nombre d'itérations : 200 temps 70 secondes	coût : 700 nombre d'itérations : + 200 temps 70 secondes		coût : 430 nombre d'itérations : 200 temps 15 secondes	coût : 430 nombre d'itérations : 200 temps ≈ 10 secondes
Aléatoire entre 1 et 100	50	coût : 1350 nombre d'itérations : 20 temps 150 secondes			coût : 650 nombre d'itérations : 50		

Bien qu'il est difficile de comparer les deux premières méthodes et la troisième puisque le nombre total de commandes n'est pas le même (et donc le coût de la méthode 3 sera *de facto* nécessairement plus élevé), après cette première approche on peut tout de même arriver au constat que les méthodes Tabou et AG semble plus efficace en temps et en nombre d'itérations que la méthode du Recuit Simulé.

D'autre part, on remarque que l'apprentissage par renforcement appliqué aux différentes métaheuristiques améliore très largement le coût optimal pour certaines méthodes et réduit le temps de calcul de l'algorithme.

<u>Comparaison avec collaboration</u>

Nombre de clients	Métaheuristique SMA sans apprentissage		Métaheuristique SMA avec apprentissage	
	amis	ennemis	amis	ennemis
50	coût optimal du pool de solution : 400 8 itérations	coût otpimal moyen : 600 (en focntion de la méthode) 5 itérations	coût optimal du pool de solution : 350 dizaines d'itérations	coût otpimal moyen : 500/450 (en focntion de la méthode)

Enfin avec une base de données de taille moyenne, on note que la méthode de calcul avec collaboration est nettement plus performante que la méthode dite ennemis pour un même pool de solutions initial.
Et *a fortiori* comme précédemment lorsque l'on applique une méthode d'apprentissage aux métaheuristiques la performance et la précision du calcul d'optimisation s'améliore à nouveau.

Conclusion et perspectives

Nous avons réussi à appréhender l'intérêt des métaheuristiques et des systèmes multi-agents. Nous avons également pu comprendre la nécessité de mixer les approches amis et ennemies. Le QLearning était intéressant à mettre en place. En plus de l'implémentation sur le choix de la méthode de choix du voisinage, on aurait pu le mettre en place pour la sélection des hyperparamètres des algorithmes RS et Tabou.

On aurait aussi pu étendre notre problème en définissant des camions de taille non constante, et optimisant le choix des camions en fonction de leur capacité maximum.

Enfin on a réussi à optimiser le temps de calcul grâce à du calcul multi-processeur, ce qui nous a permis de tester plusieurs scénarii rapidement.

Annexes

Code: https://github.com/cedced19/ICO-fil-rouge