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 voisinage.

Réalisation du travail demandé

Modélisation

Vu la taille conséquente du projet, nous avons décidé de fournir un travail important sur la structure même du projet, nous avons tâché à utiliser un modèle le plus objet possible avec Java. Notre projet est donc découpé en de multiples packages regroupant nos classes selon leurs fonctionnalités. Dans le package :

- *algo*, nos deux algorithmes Recuit simulé et Tabou.
- *data,* la classe qui permet de charger dans une liste nos données
- *display,* la classe permettant l'affichage de nos résultats
- *graph,* toutes les classes permettant de générer nos Tours, nos Clients, et tous les objets que nous avions utilisé pour nos algorithmes
- *utils,* toutes les classes utilitaires, que ce soit pour générer nos random, nos exceptions…

Pour la modélisation graphique, nous n'avons pas importé de librairies particulières, nous avons utilisé AWT Graph qui est intégré directement à Java.

Concernant nos objets, nous avons utilisé Java 11 et toutes les fonctionnalités que cette version intègre (expressions lambda, parallel stream qui permettent un multithreading, …).

Nous avons aussi essayé d'optimiser un maximum notre programme en réduisant la complexité boucles et algorithmes afin de gagner en vitesse de calcul et en RAM.

L'autre objectif était aussi de rendre notre code réutilisable et compréhensible par tous. Nous avons donc durement commenté nos sources, utilisé des variables claires et créé une architecture la plus objet possible.

Méthodes de voisinage

Pour générer nos voisins aléatoires, nous avons créé trois fonctions de voisinage :

- 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 des résultats

Pour la présentation des résultats suivante, nous aborderons seulement les solutions des fichier A3205 et A3305.

Algorithme Simulated Annealing

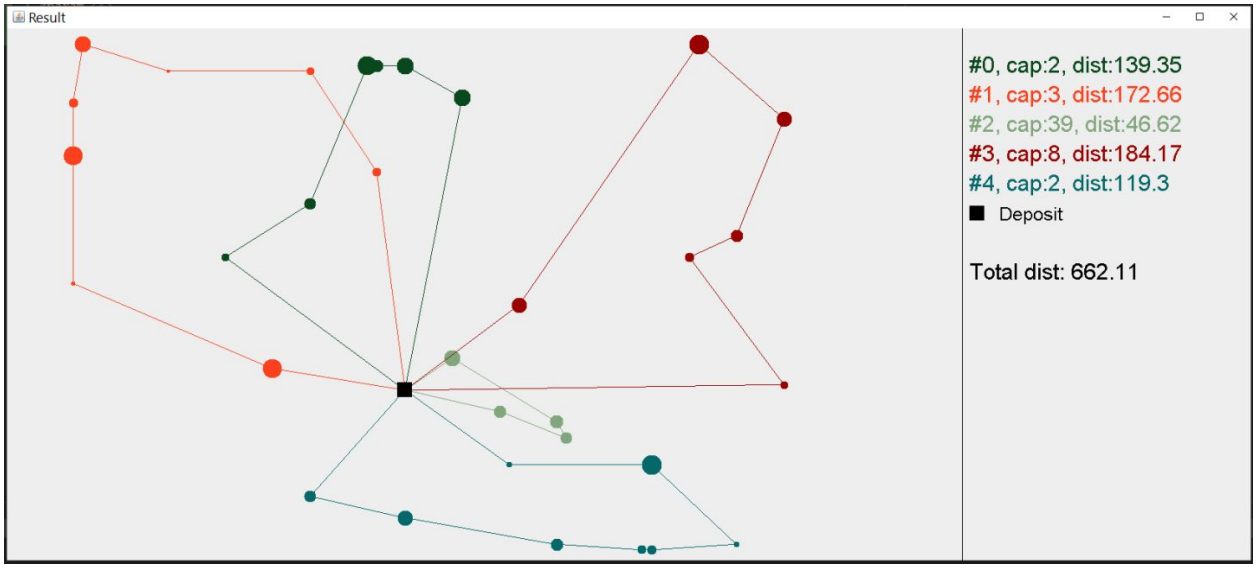
Pour cet algorithme, nous avons suivi votre pseudo code avec cependant un léger twist. Lors de la génération de notre voisin aléatoire, nous avons une boucle génératrice de voisins, si ce dernier est meilleure que l'ancien, on le garde pour générer nos voisins suivants avec néanmoins une infime probabilité de garder ce moins bon voisin générateur.

