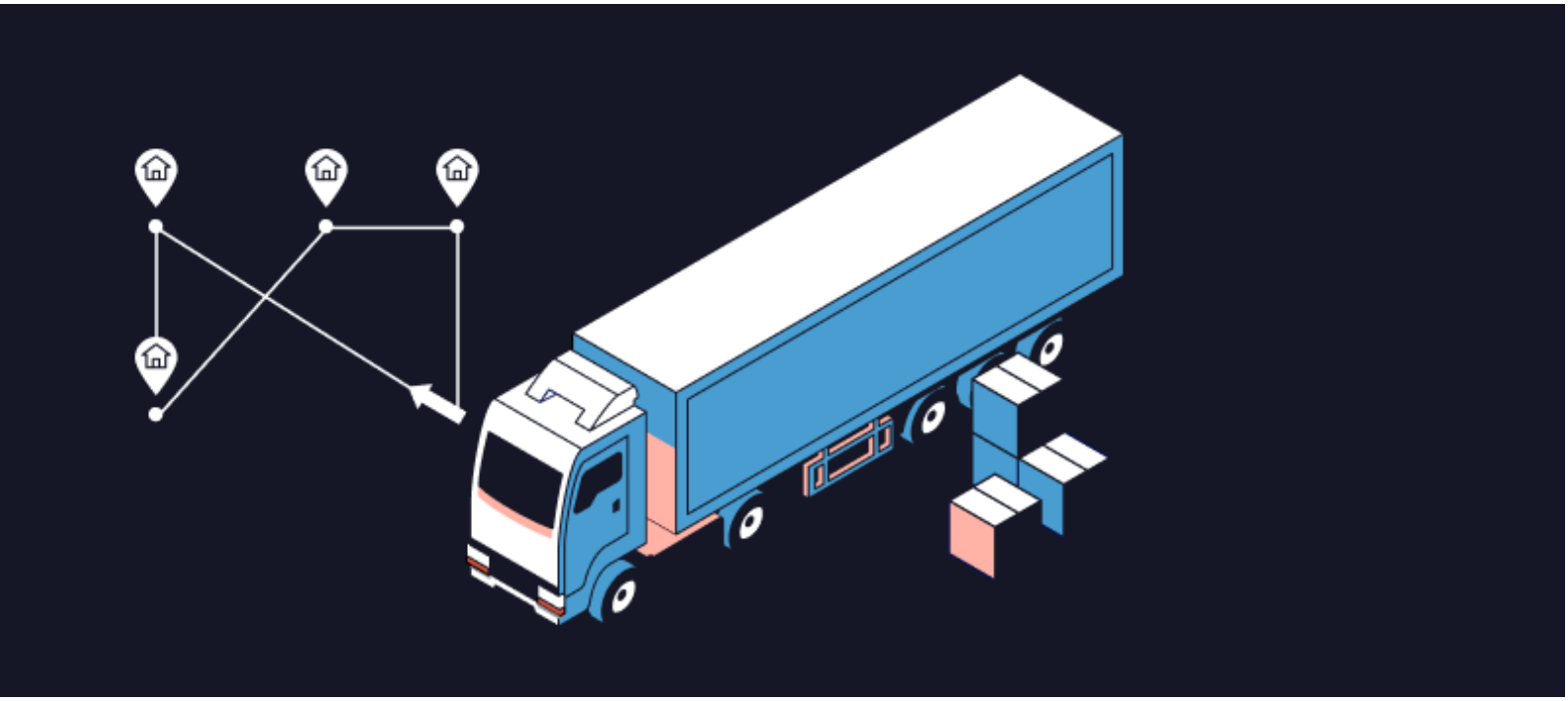
Optimisation Discrète

Rapport Projet CVRP

Année 2021 – 2022



Réalisé par :

BRAGA Jonathan
OULED MOUSSA Yanis

Responsable de l'UE :

M. BONNEVAY Stéphane

Table des matières

Jonathan BRAGA
Yanis OULED MOUSSA

Introduction

Le problème des *Capacitated Vehicle Routing Problem* (CVRP) aussi appelé VRP pour *Vehicle Routing Problem* est un problème connu dans les domaines de la recherche opérationnelle et de l'optimisation combinatoire. Le CVRP se trouve être un problème de la catégorie des problèmes dits "NP-Complet", ce qui signifie qu'il n'existe pas d'algorithme permettant d'obtenir une solution optimale dans avec une complexité en temps polynomial. Pour résoudre un tel problème, il nous faudra donc nous contenter de trouver des solutions que l'on peut qualifier de "bonne qualité".

Pour y parvenir, nous nous sommes tournés vers des méthodes approchées par des heuristiques. On appelle heuristique une méthode de calcul algorithmique permettant d'approcher rapidement une solution à un problème complexe donné. Nous avons donc développé un logiciel permettant d'appliquer 2 algorithmes de ce type afin de pouvoir résoudre ce problème : https://github.com/Yaimuu/CapacitedRoutingProblem.

Jonathan BRAGA
Yanis OULED MOUSSA

Modélisation du problème

Pour modéliser le problème du CVRP, nous avons déjà commencé par trouver quelle est le fitness qu'il faudra minimiser. Après réflexion, le fitness que nous avons décidé de modéliser est la somme des distances entre tous les clients. Elle peut se représenter par l'équation suivante :

$$F = \sum_{i=0}^{n-1} dist(Ci\ , Ci + 1)$$

Ici n est le nombre total de clients et C_i sont les différents clients.
Pour organiser la structure de notre code nous avons décidés de modéliser les classes suivantes :

- **Course**

Cette classe représente une solution au problème. Cette dernière possède une liste de Truck qui lui permet d'utiliser des algorithmes de voisinage sur ces derniers.

