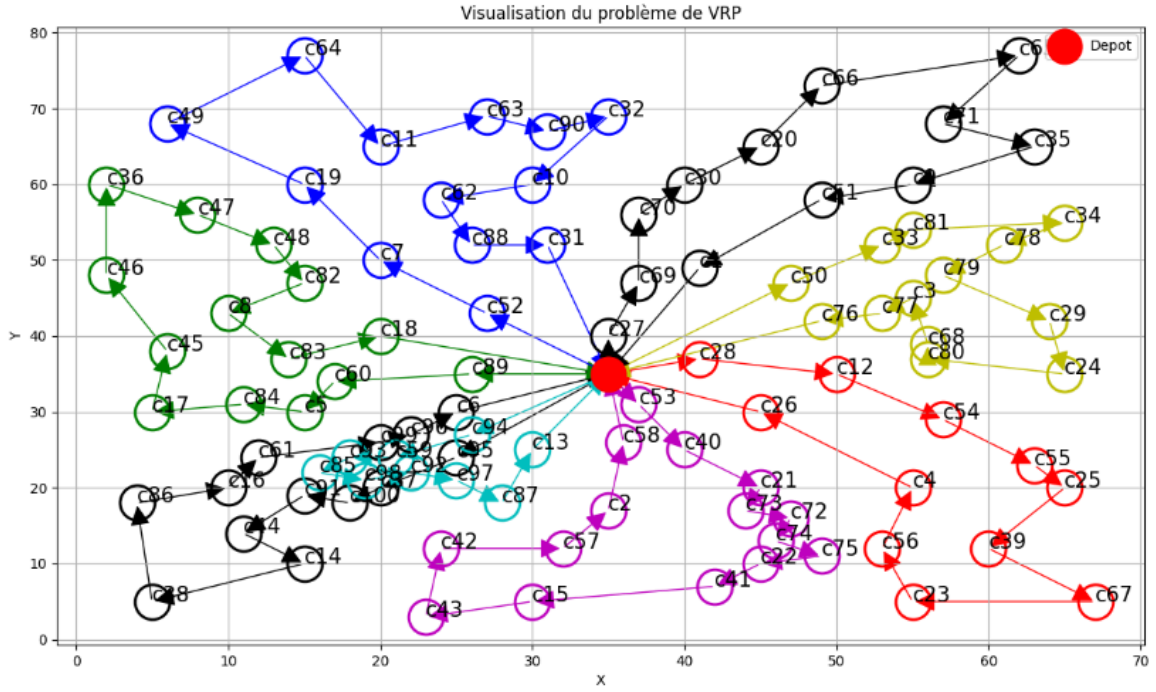
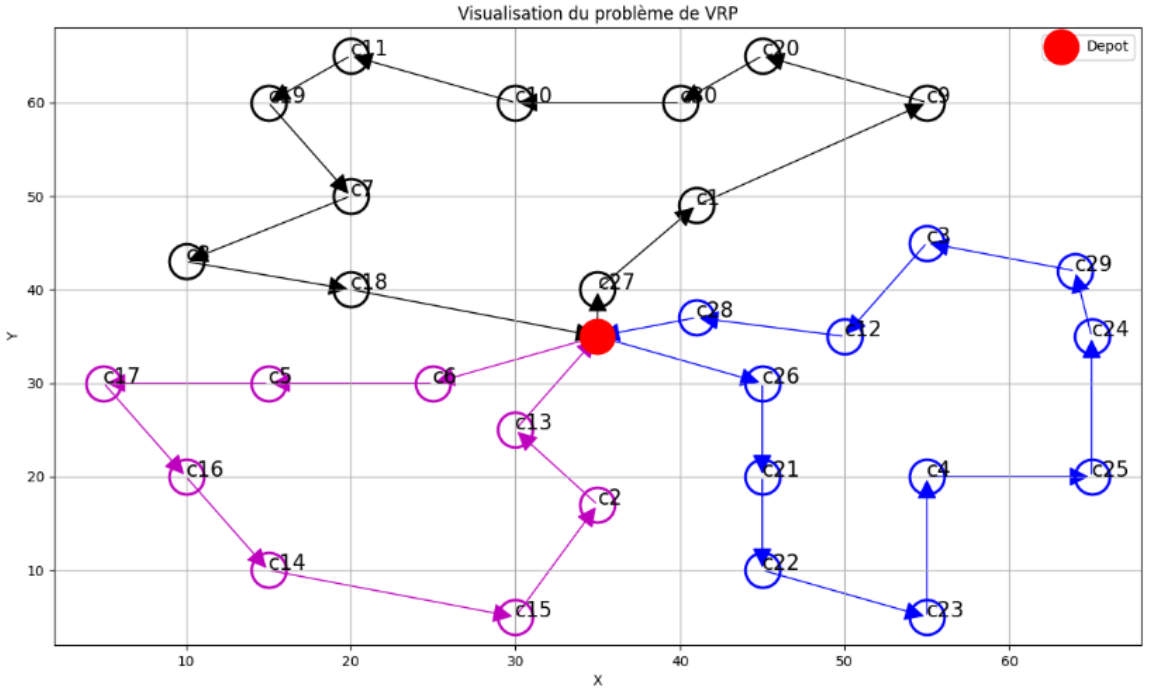
CHAIZE Alice
DUCHENE Guillaume

Projet – Capacitated Vehicle Routing Problem





Sommaire

Introduction

Le problème d'acheminement de véhicules avec capacité (Capacitated Vehicle Routing Problem) est un problème très connu dans le monde la livraison. En effet, trouver l'itinéraire à utiliser afin d’avoir besoin du moins de camion possible tout en respectant les différentes contraintes imposées est un problème récurrent.

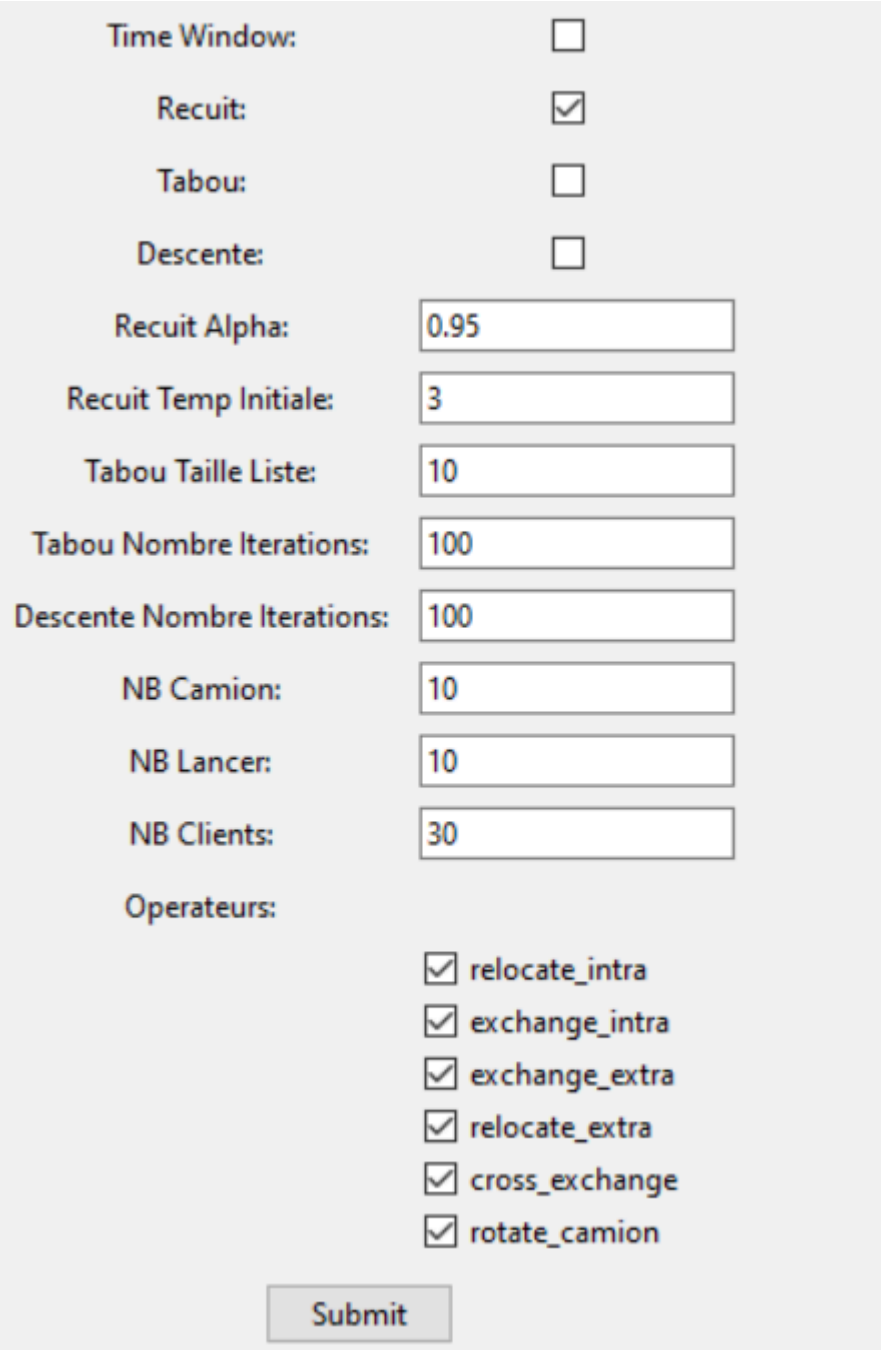
Dans notre cas nous avons pour contraintes d’utiliser les capacités de nos camions à leur maximum, tout en faisant en sorte qu’un client soit visité par un unique camion. Tous les clients doivent être évidemment livrés et ceux dans la limite de temps données impartie. Tout cela doit être fait en minimisant la distance que parcourt chaque camion.

