Optimisation Discrète

Rapport de Programmation

Lucas AUPOIL & Ewald JANIN

Introduction

Dans le cadre du cours d'Optimisation Discrète, nous avons dû réaliser une une application pour résoudre le *Capacited Vehicle Problem* en utilisant des méthodes à base de population.

Réalisation du travail demandé

Modélisation

Pour ce projet, nous avons repris la structure même de notre code utilisé pour les méthodes à base de voisinage, nous avons dû quelques légères modifications et ajouter le package genetics. Ce package contient notre class utilisée pour le crossover et notre class pour effectuer notre algorithme génétique.

Algorithme génétique

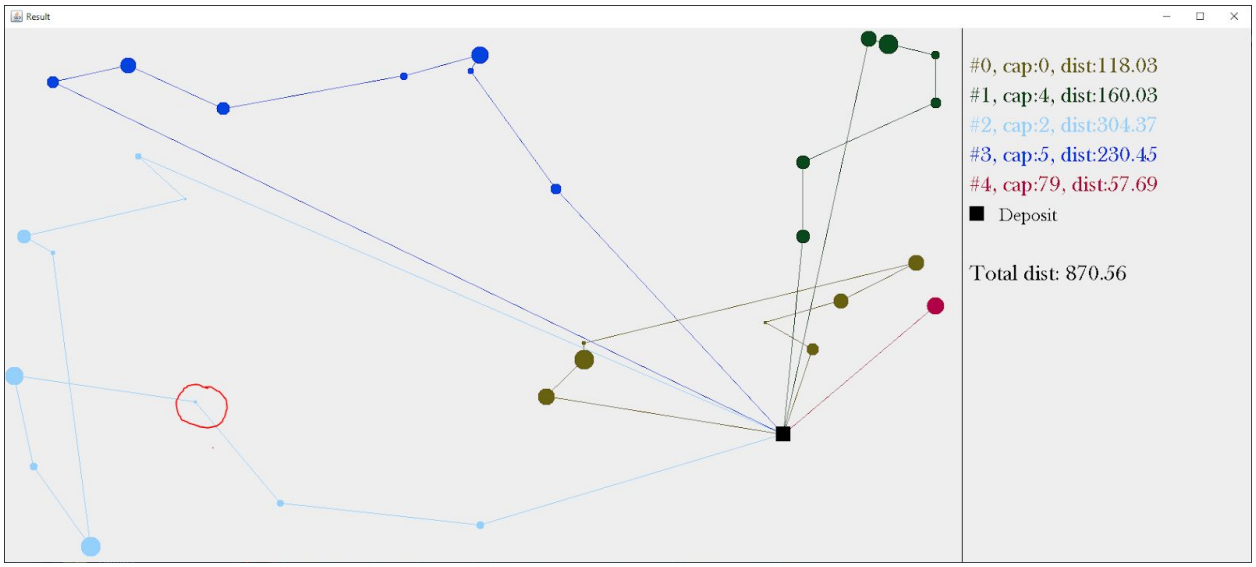
Sélection : *Roulette*

Pour la reproduction et la sélection, nous avons choisi d'utiliser la méthode de Roulette biaisée. Cette sélection donne à chaque individu une portion de la roulette proportionnelle à sa fitness, ainsi dans notre cas, plus la fitness d'une solution (individu de notre population) est faible, plus elle a des chances d'être sélectionnée pour être reproduite. Avec cette méthode, on s'assure que nos individus performants ont plus de chances d'être sélectionné mais on garde quand même une exploration en faisant se reproduire aussi des solutions moins bonnes.

Croisement

Concernant notre croisement, nous utilisons deux index tirés aléatoirement. Le premier entre 0 et la taille de notre parent 1. Le deuxième, entre 1 et la taille de notre parent tout en évitant la même valeur que le premier index, on s'assure alors d'effectuer un croisement. Notre premier enfant sera donc construit avec la première partie du parent 1, la deuxième partie du parent 2 et la dernière partie du parent 1. On s'assure d'éviter des doubles en vérifiant que l'enfant ne contient déjà pas le client.