- **Truck**

Cette classe représente les différents camions présents dans la Course. Chaque camion possède une liste de clients qui lui permettent de connaître sa capacité ainsi que la distance qu'il doit parcourir.

- **Client**

Cette classe représente les clients de la Course. Chaque client est associé à un et un seul camion (à l'exception du dépôt). Les clients possèdent aussi chacun une quantité qui lui est associée.

Pour la gestion de l'affichage, nous avons décidé d'utiliser une structure MVC, toute la partie de l'affichage est gérée dans le package View du projet. Les différentes classes servent à mettre à jour l'affichage après l'exécution d'un algorithme.

Nombre minimum de véhicules

Pour calculer le nombre de véhicules minimum pour chaque jeu de données, on peut utiliser la formule suivante :

$$Nombre\ minimum\ de\ véhicules\ = \frac{\sum_{i=0}^{n} C_i}{100}$$

Ici n est le nombre total de clients et C_i sont les quantités des différents clients de la course. On divise la somme par la capacité maximale d'un véhicule (qui est égale à 100) pour obtenir le nombre minimal de véhicules pour une course.

Dans le tableau suivant, nous pouvons trouver le nombre minimal de véhicule pour chaque jeu de données :

Jonathan BRAGA
Yanis OULED MOUSSA

Projet CVRP

Nom du jeu de données	Nombre minimal de véhicules
A3205.txt	5
A3305.txt	5
A3306.txt	6
A3405.txt	5
A3605.txt	5
A3705.txt	5
A3706.txt	6
A3805.txt	5
A3905.txt	5
A3906.txt	6
A4406.txt	6
A4506.txt	6
A4507.txt	7
A4607.txt	7
A5307.txt	7
A5407.txt	7
A5509.txt	9
A6009.txt	9
A6109.txt	9
A6208.txt	8
A6310.txt	10
A6409.txt	9
A6509.txt	9
A6909.txt	9
A8010.txt	10

C101.txt	10
C201.txt	10
R101.txt	8

Génération de solutions de bases

Afin de pouvoir appliquer une heuristique pour résoudre le problème, il faut tout d'abord que l'algorithme parte d'une solution initiale non-optimale. Pour cela nous avons décidé de générer des solutions aléatoires à chaque fois que l'on charge un fichier. Notre génération de solution de base fonctionne de la manière suivante :

Tant que tous les clients ne sont pas desservis, nous créons un nouveau trajet que nous remplissons à un pourcentage donné de sa capacité maximale. Lorsque cette capacité de base est atteinte, nous créons un nouveau Truck et réitérons l'opération jusqu'à ce que tous les clients ne soient desservis.

Métaheuristiques

Voisinages implémentés

Les heuristiques que nous avons mis en place, se reposent notamment sur un principe de voisinage. Cela consiste à faire des modifications aléatoires entre les tournées de différentes manières. Nous avons mis en place 2 types de voisinages différents :

Voisinages Intra-Route

Les voisinages intra-route correspondent aux changements de voisins que l'on effectue dans un même trajet.

Voici la liste des voisinages intra-route que nous avons implémenté :

- **Relocate**

Ce voisinage consiste à changer l'ordre de passage d'un client dans la liste des clients.

- **Exchange**

Ce voisinage consiste à échanger 2 clients dans la liste des clients d'un camion.

- **2-opt**

Ce voisinage consiste à "casser" 2 liaisons dans la liste des clients et de reconstruire les liaisons cassées avec les autres clients.

Voisinages Inter-Route

Les voisinages inter-route correspondent aux changements de voisins que l'on effectue entre deux trajets distincts.

Voici la liste des voisinages inter-route que nous avons implémenté :

- **Cross-Exchange**

Ce voisinage consiste à changer une partie des clients d'un trajet à un autre sous la condition que le trajet dans lequel ils sont transférés ait la capacité de les desservir.

- **Exchange**

Ce voisinage consiste à échanger 2 clients faisant partie de 2 tournées différentes.

- ***Merge***

Ce voisinage consiste à fusionner 2 camions. Ce qui signifie que nous prenons tous les clients d'un camion et que nous les ajoutons au second camion, le premier camion est alors supprimé de la liste des camions de la course. Cela ne se fait que si les capacités des camions le permettent bien sûr.

- ***Add Client***

Ce voisinage consiste à ajouter un client d'un camion dans la liste des clients d'un autre camion. Le client est supprimé de la liste des clients du premier camion. Cette opération ne peut s'effectuer que si les capacités des camions le permettent.

Implémentation

Nous avons donc implémenté 2 métaheuristiques, le recuit simulé et la méthode tabou. Pour tester les métaheuristiques, nous avons définis des critères qui doivent être validés :

- Tous les clients de la course doivent être desservis.
- Les capacités des camions ne sont jamais dépassées.
- Le nombre de camions du résultat s'approche du nombre minimal de camions pour ce jeu de données. Il est par conséquent inférieur aux nombres de camions de la solution générés aléatoirement au début.

Recuit

Pour implémenter la méthode du recuit simulé, nous avons créé une classe qui correspondra à cette méthode. La classe se trouve dans le package Metaheuristics et se nomme RecuitSimule.