Afin de résoudre ce problème, nous avons utilisé deux métaheuristiques, la méthode Tabou et le recuit simulé. Nous avons développé un programme python qui permet de définir les chemins à emprunter, le nombre de camions à utiliser et afin de se représenter tout cela un graphe montrant la solution obtenue.

Ce rapport continent l’explication de notre raisonnement sur le problème donné et comment nous avons fait pour obtenir cette solution. Vous trouverez également le résultat de simulation que nous avons pu effectuer.

Code

Voici la fenêtre d'exécution du code qui permet de choisir les paramètres souhaités et la métaheuristique voulue. Pour cela, il suffit d'exécuter le fichier main, qui lance la méthode start(). Les résultats sont dans la console et à la fin du programme une visualisation est affichée ainsi qu'une écriture des résultats dans un fichier CSV.

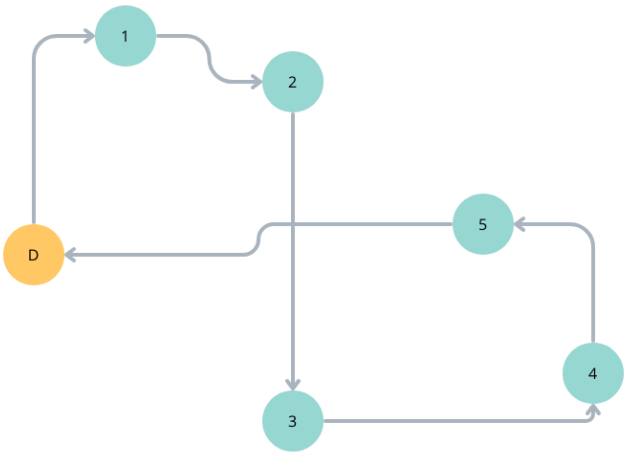


Opérateurs de voisinage

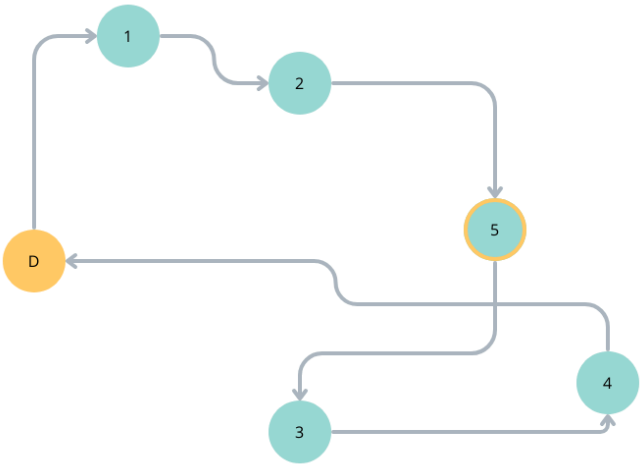
Dans cette partie, nous allons vous expliquer les différents opérateurs que nous avons mis en place.

Relocate intra

Cette opération nous permet de changer l'ordre dans lequel les clients seront visités par un même camion. Cela permet donc de changer les chemins par lesquels va passer notre camion.
Par exemple le trajet du camion bleu représenté sur ce graphique :

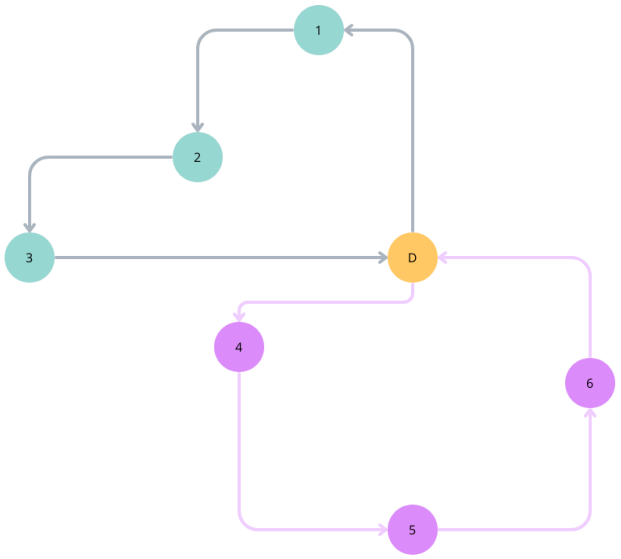


Devient après l'opération relocate celui-ci :

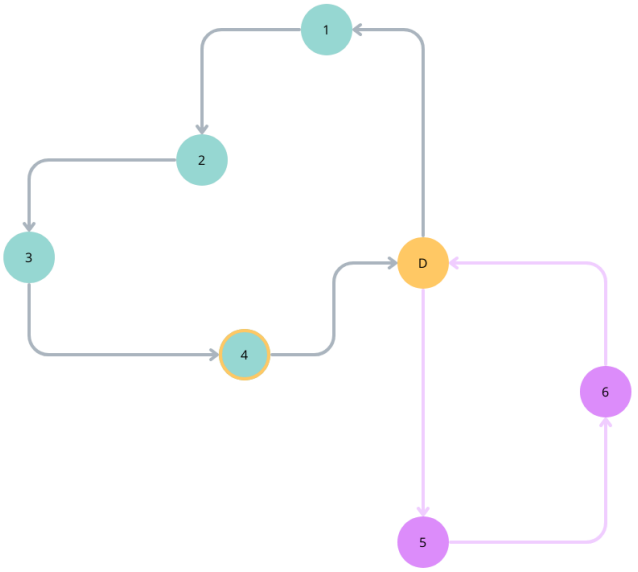


Relocate extra

L’opération relocate extra va changer de tournée un client, il ne sera plus livré par le même camion. C’est un opérateur qui permet de supprimer un camion ce qui nous permet d’améliorer grandement les solutions.
Par exemple avant utilisation de l’opération les trajet du camion bleu et du camion rose ressemble à ça :

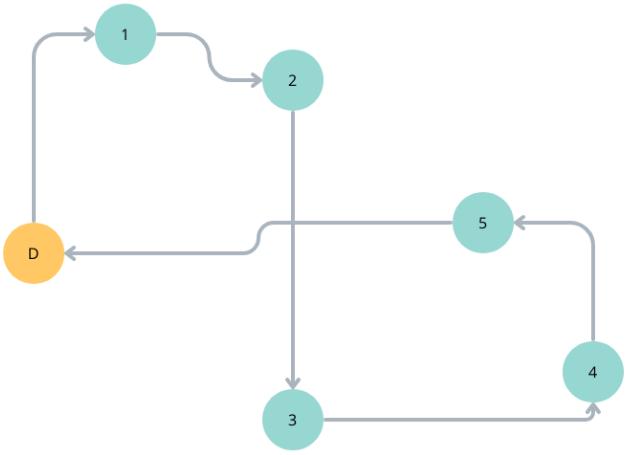


Et une fois l’opération exécutée les trajets sont devenu comme ceci :