Étape importante après notre croisement, on ordonne les tournées de nos enfants. Pour cette étape, on fait tourner un léger algorithme qui à chaque client, va le relier au client le plus proche de ce dernier. Si cet algorithme a comme avantage d'être rapide, il nous fait gagner de la distance et ainsi obtenir une fitness moins bonne, exemple :



On remarque bien dans cette solution que notre client de la tournée 2 choisi le plus proche pour lui, mais que ce n'est pas le meilleur choix en terme de distance globale de cette tournée.

Une solution pour résoudre à ce problème serait d'utiliser un autre algorithme, plus coûteux, qui reconstruirait nos tournées avec le chemin le plus optimisé possible. Une autre solution serait de faire le même parcours avec le client le plus proche mais en commençant de l'autre côté de notre tournée, comparer les deux solutions et prendre la meilleure.

Mutation

Pour nos mutations, nous avons repris nos méthodes de voisinage, nous avons donc trois mutations possibles :

- Echange de deux clients à l'intérieur d'une tournée
- Echange de deux clients entre deux tournées différentes
- Suppression d'un client puis ajout de ce dernier dans une tournée différente.

Description de notre algorithme génétique

```
public Solution process() {
    for(int genIndex = 0; genIndex < GENERATIONS_NB; genIndex++) {
        selectPopulation();
        doCrossover();
        doMutation();
        updateValues();
        if (genIndex % 1000 == 0) {
            System.out.println("Generation : " + genIndex + ", best distance : " + FormatUtils.round(_bestDistance, places: 2)); }
    }
    return _bestSolution;
}
```

On remarque que le principe même de notre algorithme est très simple. En plus de cet algorithme, nous avons essayé d'implémenter une méthode de descente, cette méthode va nous permettre de chercher un minimum local à chaque (ou un chaque n action : crossover et mutation). Si nous pensions que cette méthode allait grandement augmenter nos résultats, nous avons au final pas forcément obtenu des résultats satisfaisant.

Description des résultats

Tableau des résultats obtenus avec notre algorithme génétique sans descente, les résultats affichés dans le tableau sont à peu près à +10/-10 (dépend des exécutions) du maximum obtenu avec le fichier.

Fichier	Coût initial	Nombre de générations	Taille de la population	Probabilité de mutation	Résultat
A3205	2091.46	300000	20	5%	875.4
A3305	1868.24	300000	20	5%	733.35
A4506	3032.97	300000	20	5%	1201.16
A3705	1862.67	300000	20	5%	809.95

Nous avons choisi d'utiliser un grand nombre de générations, ce nombre est plus ou moins utile selon les fichiers, plus le nombre de clients est important, plus le nombre de générations est impactant. Ce grand nombre de générations impacte aussi nos mutations, elles seront plus nombreuses avec une même probabilité de mutation. Ce qu'on remarque aussi c'est qu'on obtient des résultats satisfaisant très rapidement (quelques secondes) mais il nous faut beaucoup de générations pour atteindre un résultat plus proche de ce qui semble être une des meilleures solutions.

Exemple en prenant en seulement 10 000 générations :

Fichier	Coût initial	Nombre de générations	Taille de la population	Probabilité de mutation	Résultat
A3205	1887.74	10000	20	5%	914.56
A3305	2231.48	10000	20	5%	777.25
A4506	2724.9	10000	20	5%	1288.3
A3705	2145.89	10000	20	5%	835.78

On remarque bien qu'on perd effectivement en fitness mais le temps d'exécution passe de 2-3 minutes à au maximum une dizaine de seconde, ce gain de temps pourrait justifier une perte de fitness.

Testons maintenant l'impact de la taille de la population, divisons la par deux et reprenons nos paramètres initiaux.

Fichier	Coût initial	Nombre de générations	Taille de la population	Probabilité de mutation	Résultat
A3205	2139.43	300000	10	5%	862.72
A3305	1806.95	300000	10	5%	740.58
A4506	2907.71	300000	10	5%	1234.2
A3705	1931.74	300000	10	5%	820.52

En doublant la taille de la population maintenant :

Fichier	Coût initial	Nombre de générations	Taille de la population	Probabilité de mutation	Résultat
A3205	2071.12	300000	40	5%	862.44
A3305	1977.13	300000	40	5%	740.25
A4506	2698.68	300000	40	5%	1224.47
A3705	2012.66	300000	40	5%	790.8

En moyenne pour nos fichiers, il n'y a aucune amélioration, ou alors elle est très faible, ce qui est lié au caractère aléatoire de notre algorithme. La taille de la population ne semble pas influer énormément dans notre cas.