La méthode du recuit simulé se déroule en plusieurs étapes :

- En paramètres sont passés la solution aléatoire ainsi que la température t0, mais aussi les paramètres n_1 et n_2.
- Nous effectuons une copie de la solution aléatoire et nous stockons le fitness de cette solution dans une variable qui correspond à la fitness minimum. Nous définissons aussi certaines variables comme, nous définissons t_k à t_0.
- Nous effectuons ensuite deux boucles for imbriqués, la première allant de 1 à n_1 et la seconde allant de 0 à n_2.
- Dans cette boucle, nous choisissons d'utiliser une méthode de voisinage au hasard parmi la liste des méthodes de voisinages implémentés pour le recuit simulé. Les paramètres pris pour effectuer l'opération de voisinage sont eux aussi aléatoires.

- Nous récupérons ensuite la fitness de ce voisinage. Et si cette solution est meilleure, alors nous prenons cette solution pour la solution actuelle et mettons à jour la fitness minimum.
- Dans le cas où la solution est moins bonne, il y a quand même une probabilité de la choisir pour la solution actuelle, cette probabilité est définie par l'équation suivante :

$$P \leq e^{\frac{\Delta f}{t_k}}$$

Ici, P est une valeur aléatoire entre 0 et 1, Δf est la différence entre la fitness de la solution actuelle et celle du voisinage aléatoire. t_k, possède sa valeur actuelle.

- Nous mettons ensuite à jour la valeur de t_k en la multipliant par μ.
- Une fois que les deux boucles for sont terminées, nous retournons le résultat de l'exécution donc la fitness et le nombre de camions de la solution. Nous faisons aussi un affichage graphique de la solution.

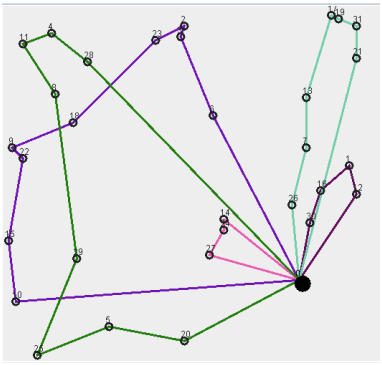
Ci-dessous les résultats de l'exécution de notre méthode du recuit simulé sur différents jeu de tests :

Pour les tests sur le recuit simulé, toutes les méthodes de voisinages ont été utilisées en même temps. Ici les valeurs sont les suivantes : n_1 vaut 100, n_2 vaut 100, t_0 vaut 300 et vaut 0.99.