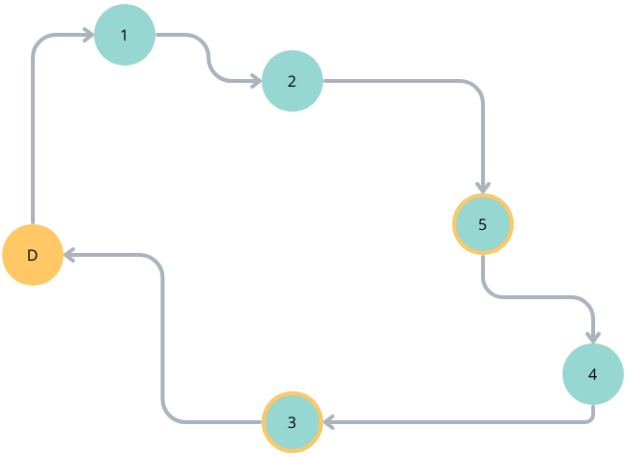


Exchange intra

L’exchange-intra va quant à lui nous permettre d'échanger deux clients au sein de la même tournée.
Par exemple le trajet du camion bleu représenté sur ce graphique :

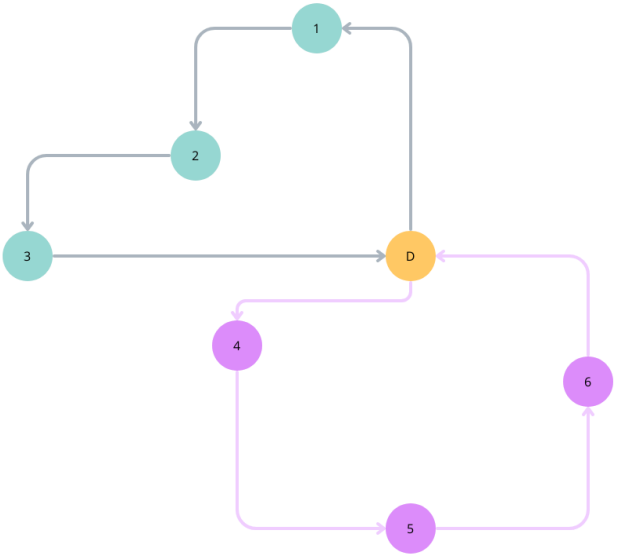


Après l’opération le trajet du camion bleu ressemble à ça :

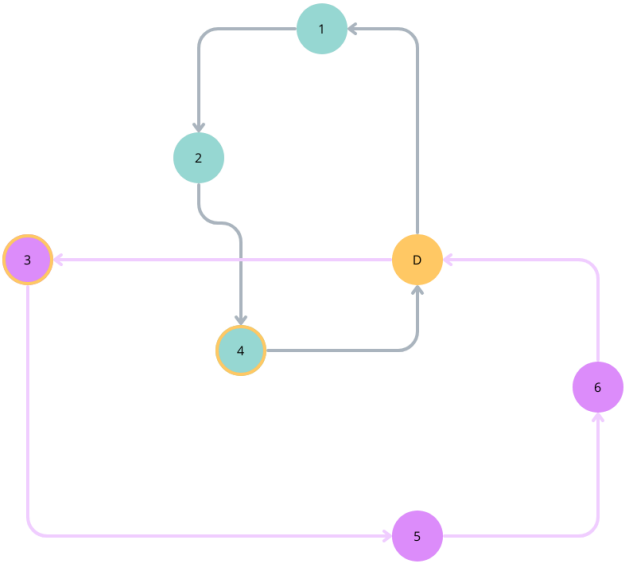


Exchange extra

Cette même opération mais extra va elle aussi permettre d'échanger deux clients mais cette fois entre deux tournées différentes.
Par exemple avant utilisation de l’opération les trajet du camion bleu et du camion rose ressemble à ça :

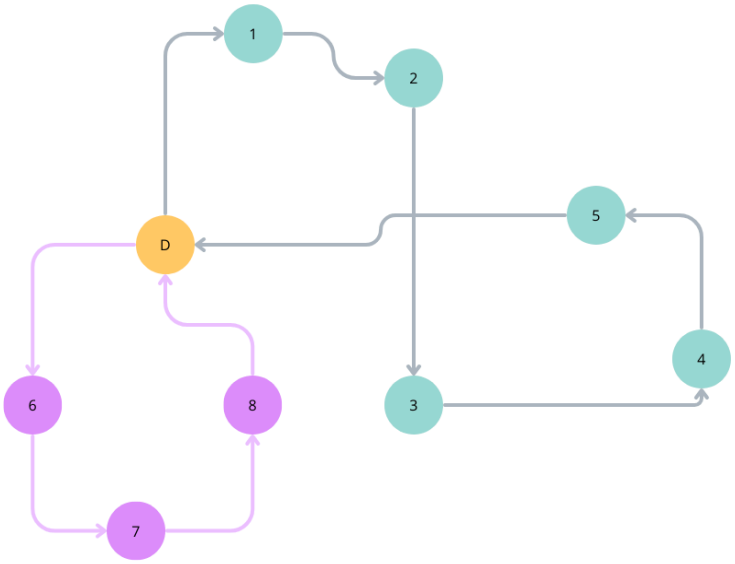


Et une fois l'opération exécutée les trajets sont devenu comme ceci :

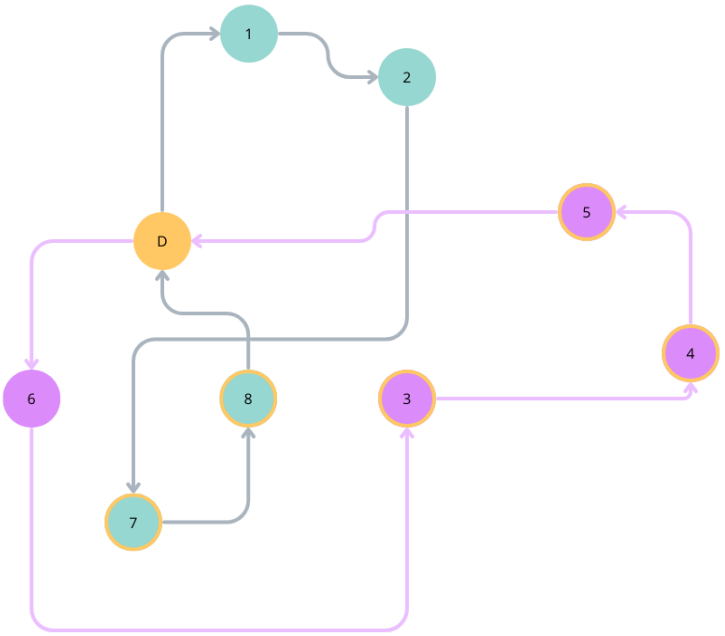


Cross exchange

L'opération cross exchange va nous permettre d'échanger un ensemble de client d'une tournée contre un ensemble de client d'une autre tournée.
Par exemple avant utilisation de l'opération les trajet du camion bleu et du camion rose ressemble à ça :



Et une fois l'opération exécutée, les trajets sont devenus comme ceci :