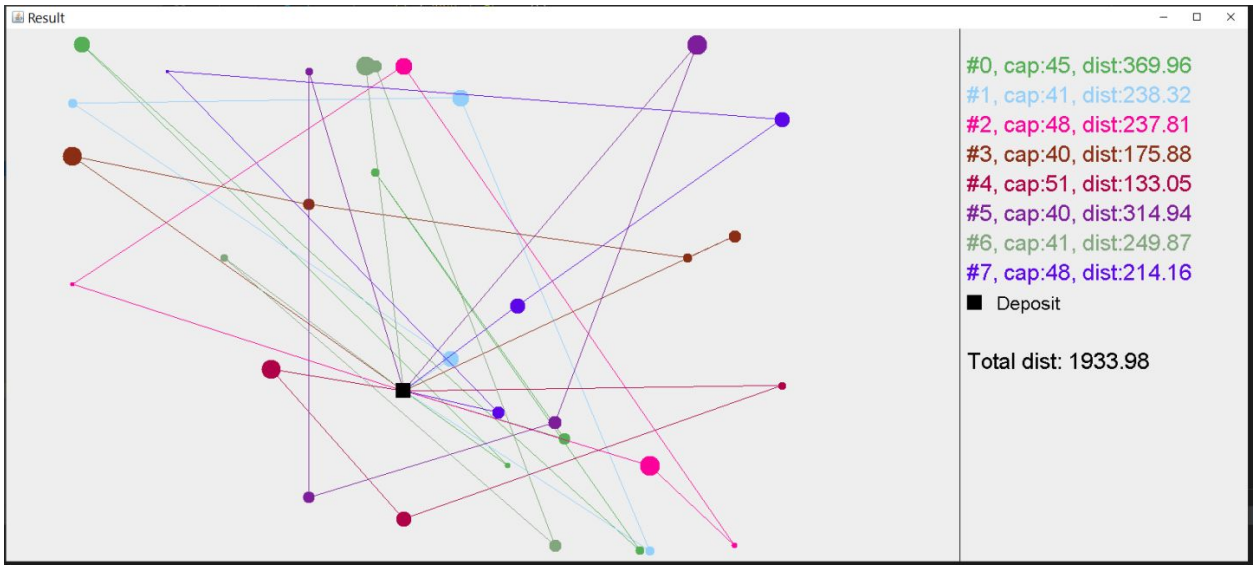
Concernant les choix des paramètres, nous avons testé de multiples solutions et nous sommes arrivés à la conclusion que d'avoir une température à 1000, un cooling factor à 0.99 et un nombre de voisins à 2000 (par boucle génératrice) nous permettait d'avoir une très bonne solution. Pour obtenir les meilleurs paramètres, nous avons laissé tourner un programme toute une nuit qui a généré un grand nombre d'exemples en changeant les paramètres de ces derniers. Les paramètres obtenus étaients : température : 1300, cooling factor : 0.994, nombre de voisins : 1230. Nous avons ensuite effectué quelques

modifications dans une fonction de voisinage et nous avons été amené à utiliser les paramètres précédemment donnés qui combinent un temps d'exécution raisonnable avec des résultats plus que corrects.

Facteur important à prendre en compte : chaque initialisation de notre algorithme se base sur une première génération aléatoire instanciée avec des camions remplis à 60% de leur capacité max.

Après l'exécution de notre programme nous avons obtenu ce résultat à partir de la génération aléatoire suivante.