- **Fichier A3205.txt**

Nombre de camions finaux : 5

Fitness obtenue : 820



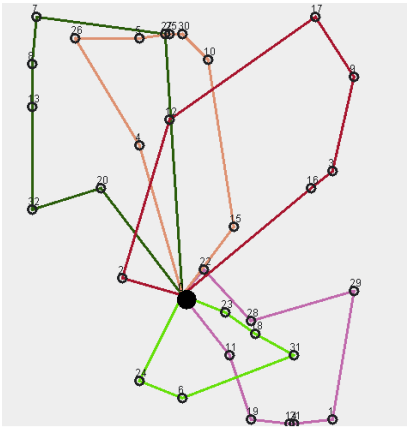
- **Fichier A3305.txt**

Nombre de camions finaux : 5

Fitness obtenue : 732

Jonathan BRAGA
Yanis OULED MOUSSA

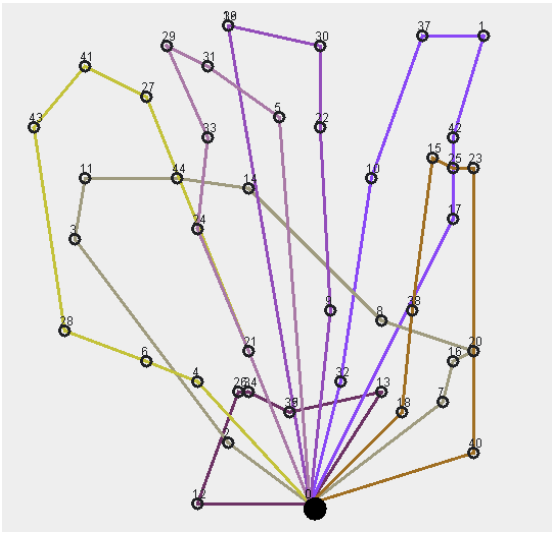
Projet CVRP



- **Fichier A4507.txt**

Nombre de camions finaux : 7

Fitness obtenue : 1298



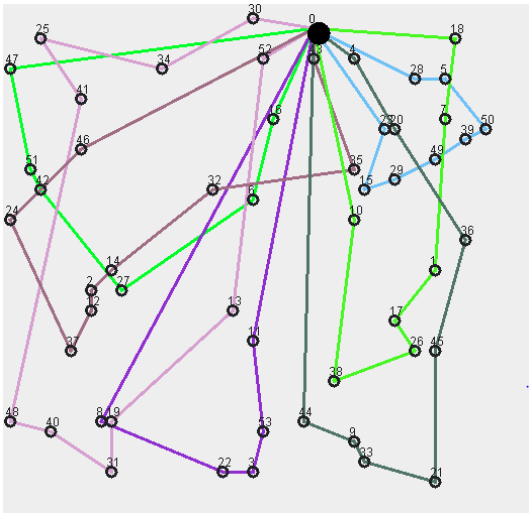
- **Fichier A5407.txt**

Nombre de camions finaux : 7

Fitness obtenue : 1328

Jonathan BRAGA
Yanis OULED MOUSSA

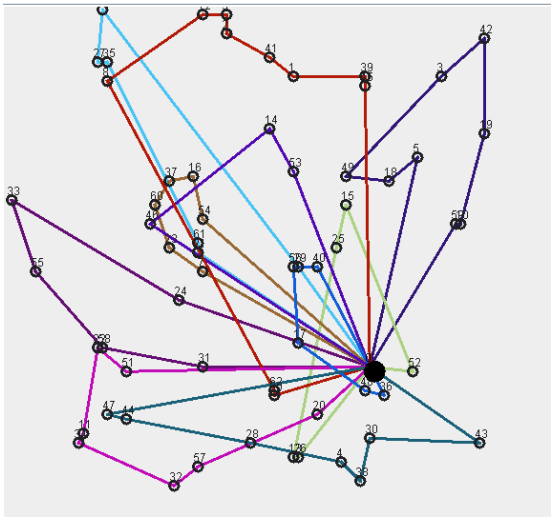
Projet CVRP



- **Fichier A6310.txt**

Nombre de camions finaux : 10

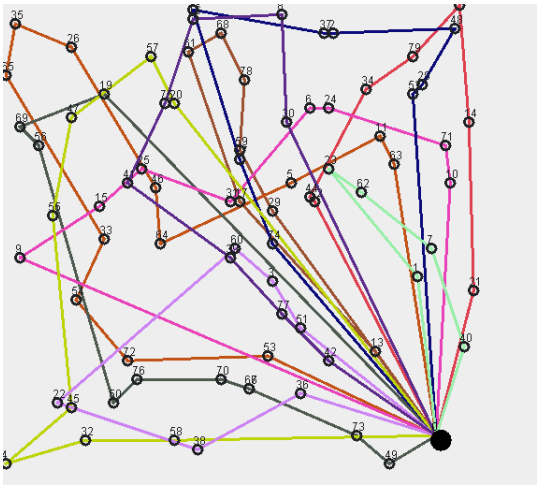
Fitness obtenue : 1533



- **Fichier A8010.txt**

Nombre de camions finaux : 10

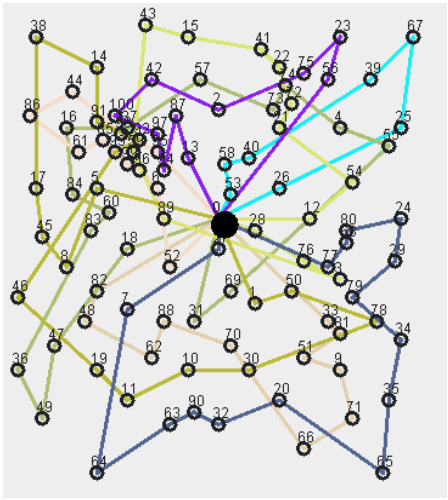
Fitness obtenue : 2360



- **Fichier r101.txt**

Nombre de camions finaux : 8

Fitness obtenue : 1226



En observant ces résultats, nous pouvons constater que lors de ces tests, notre méthode de recuit simulé obtient des résultats proches des minimums. En effet, le nombre de camions des solutions obtenus s'approche des camions minimum pour chacun de ces fichiers. De plus, graphiquement nous pouvons voir que cette métaheuristique arrive à regrouper les clients dans les différents pour faire des trajets de camions qui semblent cohérents.

En revanche, nous n'avons pas d'indication sur la validité de la fitness finale.

Tabou

Pour implémenter la méthode de recherche tabou, nous avons créé une classe qui correspond à cette méthode. Elle se situe dans le package Metaheuristics et elle se nomme TabouMethod.

La méthode de la recherche Tabou s'effectue en plusieurs étapes :

- En paramètre est passé le nombre d'itérations à effectuer pour l'algorithme, ainsi que la solution aléatoire.
- Nous définissons ensuite les variables de fitness minimal et de solution minimale et actuelle à partir de la solution aléatoire.
- Nous définissons ensuite la liste Tabou qui sera vide au début de l'exécution.
- Nous effectuons ensuite une boucle for allant de 0 au nombre d'itérations passés en paramètres.
- Dans cette boucle, nous obtenons tout d'abord la meilleure modification pour chaque opération de voisinage. C'est-à-dire la modification qui réduira au mieux la fitness pour cet algorithme de voisinage.
- Dans la liste de ces résultats, on récupère ensuite la modification qui réduira au mieux la fitness parmi toutes les méthodes de voisinages.
- On effectue la modification choisie et on récupère la fitness de ce voisin.
- Si cette solution est meilleure que la solution minimale, alors on modifie les variables de fitness minimale ainsi que la meilleure solution.
- Si cette solution est moins bonne que la solution actuelle, alors on ajoute la modification dans la liste tabou pour ne pas pouvoir retourner en arrière.
- On modifie alors la solution actuelle par le voisin sélectionné.
- Nous retournons alors la fitness finale ainsi que le nombre de camions de la solution. Nous affichons aussi graphiquement la solution.