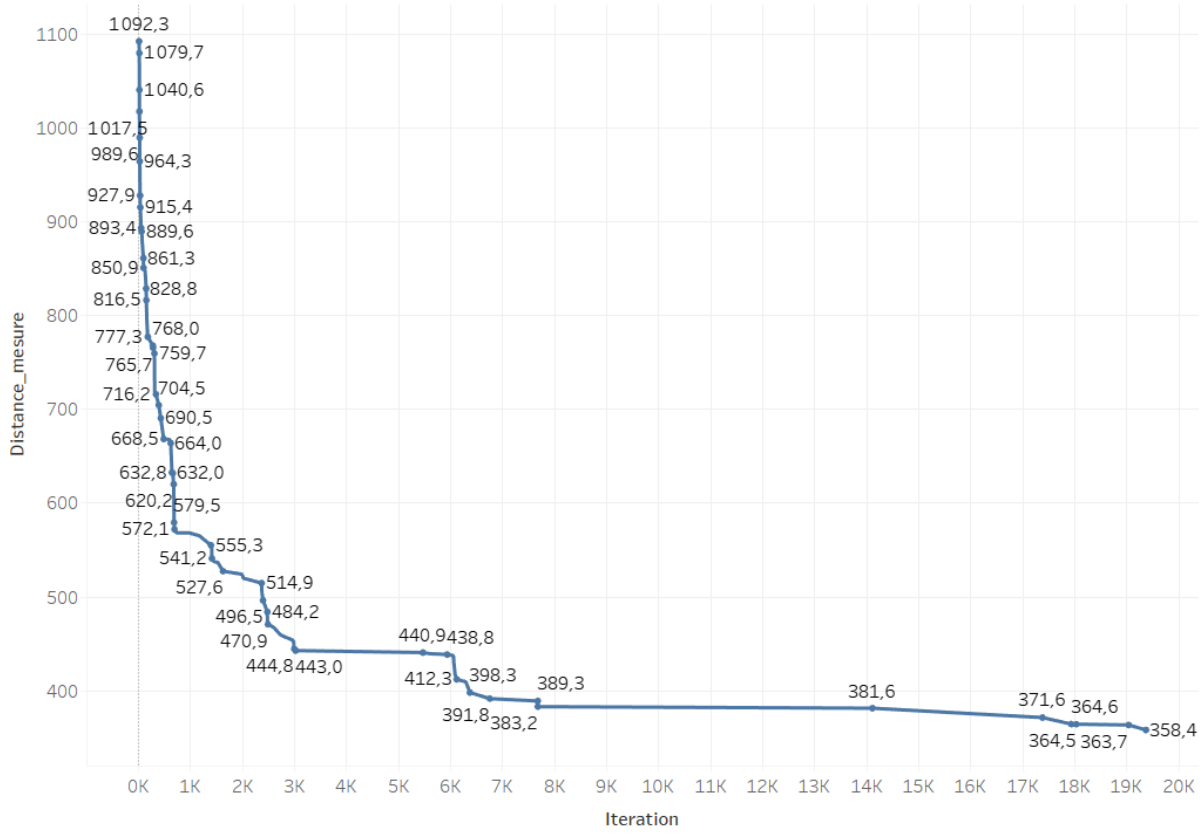
Métaheuristiques

Ensuite, nous allons vous présenter les deux métaheuristiques que nous avons implémentés lors de ce projet.

Recuit simulé

Dans un premier temps, le recuit simulé a été le plus facile à implémenter puisqu'il a suffi de créer les opérateurs en gérant leur code qu'avec de l'aléatoire, à l'inverse de la méthode Tabou où il faut générer tous les voisins. Nous avons commencé donc à créer le recuit simulé avec les 4 opérateurs suivants : Relocate Intra, relocate Extra, Exchange Intra, Exchange Extra.

Dans notre application, le recuit est la métaheuristique la plus rapide et qui converge le plus rapidement vers l'optimum global. En effet, pour tester nos différents opérateurs, nous avons compilé plusieurs fois le programme pour voir quelle était la meilleure fitness quand nous gardons les 30 premiers clients. Pour présenter les résultats, nous avons choisi d'utiliser Tableau qui est un outil pour créer des rapports et des tableaux de bords. Voici le résultat du recuit pour 30 clients :

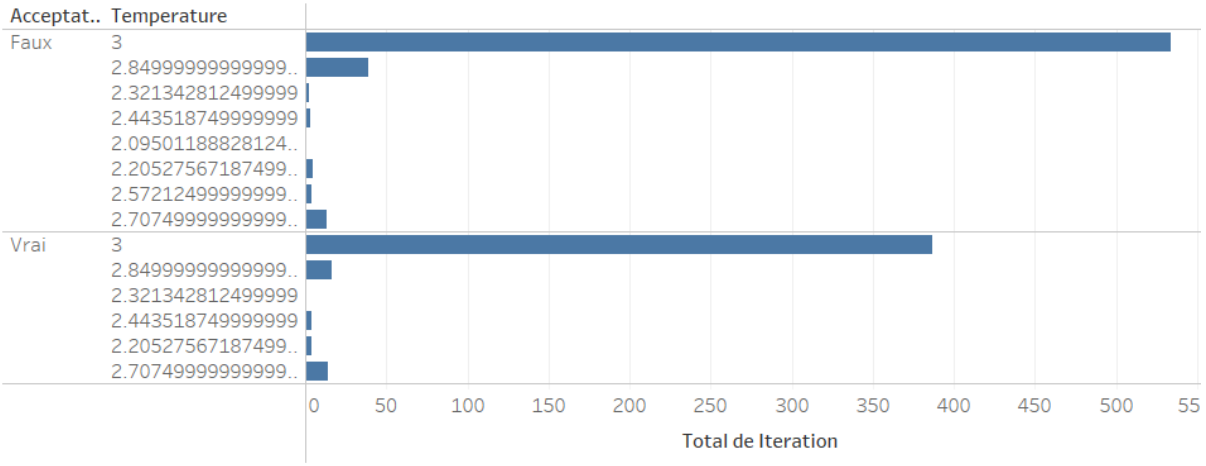


On voit que dans cette exécution-ci, la solution a atteint l'optimum global.

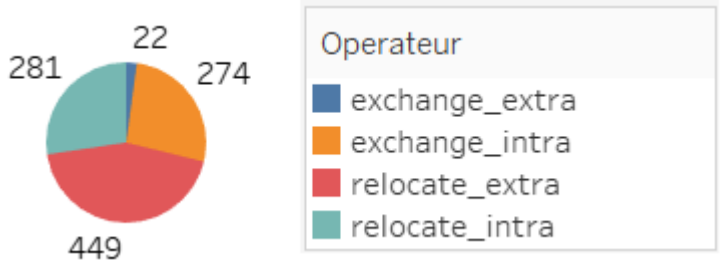
Dans cette exécution, nous avons une température à 3, un alpha à 0.95 et un N2 à 10.000. On peut voir qu'après 8.000 itérations, aucune meilleure solution n'a été trouvé avant là 14.000. Or, nous avons donc changé de température après 10.000 itérations, nous permettant ainsi d'accepter moins de solution aléatoire.

Voici le graphique qui montre le taux d'acceptation par la température d'une meilleure solution. Cela signifie, que lorsque c'est vrai, la fitness est dégradé, mais la solution est définie comme meilleure solution pour nous permettre d'explorer de nouvelles solutions.

Or, ce schéma à du sens, s'il on le compare avec la température

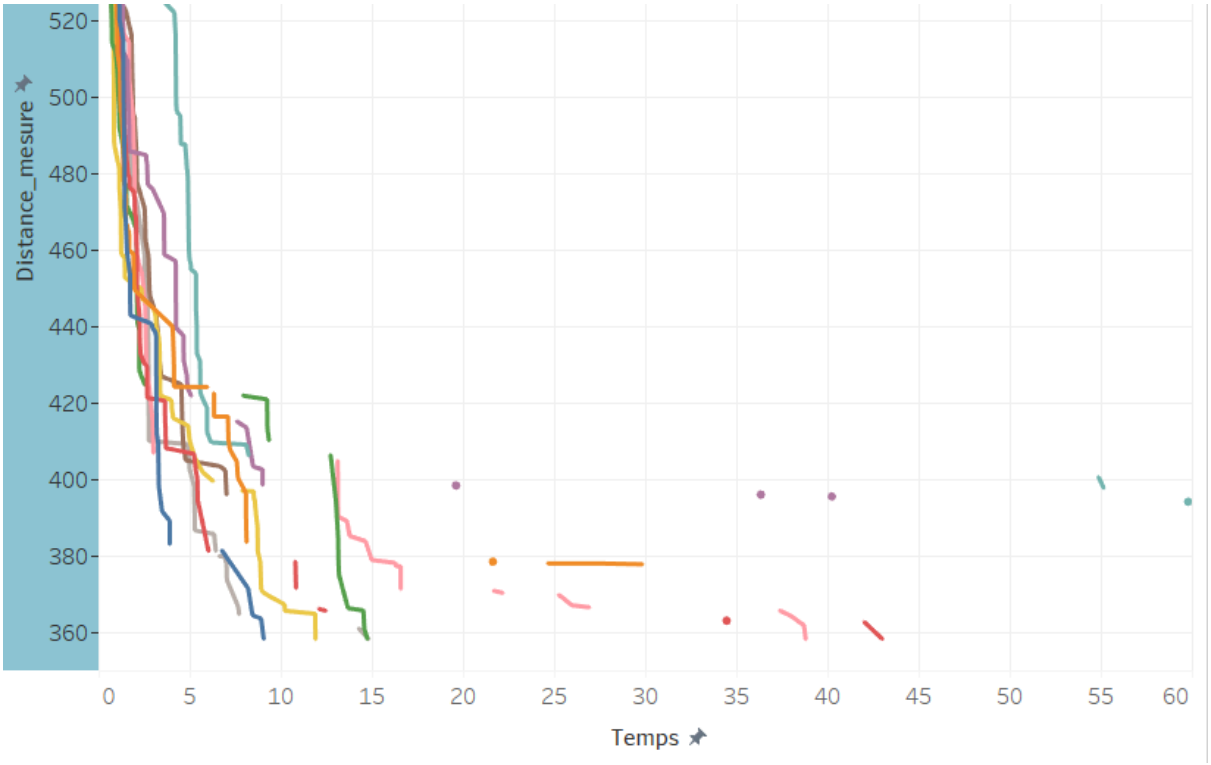


Il est difficile de conclure ici, sur le taux d'acceptation et la température car notre échantillon est un peu faible. Nous verrons plus tard la corrélation entre les 2. Maintenant, comparons les opérateurs :