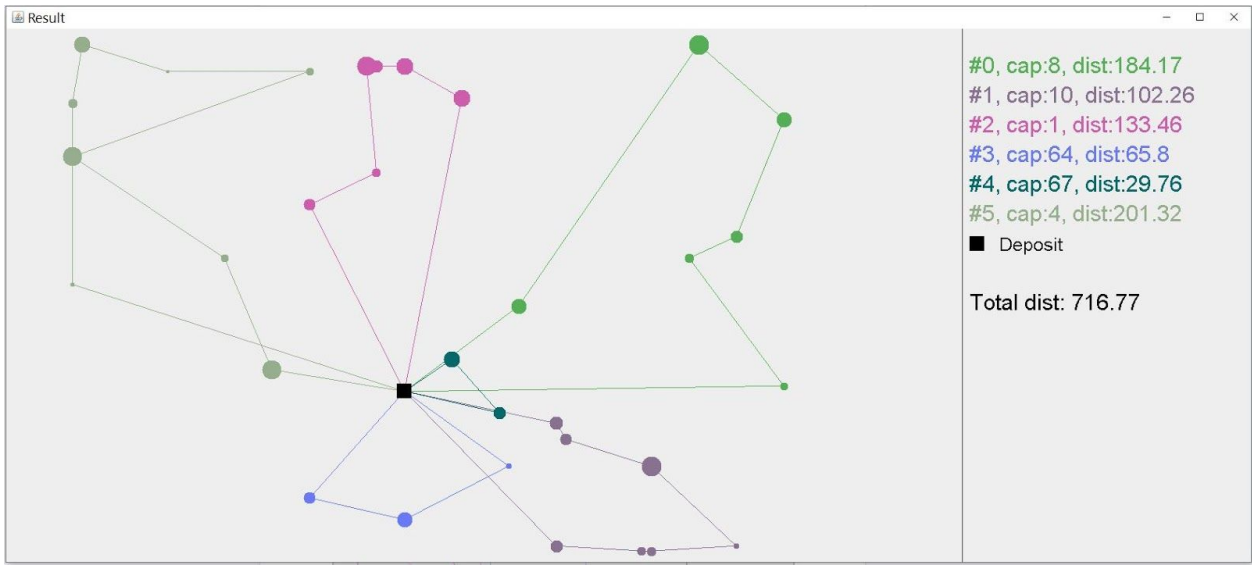


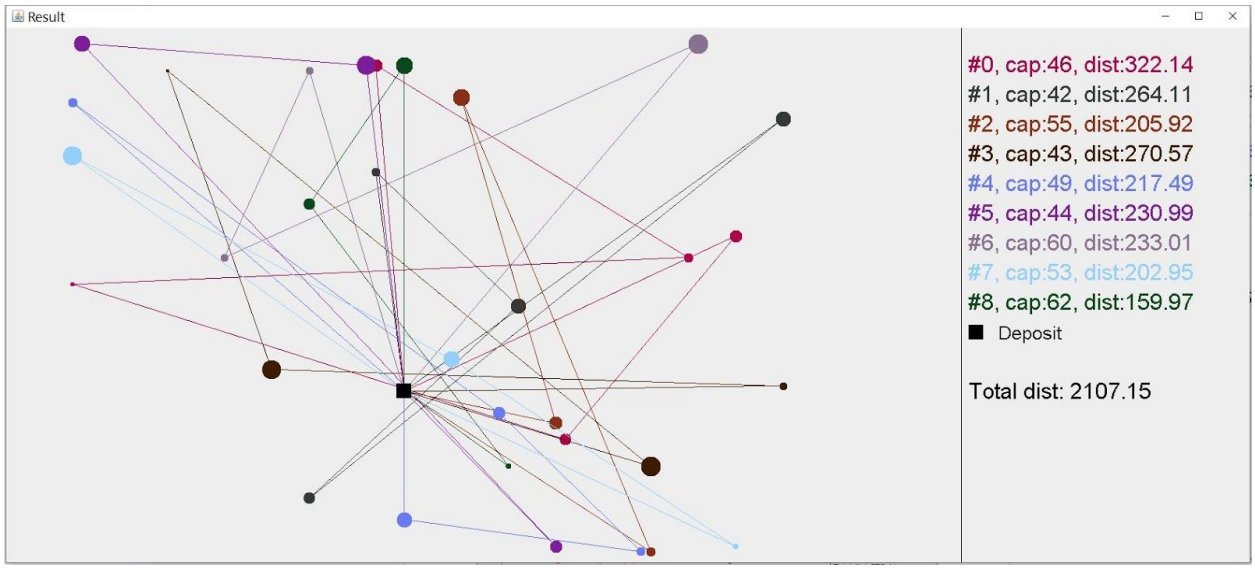
Le résultat obtenu est très bon, on arrive à une distance de 662.11 qui est notre meilleure solution. Ce qu'il faut savoir c'est que la génération aléatoire n'influe **pas** notre distance finale, ce qui est le cas pour l'algorithme Tabou.