Ci-dessous les résultats de l'exécution de notre méthode de recherche Tabou sur certains jeux de données :

Les tests ont été effectués avec les méthodes de voisinages suivantes :

- Fusion
- Exchange - Intra
- Exchange - Inter

Les valeurs suivantes ont été utilisés pour les tests :

- Le nombre d'itérations est de 1000

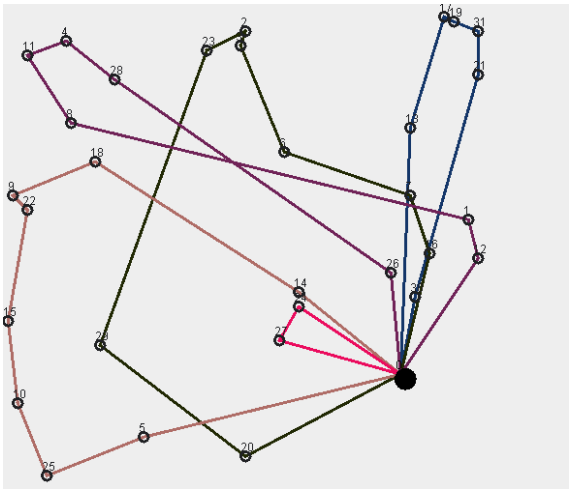
Jonathan BRAGA
Yanis OULED MOUSSA

Projet CVRP

- **Fichier A3205.txt**

Nombre de camions finaux : 5

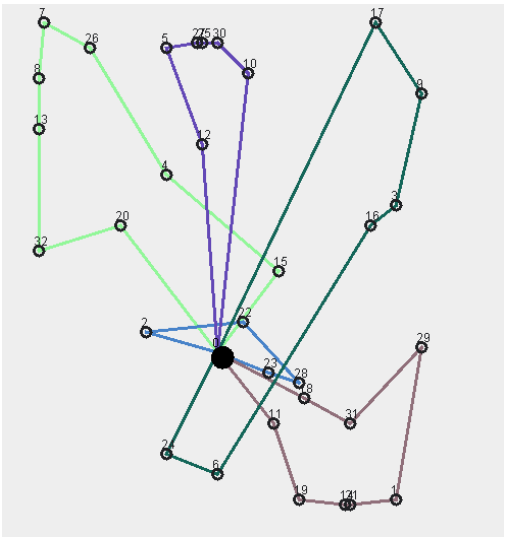
Fitness obtenue : 922



- **Fichier 3305.txt**

Nombre de camions finaux : 5

Fitness obtenue : 714



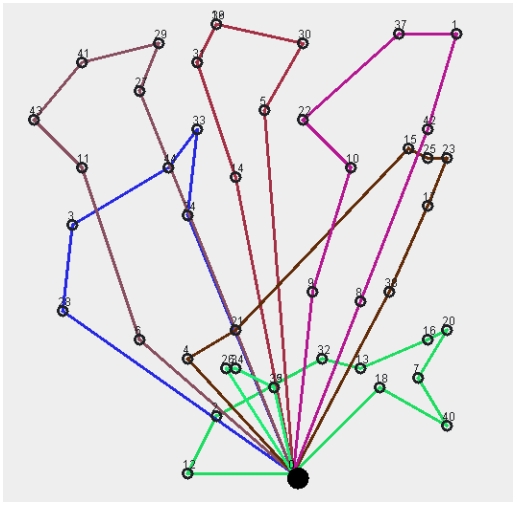
Fichier A4507.txt

Nombre de camions finaux : 7

Fitness obtenue : 1202

Jonathan BRAGA
Yanis OULED MOUSSA

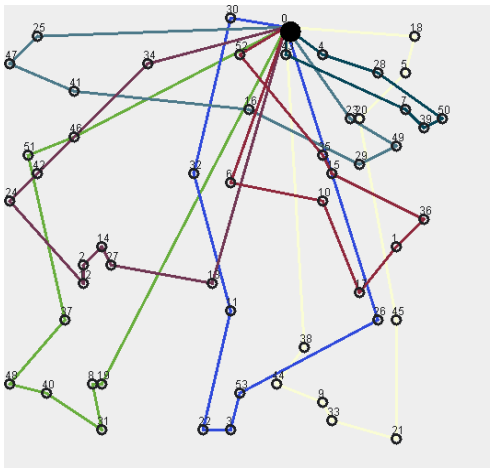
Projet CVRP



Fichier A5407.txt

Nombre de camions finaux : 7

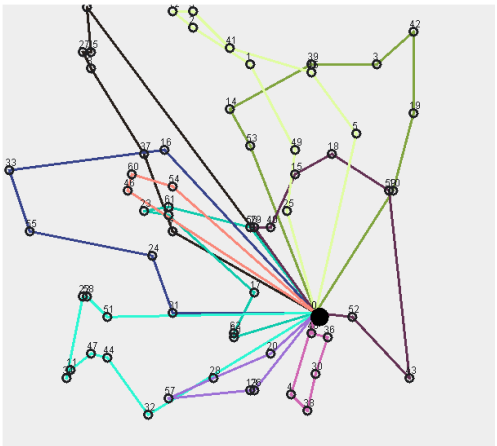
Fitness obtenue : 1299



- **Fichier A6310.txt**

Nombre de camions finaux : 10

Fitness obtenue : 1413



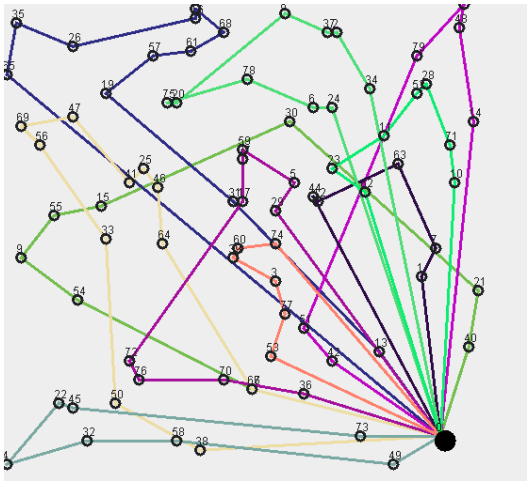
Jonathan BRAGA
Yanis OULED MOUSSA

Projet CVRP

- **Fichier A8010.txt**

Nombre de camions finaux : 10

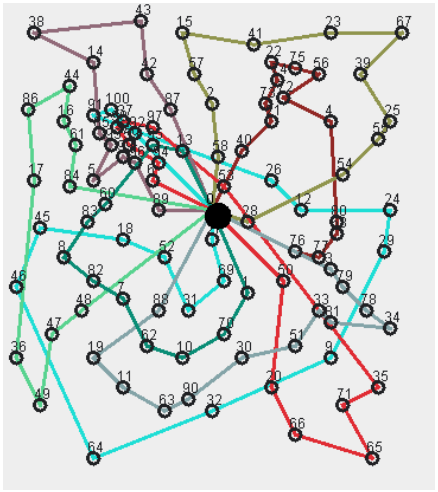
Fitness obtenue : 2075



- **Fichier r101.txt**

Nombre de camions finaux : 8

Fitness obtenue : 1079



En observant les résultats obtenus sur ces différents jeux de tests, on peut constater que cette méthode améliore grandement la solution. En effet, le nombre de camions finaux s'approche du nombre de camions minimum pour chaque jeux de tests. On peut aussi voir graphiquement que la méthode regroupe bien les clients pour faire des trajets cohérents comme dans la méthode du recuit simulé.

Nous n'avons pas d'information pour valider la fitness finale non plus, mais elle est grandement inférieure à la fitness de la solution générée aléatoirement au début de l'exécution de l'algorithme.

Jonathan BRAGA
Yanis OULED MOUSSA

Il est possible de faire une comparaison des résultats entre la méthode de recuit simulé et la méthode tabou. En effet les deux méthodes améliorent grandement la solution mais nous pouvons constater des différences. Dans la majorité des cas, la méthode tabou retourne un meilleur résultat que la méthode du recuit simulé. Ce constat semble logique car la méthode tabou cherche à avoir le meilleur des résultats des différents voisinages tandis que la méthode du recuit simulé en choisit un au hasard. En revanche, il est possible de voir des exceptions, notamment pour le fichier A3205.txt. Dans ce dernier la méthode du recuit simulé retourne un meilleur résultat que la méthode tabou. Les résultats dans les fichiers avec un nombre de clients