On voit que l'opérateur exchange extra est très faible. Cela est dû au fait qu'il est compliqué d'échanger 2 clients entre 2 camions parce qu'ils ne respectent plus les contraintes de temps ou de poids. Ensuite, le recolate_extra est le meilleur opérateur car il permet dans un premier temps, de supprimer un camion. En effet, les 3 autres opérateurs ne permettent pas cela, donc sans le relocate extra, le résultat dépendra de la solution initiale et du nombre de

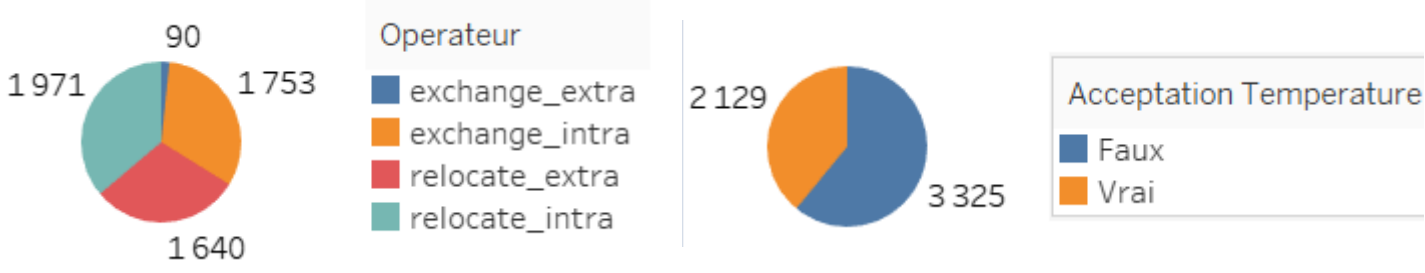
camions présent dedans. Ensuite, c'est le 2ème seul opérateur à déplacer un client dans un autre camion, or les conditions de temps et de capacité sont bien plus facile à acceptée dans ce sens qu'avec un échange.

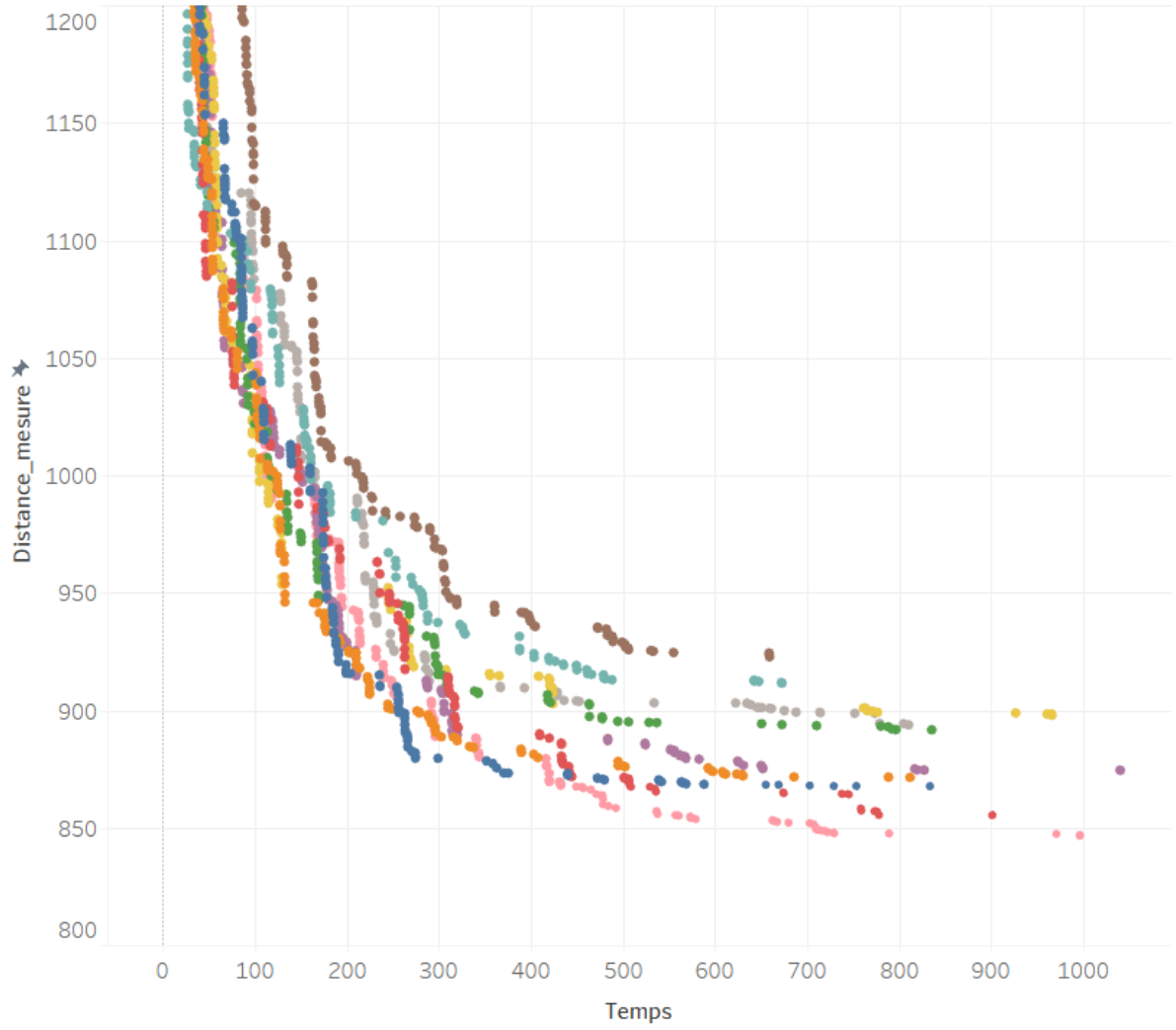


Voici, le résultat d'exécution de 10 recuit simulé avec 30 clients. On peut voir que parmi, c'est 10 exécutions, il y en a 6 qui atteignent l'optimum global. Les espaces représentent que la température a changé et qu'aucune meilleure solution n'a était trouvé. Par exemple, pour la courbe rose, on peut voir qu'il y a eu 4 changements de température avant l'obtention de l'optimum. De plus, la courbe bleue est la plus rapide à accéder à l'optimum, en 8 secondes, alors que la rouge l'atteint en 43 secondes. On peut également analyser la turquoise qui atteint la pire fitness dans le plus long temps. Cela nous prouve donc qu'il est important d'exécuter plusieurs fois les métaheuristiques pour analyse les différents résultats.

Pour 100 clients :