Maintenant, voyons l'influence de la probabilité de mutation qui est censée être faible. Testons avec 1% et ensuite 20% (les tests avec 5% sont présents au début)

Avec 1% :

Fichier	Coût initial	Nombre de générations	Taille de la population	Probabilité de mutation	Résultat
A3205	2157.3	300000	20	1%	879.65
A3305	1838.29	300000	20	1%	739.7
A4506	2704.91	300000	20	1%	1225.95
A3705	1871.98	300000	20	1%	811.15

Avec 20% :

Fichier	Coût initial	Nombre de générations	Taille de la population	Probabilité de mutation	Résultat
A3205	2226.06	300000	20	20%	864.88
A3305	1729.21	300000	20	20%	748.61
A4506	2937.72	300000	20	20%	1224.83
A3705	1963.76	300000	20	20%	800.23

Ce qu'on retient, c'est que la probabilité de mutation en elle-même n'influe pas de façon assez importante sur nos résultats. Effectivement, ce qui importe c'est le nombre de générations, notre probabilité de mutation aura beaucoup plus d'impact sur un petit nombre de générations si elle est haute, et beaucoup moins d'impact si le nombre de générations est faible. Le nombre de génération est le paramètre qui va le plus influer sur les autres.

Testons maintenant l'impact de notre méthode de descente. Celle-ci est travaillée pour faire un maximum de multi-threading et ralentir un minimum notre algorithme pour les générations où nous l'utilisons. les paramètres du constructeur de notre algorithme génétique nous permettent de définir toutes les combien de générations nous voulons passer une descente sur notre population, et avec quelle profondeur maximale (pour ne pas se perdre en n'ayant que de très faibles améliorations de notre résultat).

Nos tests qui suivent sont réalisés avec nos valeurs de paramétrage de références, obtenues en faisant de nombreux tests sur le fichier *A3305.txt*. Ces tests sont présentés dans le fichier Excel joint à ce rapport : *parametrage.xlsx*. Des captures d'écrans des résultats de nos tests sont disponibles dans le dossier *parametrage*, tandis que des captures d'écran de nos résultats sont disponibles dans le dossier *resultats*.

Nous comparons les résultats que nous avons obtenu avec notre algorithme génétique amélioré avec les métaheuristiques à base de voisinage du TP précédent :

Fichier	Simulated Annealing	Tabu	Tabu puis Simulated Annealing	Genetic
A3205	801.71	855.39	829.41	790.88
A3305	662.26	790.88	662.11	674.58
A3705	682.33	840.41	672.35	718.72

Nous voyons que notre Algorithme Génétique est largement meilleur que le Tabu Search, tant en résultat qu'en durée d'exécution (généralement 2 fois plus court). Cependant, le Simulated Annealing obtient de bien meilleurs résultats, mais dans un laps de temps beaucoup plus élevé (environ 10 fois plus long que l'Algorithme Génétique).

Conclusion

Pour ce projet, nous avons pu bénéficier de la structure que nous avions mis en place pour les métaheurisitques à base de voisinage. Nous n'avons donc presque pas touché à notre structure, seulement ajouté un algorithme Greedy de type proche-en-proche pour organiser nos noeuds dans nos camions après les crossovers, et un constructeur de Solution qui prend en paramètre une liste ordonnée de noeuds, nous permettant ainsi de remplir les camions après le crossover.

Cependant, le développement de notre algorithme génétique a été un petit challenge, car bien que très rapide au départ (600 000 générations en 2 minutes sans descente), il n'était pas très efficace, les résultats n'étaient pas excellents. Néanmoins, nous avons réutilisé notre méthode pour obtenir le meilleur voisin pour Tabu Search afin de mettre en place une descente sur le minimum local, le tout en utilisant le multithreading au maximum ! Ce qui a payé, puisque nous avons un algorithme génétique donnant de bons résultats rapidement en utilisant 100% du CPU !

Beaucoup plus rapide que notre Simulated Annealing du précédent TP, nous pensons que nous avons su prendre à coeur ce projet pour obtenir de bons résultats et un algorithme qui, s'il ne donne pas toujours le meilleur résultat, nous donne rapidement une valeur très bonne !

Nous sommes fier d'avoir réussi à améliorer ce que nous avions fait au TP précédent.