peu élevé (entre 30 et 50) les méthodes du recuit simulé et du tabou ont des résultats très similaires même si dans la majorité la méthode tabou retourne une meilleure fitness. Cependant ce n'est pas le cas pour les fichiers avec un grand nombre de clients (entre 80 et 100). Dans ces derniers, la méthode tabou retourne un bien meilleur résultat que le recuit simulé. On pourrait donc en déduire que la méthode du tabou est beaucoup plus efficace que la méthode du recuit simulé sur des fichiers lourds.

Jonathan BRAGA
Yanis OULED MOUSSA

Analyse Comparative

Afin de comparer les deux métaheuristiques implémentées, nous avons décidé de les appliquer 10 fois sur chaque fichier de test afin d'en ressortir un certain nombre de données qui nous permettront de comparer ces 2 méthodes sur différents aspects. Pour chaque métaheuristique, les paramètres utilisés sont spécifiés. Il est à noter que les résultats obtenus peuvent également varier en fonction de la valeur des différents paramètres.

Ici, à partir des différents paramètres choisis, nous avons décidé de comparer 3 aspects des résultats des algorithmes :

- La valeur moyenne de la fitness,

Permet d'évaluer la qualité des résultats,

- Le nombre de camions moyens trouvés

Permet d'évaluer la constance des résultats

- Le temps d'exécution moyen

Permet d'évaluer la rapidité avec laquelle on peut s'attendre à obtenir un résultat convaincant

Jonathan BRAGA
Yanis OULED MOUSSA

Projet CVRP

Résultats obtenus

Recuit Simulé

Paramètres

Température initiale	***N1***	***N2***	***Mu***
300	100	100	0.99

Fichier	Fitness de départ	Fitness finale	Nb Trucks	Temps d'exécution (en ms)
A3205.txt	3088,00	924,70	5,00	35,02
A3305.txt	2354,00	742,50	5,00	24,59
A3306.txt	2327,00	819,60	6,00	22,37
A3405.txt	2737,00	869,40	5,00	21,12
A3605.txt	2998,00	946,50	5,00	22,65
A3705.txt	2282,00	780,90	5,00	22,59
A3706.txt	2978,00	1114,80	6,40	24,62
A3805.txt	2559,00	850,40	5,50	23,12
A3905.txt	2928,00	1026,90	5,20	24,09
A3906.txt	2966,00	1032,60	6,00	24,62
A4406.txt	3684,00	1153,20	6,20	29,38
A4506.txt	3626,00	1166,00	7,00	26,72
A4507.txt	4138,00	1343,00	7,00	27,36
A4607.txt	3419,00	1119,50	7,00	28,04
A5307.txt	4125,00	1307,70	7,70	32,06
A5407.txt	4743,00	1457,20	7,40	31,25
A5509.txt	3969,00	1351,50	9,40	32,28
A6009.txt	5352,00	1737,20	9,10	35,12
A6109.txt	3759,00	1337,40	10,00	42,72
A6208.txt	5470,00	1760,10	8,10	32,95
A6310.txt	4767,00	1688,00	10,60	36,20
A6409.txt	5640,00	1853,90	9,40	37,97
A6509.txt	4963,00	1568,00	10,10	36,47
A6909.txt	4716,00	1612,20	9,60	38,22
A8010.txt	8249,00	2484,60	10,20	42,32
c101.txt	4751,00	1617,70	10,00	52,08
c201.txt	4748,00	1609,90	10,40	52,49
r101.txt	3973,00	1375,00	8,30	51,26