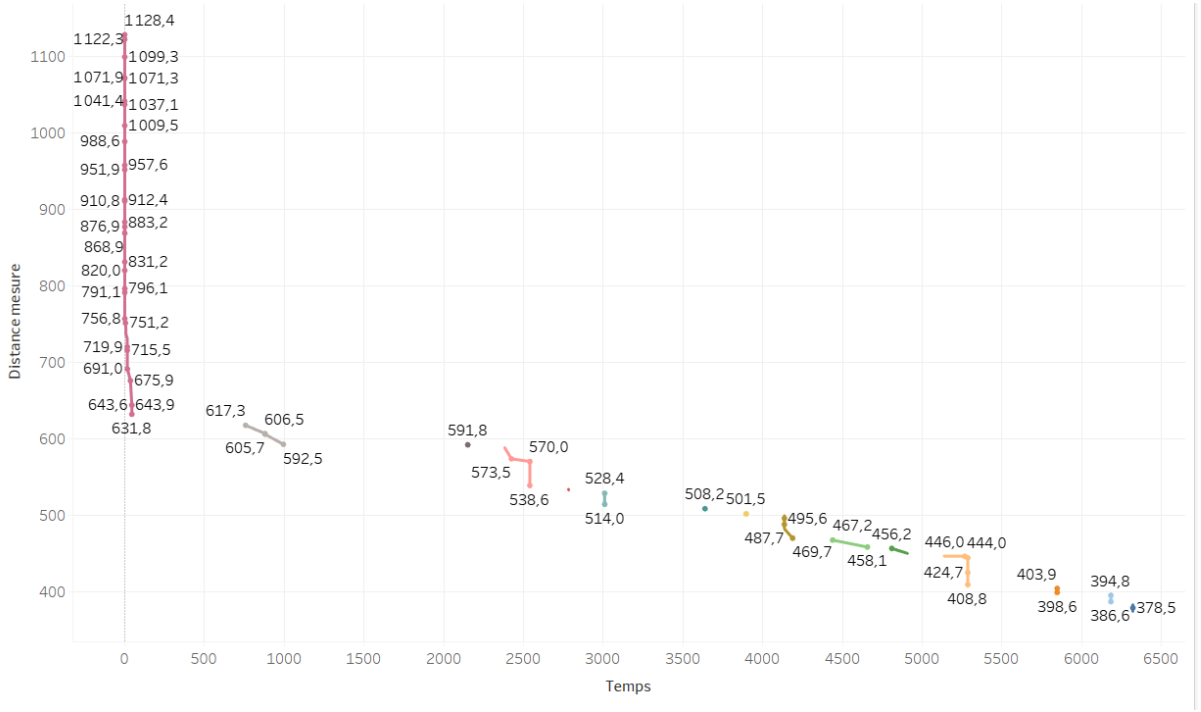
Même constant ici, les pourcentages sont identiques qu'avec 30 clients.





Voici, la fitness affiché en fonction du temps pour 10 éxecutions. Chaque point représente une amélioration de la solution. On peut constater que la meilleure solution a été atteint par la courbe rose ayant le 2ème plus grand temps d'exécution.

Voici l'exécution du recuit, 30 clients, avec une température initiale de 47 est un alpha : 0.90.

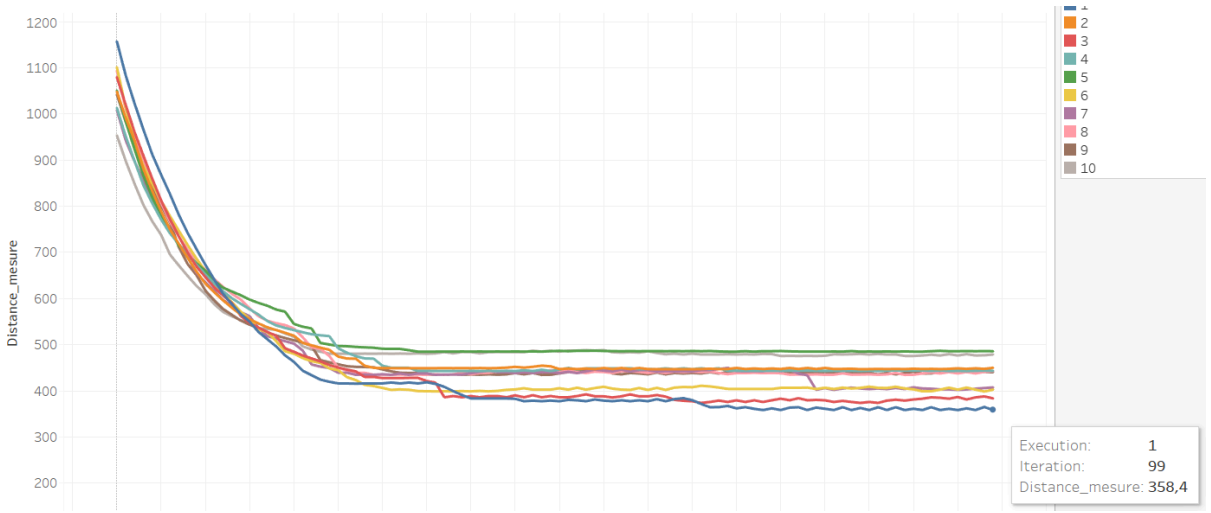


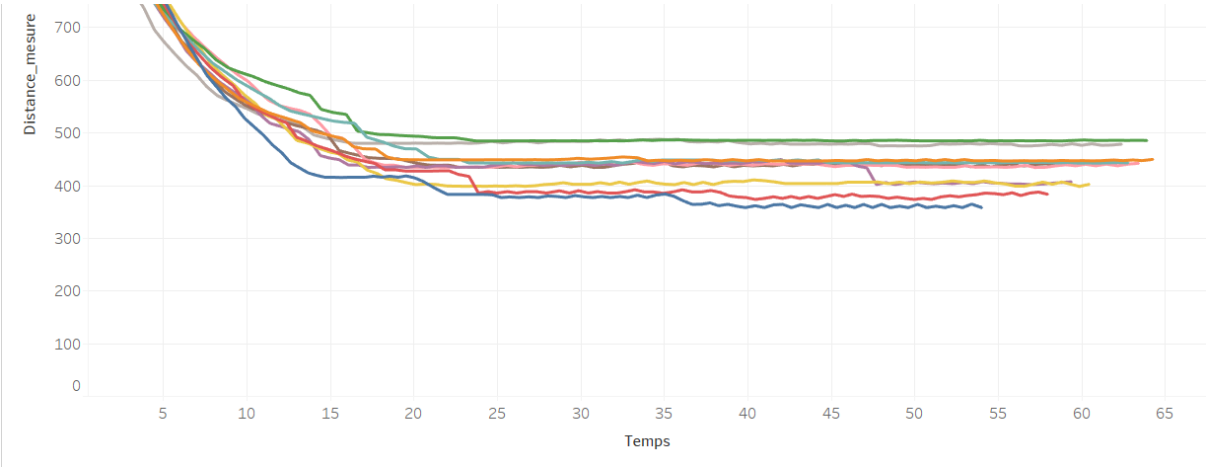
On peut voir qu'avec une température élevée, la solution reste haute au début et met longtemps avant de trouver de meilleures solutions. Après environ 40 minutes, et 8 changements de températures, le recuit explore de nouvelles meilleures solutions et permet d'en trouver plus dans des temps plus rapide. On peut en déduire qu'une température initiale haute, n'est pas optimal dans notre application.

Méthode tabou

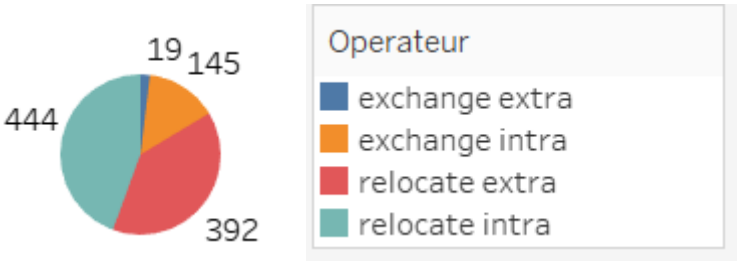
La méthode tabou a été beaucoup plus compliqué à implémenter. En effet, après avoir implémenté le recuit, il a fallu revoir tous les opérateurs car dans la méthode Tabou, il faut générer toutes les combinaisons possibles de chaque opérateur et s'assure que chaque cas est bien implémenté. Cela nous à donner du fil à retordre parce qu'il est compliqué de vérifier si tous les opérateurs fonctionnent correctement. Ensuite, il a fallu prendre en compte les opérateurs inverse du meilleur voisin généré à chaque fois. En effet, dans la méthode tabou, on met l'opérateur inverse de l'opération précédemment effectuée pour ne pas revenir à cette solution précédente. Cela a été aussi compliqué à implémenter et surtout pris beaucoup de temps.

Dans le cas avec 30 clients, on constate que la méthode tabou est moins efficace que le recuit simulé. En effet, on voit qu'une seule exécution a atteint l'optimum global. On constate également qu'après une trentaine d'itérations, la solution tourne autour de la même fitness et n'explore pas autant qu'un recuit simulé.