Algorithme Tabu

Pour cet algorithme, nous avons cette fois totalement suivi votre pseudo code. Nous avons eu du mal à trouver les paramètres adéquats pour cet algorithme. Après de nombreux tâtonnements, avec un nombre maximum d'itérations fixé à 6000, une taille de liste Taboue fixée à 150, et une condition d'arrêt à fixée à 500 itérations sans amélioration de la meilleure solution, nous trouvons des solutions se rapprochant de la meilleure, dans un temps de calcul raisonnable. La liste Taboue est remplie avec les Solutions précédemment visitées selon les conditions de l'algorithme, mais nous n'avons pas introduit la notion de transformation au sens Tabu Search dans notre programme.

Voici un exemple d'exécution, où nous avons obtenu la valeur de 716.77, la capture d'écran suivante correspondant à la solution aléatoire initiale. La génération est faite de la même manière que pour l'algorithme Simulated Annealing.





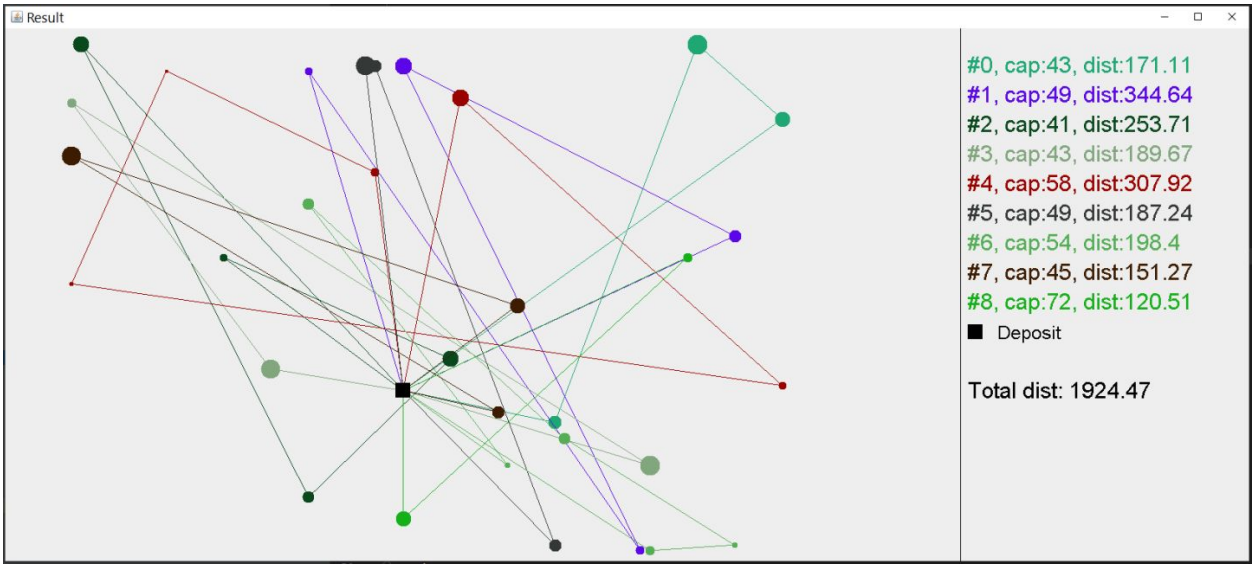
Nous n'avons jamais cependant approché à une distance de moins de 25 de la meilleure solution obtenue avec Simulated Annealing, quelques soient nos paramètres de Tabu Search.