Jonathan BRAGA
Yanis OULED MOUSSA

Projet CVRP

Tabou

Paramètres

Nombre d'itérations

1000

Fichier	Fitness de départ	Fitness finale	Nb Trucks	Temps d'exécution (en ms)
A3205.txt	3090,00	970,00	5,00	397,27
A3305.txt	2396,00	703,00	5,00	419,50
A3306.txt	2172,00	789,00	6,00	336,04
A3405.txt	2668,00	866,00	5,00	362,35
A3605.txt	2990,00	853,00	5,00	600,94
A3705.txt	2380,00	850,00	5,00	527,54
A3706.txt	3178,00	1074,00	7,00	538,94
A3805.txt	2526,00	816,00	6,00	642,53
A3905.txt	2817,00	935,00	5,00	421,73
A3906.txt	2896,00	892,00	6,00	628,40
A4406.txt	3475,00	1079,00	6,00	740,25
A4506.txt	3543,00	1065,00	8,00	1215,49
A4507.txt	4171,00	1289,00	7,00	906,80
A4607.txt	3423,00	1146,00	7,00	761,05
A5307.txt	4136,00	1177,00	8,00	1603,12
A5407.txt	4323,00	1390,00	8,00	1793,81
A5509.txt	3840,00	1176,00	10,00	1764,12
A6009.txt	5119,00	1523,00	9,00	1786,80
A6109.txt	3732,00	1162,00	11,00	2645,67
A6208.txt	5279,00	1593,00	8,00	2026,16
A6310.txt	4969,00	1475,00	10,00	2259,95
A6409.txt	5619,00	1756,00	9,00	2057,55
A6509.txt	4747,00	1446,00	10,00	2743,11
A6909.txt	4621,00	1429,00	10,00	3302,99
A8010.txt	8032,00	2129,00	10,00	3273,01
c101.txt	4782,00	1011,00	10,00	11642,20
c201.txt	5068,00	1098,00	10,00	10979,53
r101.txt	4060,00	1002,00	8,00	11587,73

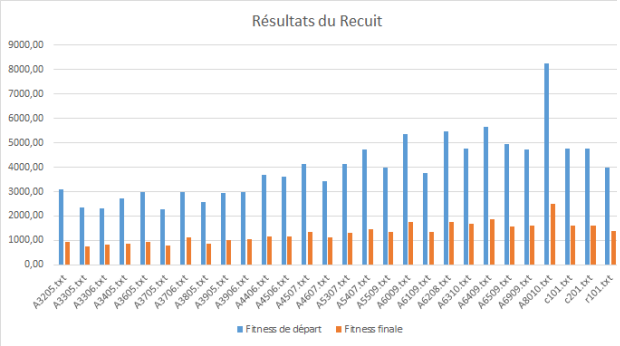
Jonathan BRAGA
Yanis OULED MOUSSA

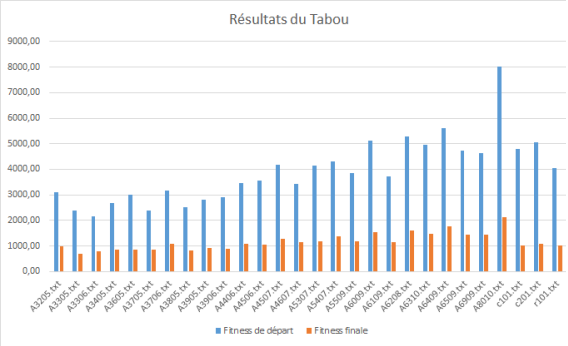
Projet CVRP

Analyse

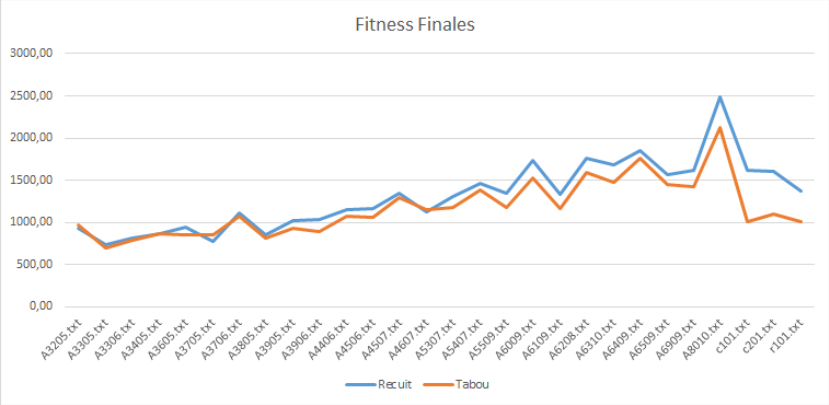
A l'aide des résultats obtenus, il a été possible d'analyser les données grâce à différents graphes :

Les premiers graphes réalisés sont une comparaison des solutions trouvés par rapport aux solutions de départ de chaque fichier.