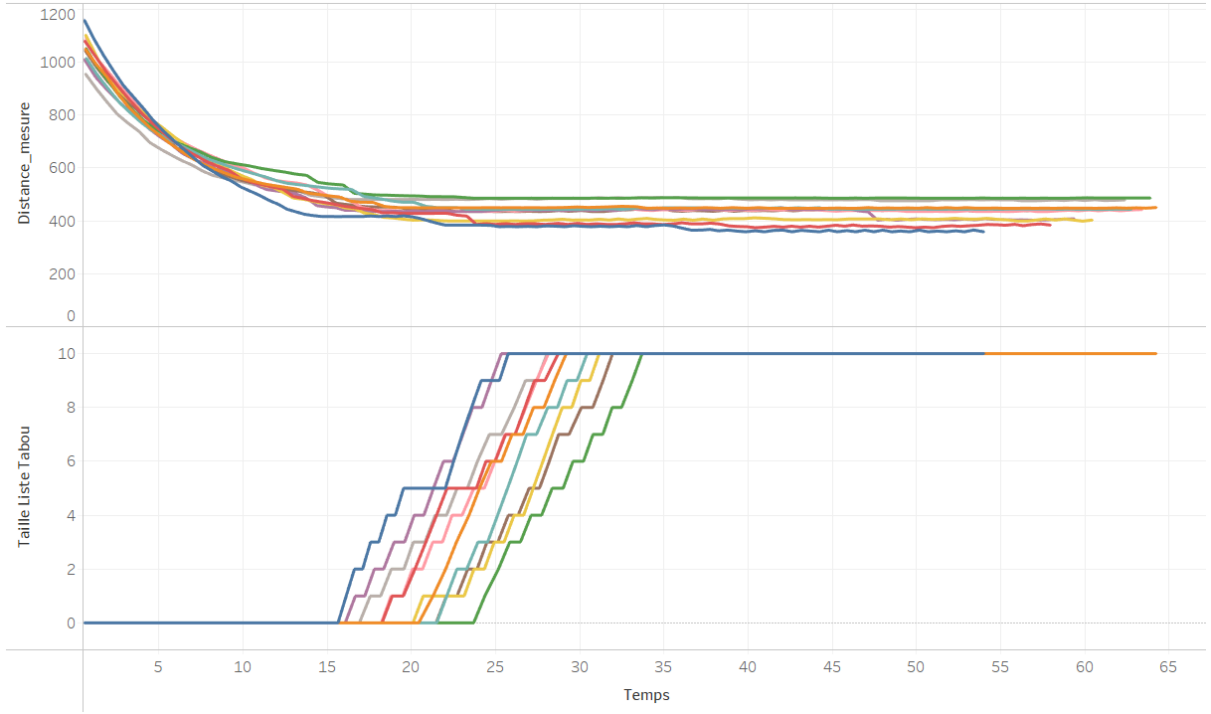




On peut également voir qu'en termes de temps, la méthode tabou est bien plus longue que le recuit simulé.



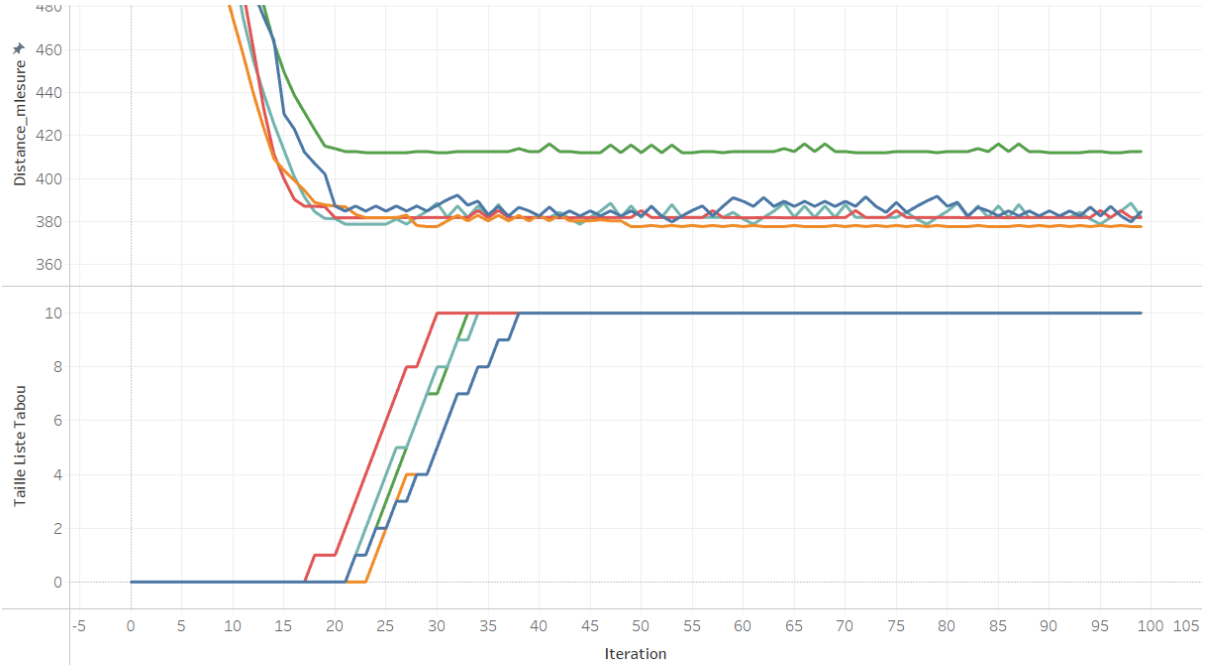
La proportion, par rapport au recuit, est un peu similaire même si on constate quand même une augmentation du relocate intra qui passe en première place.



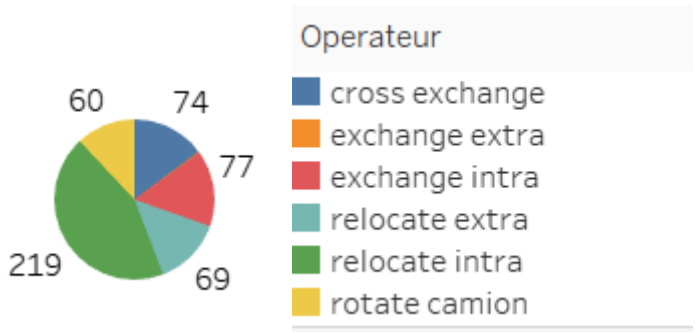
En ce qui concerne, la taille de la liste tabou, on voit qu'elle commence à être utilisée quand les fitness comment à être rectiligne. On peut conclure que la méthode de la descente est déjà très efficace. Bien que si on prend en exemple la courbe bleue, on commence à 400 de fitness pour réussir à atteindre 358 grâce à la liste Tabou.

Nous allons rajouter 2 nouveaux opérateurs :

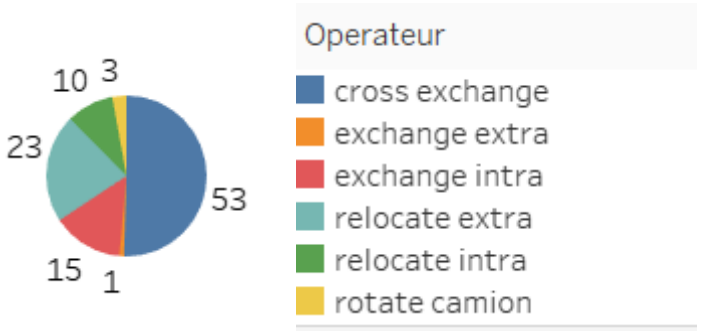
- Rotate camion, qui décale d'un client tous les clients du camion, passant ainsi les derniers en tant que premier client à visiter.
- Cross Exchange comme expliqué plus haut



On voit ici, que sur 5 exécutions, la moyenne des résultats est meilleure que sans ces 2 opérateurs. On voit également que la descente est aussi bien meilleure.