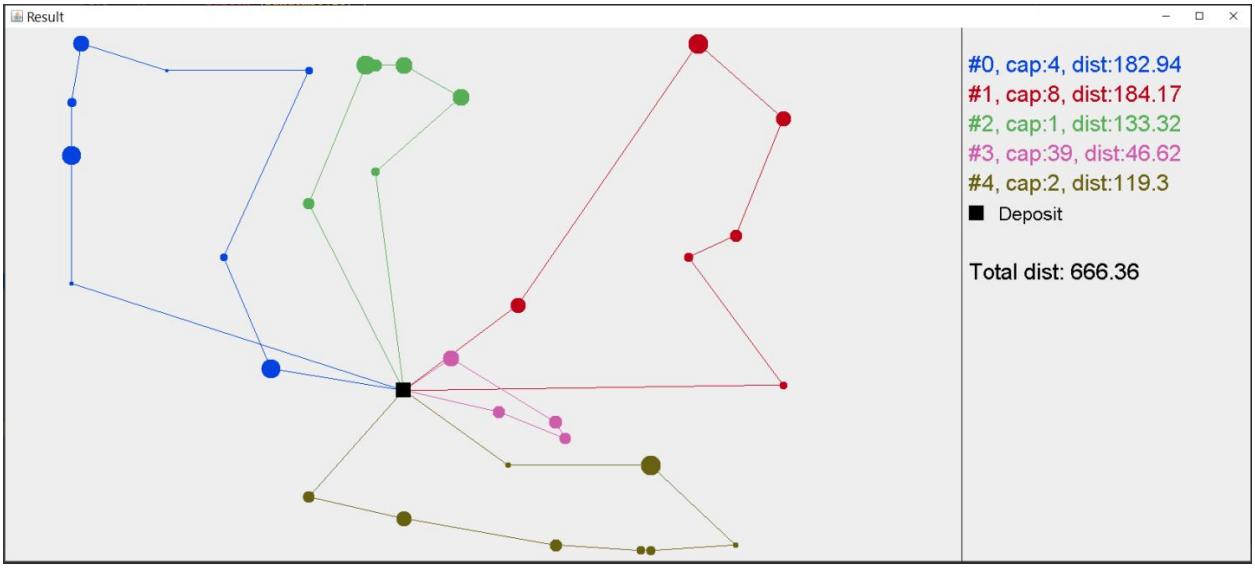
Nous en avons déduit que le nombre potentiels de voisins directs pour notre représentation et nos fonctions de voisinage est beaucoup trop énorme pour appliquer Tabu Search, ou alors il nous aurait fallu créer une méthode pour accepter des Solutions comme étant voisines de manière beaucoup plus lâche que ce que nous faisons actuellement.

Combinaison des deux algorithmes

Pour aller plus loin, nous avons testé d'utiliser nos deux algorithmes l'un après l'autre, en commençant d'abord par une première phase Tabou allégée pour dégrossir notre solution et une dernière phase Recuit Simulé elle également allégée qui permettra d'affiner la Solution.

Pour le fichier A3305, nous obtenons les résultats suivants à partir de cette génération aléatoire.





Conclusion

Ce projet fût beaucoup plus complexe et long que nous l'avions imaginé. Notre soucis de l'optimisation nous a forcé à prendre beaucoup de temps sur notre structure, plus que sur nos algorithmes. Le résultat que nous avons obtenu est heureusement pour nous grandement satisfaisant et nous sommes fiers de notre travail. Nous gardons une préférence pour l'algorithme Simulated Annealing qui est selon nous le plus efficace face à notre problème initial. S'il n'existe aucune solution parfaite pour ce problème, nous pensons s'être approché de celle-ci et avoir relevé le défi de ce projet