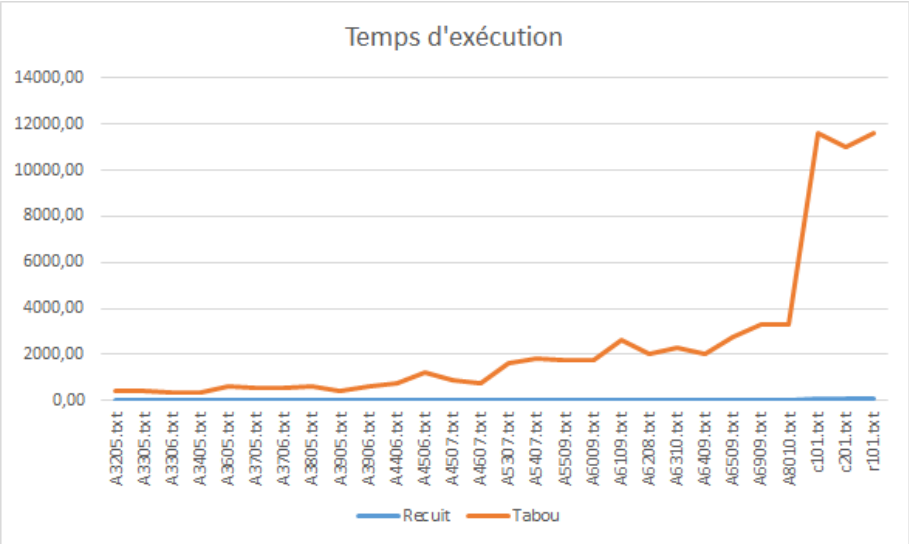


Après cela, nous avons comparé les fitness finales obtenue par les 2 méthodes grâce à ce graphique :



Les résultats obtenus nous montrent plusieurs choses concernant les algorithmes Tabou et Recuit Simulé. Tout d'abord, concernant la qualité des résultats, on observe que plus il y a de clients plus la fitness de départ et d'arrivée est élevée, comme on peut s'y attendre. L'algorithme de recuit possède une fitness moyenne plus élevée que ce que l'on peut obtenir du Tabou, qui lui possède une fitness moyenne plus basse.

Ensuite, on remarque que contrairement à Tabou, le Recuit présente des valeurs pour le nombre de camions généré par la solution finale non entière. Cela témoigne d'une inconstance des résultats bien que leur valeurs restent proches. On voit donc bien que les résultats obtenus par Tabou, sont ici, plus fiables que ceux obtenus par Recuit.



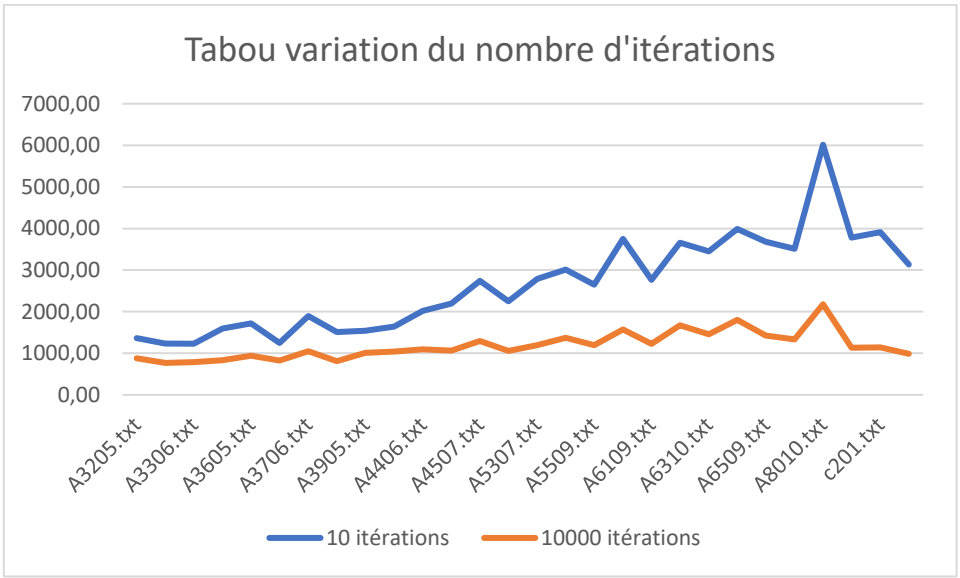
En termes de temps d'exécution, cependant, on observe que Recuit est bien plus rapide que Tabou. Cela est dû au fait que l'algorithme Tabou calcule pour chaque itération, toutes les possibilités de voisinages et choisit la meilleure.

Etude des paramètre

Les paramètres des algorithmes devraient également avoir une influence sur les résultats finaux. Nous allons donc vérifier cela expérimentalement en opposant les résultats obtenus à partir de paramètres avec des valeurs très éloignées afin de voir l'influence que peut avoir chaque paramètre.

Tabou, variation du nombre d'itérations

Le graphique suivant, a été produit en faisant tourner l'algorithme Tabou avec modification du paramètre du nombre d'itérations avec 10 et 10000 itérations. Ces nombres devraient permettre de se rendre compte d'une différence d'efficacité de l'algorithme. En effet, on se rend bien compte de la différence sur le graphique puisque pour chaque fichier, l'exécution avec 10000 itérations se trouve bien plus efficace. Plus le fichier de test a un nombre important de clients et plus la différence est élevée.



Jonathan BRAGA
Yanis OULED MOUSSA

Projet CVRP

Recuit, variation de la température t_0

Le graphique suivant nous montre l'influence de t_0 sur la fitness finale de l'algorithme Recuit. Il nous montre clairement que le paramètre t_0 seul n'a pas d'influence particulière sur le résultat de l'exécution puisque qu'il soit très faible ou très élevé, les valeurs moyennes des fitness pour chaque fichier, sont identiques.