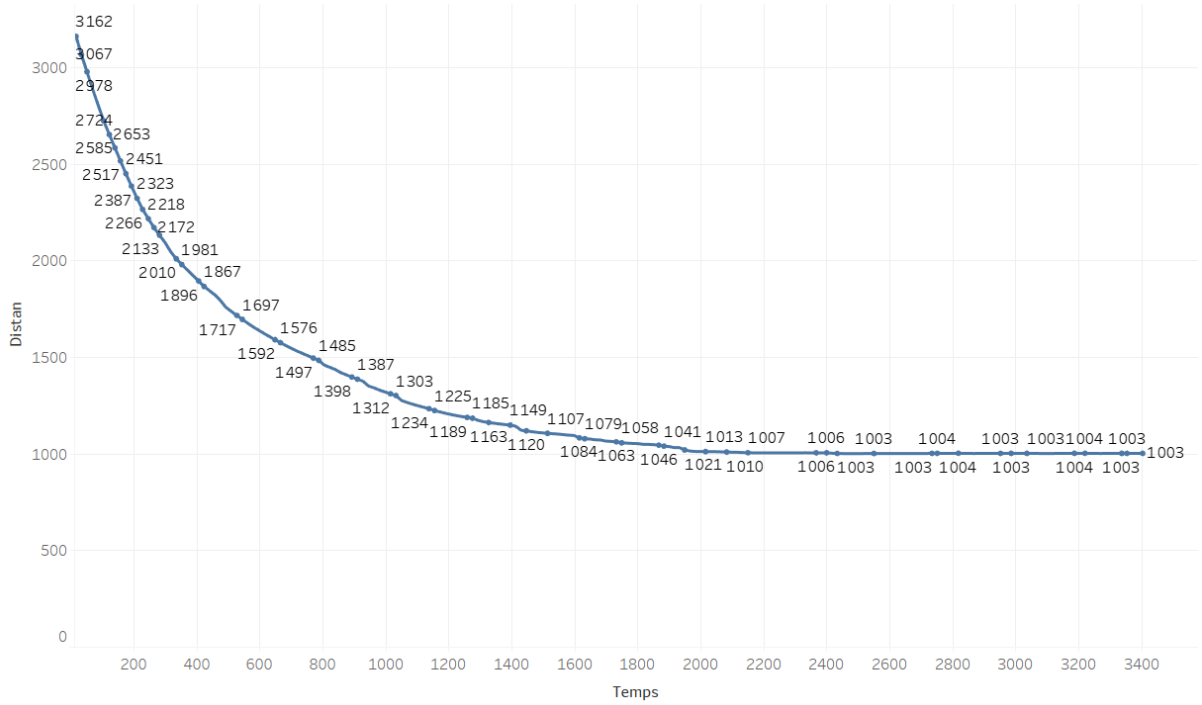


Il y a une meilleure répartition des opérateurs après le relocate intra. Cependant, ce qu'il est intéressant de voir, c'est que comme nous avons vu plus haut, nous arrivons aux alentours de l'optimum après 20 itérations, grâce à la descente, sans utiliser la liste tabou. On peut voir quelle est la répartition des opérateurs dans ces 20 itérations :



On voit que le cross exchange est donc le meilleur opérateur, ou en tout cas le plus optimal pour converger vers la moyenne des résultats. Et ensuite, c'est le relocate intra qui permet d'explorer d'autres solutions aux alentours de cette moyenne.

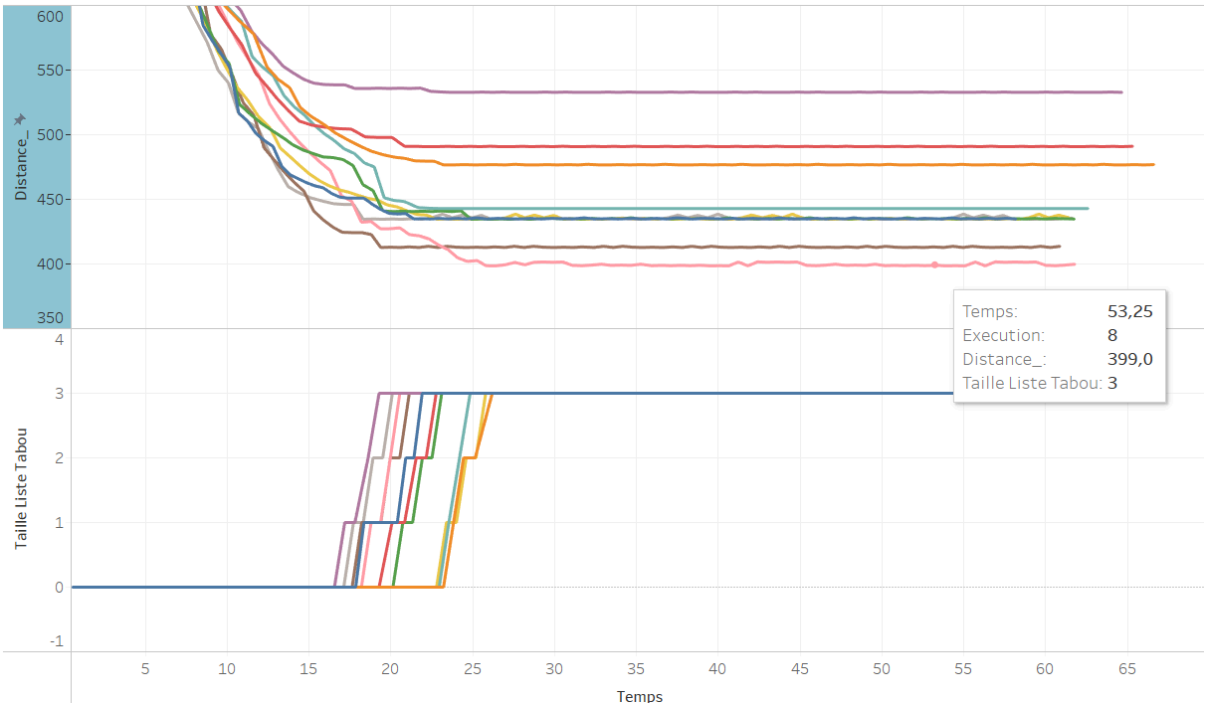
Voici l'exécution avec 100 clients :



On peut globalement voir que la qualité de la solution est beaucoup moins bien qu'un recuit simulé (847 pour la meilleure) est bien plus longue (56 minutes contre 16 minutes pour un recuit).

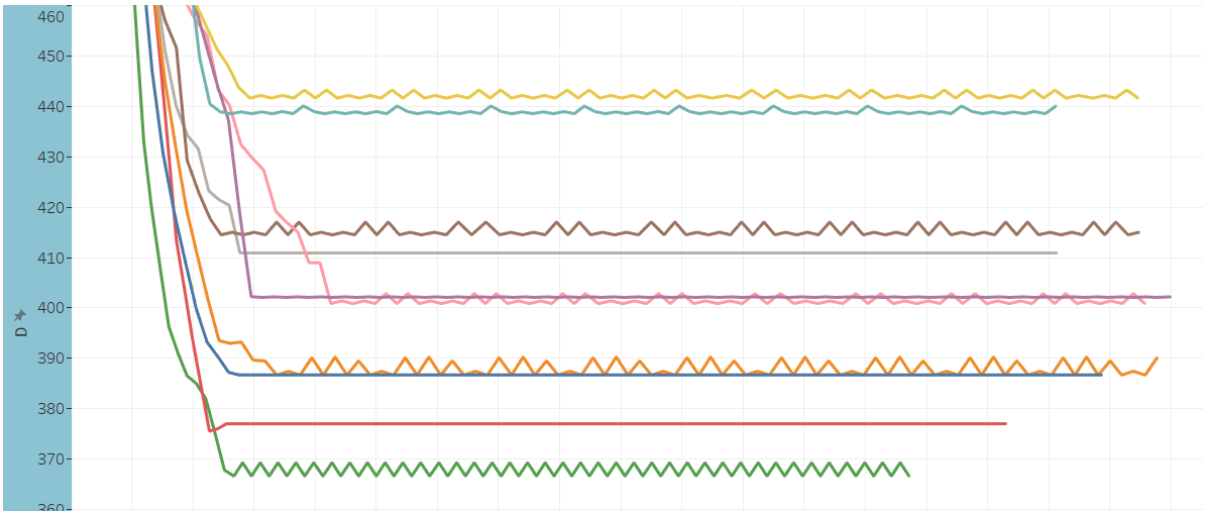
Voyons maintenant, 10 exécutions avec 30 clients et une taille de liste tabou de 3 pour comparer avec l'exécution précédente.

Sans le cross exchange :



On voit globalement, que les solutions sont moins bonnes qu'avec une taille 10. De plus, il explore très peu de solution est reste sur sa meilleure, car la taille de la liste est trop petite.

Avec le cross exchange



Même conclusion que précédemment, on peut voir que la fitness est en moyenne meilleure grâce à la descente et au cross exchange. Cependant, ce que l'on peut remarque, c'est que comme la taille est trop petite, on peut apercevoir des patterns de solutions et ce qui fait que cela boucle à l'infini sur 4 solutions, ne permettant pas ainsi d'explorer suffisamment de solution.

Conclusion

On peut conclure que le recuit simulé est la meilleure des métaheuristique dans notre application, car elle permet d'atteindre un optimum local ou global rapidement et dans beaucoup d'exécution, à l'inverse de la méthode Tabou qui n'est pas assez efficace. Le projet était très intéressant à réaliser bien qu'il a demandé énormément d'heure de travail.