ISIMA

OUTILS D'AIDE A LA DECISION

Tournée de véhicules avec fenêtre de temps

Mise en place d'algorithmes de minimisation de tournées et de distance au sein de problèmes de tournées de véhicules avec fenêtres de temps.

BARBESANGE Benjamin – GARÇON Benoît

17/01/2016

Table des matières

INTRODUCTION

Ce projet s'inscrit dans le cursus de seconde année à l'ISIMA. Nous devons ici traiter de problèmes de tournées de véhicules chez des clients qui disposent d'une heure d'ouverture et de fermeture.

Nous disposons également d'un dépôt ayant un horaire d'ouverture et de fermeture. Le nombre de véhicule mis à disposition est illimité, ce qui signifie que le nombre de tournées que nous pouvons effectuer l'est aussi. Il faut tout de même prendre en compte le chargement des véhicules qui ne peut pas excéder une certaine valeur.

Nous devons traiter tous les clients d'une tournée définie à l'aide d'un seul véhicule et en une seule fois, nous ne pouvons pas retourner au dépôt au cours de la tournée.

Le problème ici est donc de trouver une solution dans laquelle nous avons un nombre minimum de tournées qui satisfont les conditions énoncées ci-dessus et également de minimiser la distance totale des tournées.

Dans un premier temps nous construiront une solution de base à l'aide d'une heuristique d'insertion, puis nous améliorerons cette solution avec plusieurs heuristiques. Enfin une méta-heuristique va permettre d'améliorer encore plus la solution obtenue.

I – ETUDE DU PROBLEME

Le problème consiste à déterminer un nombre de tournée minimal à créer afin de pouvoir desservir un nombre de clients définis. Nous disposons d'informations sur le client, comme ses horaires d'ouverture et de fermeture, la quantité de marchandise qu'il souhaite, son temps de service. Nous disposons aussi d'une grille dans laquelle figure les distances entre chaque client que nous gérons. Notons que chaque client n'est pas forcément relié par une route.

Le dépôt va constituer le point de départ et d'arrivé pour chaque tournée que nous effectuerons. Ce dépôt dispose d'un nombre de véhicule illimité ce qui nous permet de créer autant de tournées que nous le souhaitons.

Il est nécessaire de servir tous les clients et ceux en une seule fois. Les tournées doivent s'effectuer sans retour au dépôt au milieu de celle-ci.

A – Génération d'une solution

Pour générer une première solution, nous avons 2 options.

Une première consiste à créer une tournée par client. Ainsi nous sommes sur que cette solution est réalisable.

Une seconde approche va consister à créer des tournées avec un client de base et ajouter un client à la fin de cette tournée tant que c'est possible. Lorsqu'on ne peut plus ajouter de clients, nous créons une nouvelle tournée et continuons. Ceci est à faire tant qu'il reste des clients non traités.

B – Amélioration de cette solution

Une fois une solution de base établie, il faut améliorer cette solution. Pour l'améliorer, nous pouvons utiliser successivement des heuristiques d'amélioration.

Une première heuristique va permuter les fins de deux tournées. Ainsi il est possible de fermer des tournées si on ajoute toute une tournée à la fin d'une autre.

Une autre heuristique va tester s'il est possible de déplacer un client d'une tournée dans une autre tournée. De cette manière, nous supprimons successivement des clients dans certaines tournées et donc on en limite le nombre.

Enfin une dernière heuristique va échanger deux clients de deux tournées.

En utilisant successivement ces heuristiques et en réitérant tant qu'on améliore notre solution, il est possible d'obtenir de très bonnes solutions.

C – Fabrication de meilleures solutions

Afin de fabriquer de meilleures solutions, nous allons utiliser une méta-heuristique qui va partir d'une solution de base, l'améliorer tant que possible. Ensuite nous regardons si cette solution obtenue est meilleure que celle de base (en temps ou en nombre de tournées). Si c'est le cas, on sauvegarde cette solution. Nous répétons ce processus sur un nombre d'itération maximal.

II – PRESENTATION DE LA SOLUTION

A – Heuristique de construction

Présentation

L'heuristique que nous avons implémentée ici est une heuristique d'insertion. Nous allons partir de notre liste de clients que nous mélangeons puis nous prenons le premier client pour créer le début de la tournées (en n'oubliant pas de dépôt). Nous recherchons ensuite un client à insérer en fin de cette tournée. Si aucun client ne peut être inséré, nous créons une nouvelle tournée. Ce processus est à refaire tant qu'il reste des clients non traités.

Algorithme

```
Créer une tournée vide
Faire
      Rechercher un client satisfaisant
            - La charge de la tournée
            - La fenêtre de temps
      Si il existe Alors
            L'inserer dans la tournée
      Sinon
            Créer une nouvelle tournée
      Fin Si
Tant que, il reste des clients libres
```

Implémentation

Cet algorithme est implémenté dans le fichier algorithms.cpp, avec la méthode *insertion()*.

B – Heuristiques d'amélioration

Présentation

Nous disposons de 3 heuristiques d'améliorations principales.

La première heuristique est de type 2 opt. Cette heuristique va tenter d'échanger les fins de deux tournées choisies dans la solution, tout en veillant au respect des contraintes de la solution (fenêtres de temps, charges). Nous allons également trouver sa variante 2 opt* contenant le cas particulier où nous allons concaténer toute une tournée à la fin d'une autre.

La seconde heuristique que nous utilisons est de type or opt. Ici, nous voulons déplacer un client se trouvant dans une tournée, vers une autre tournée, toujours en respectant les contraintes. Nous trouvons aussi sa variante or opt* concernant le cas où nous déplaçons un client qui compose une tournée à lui seul.

Enfin la dernière heuristique est une heuristique de type cross. Son rôle est d'échanger deux clients situés dans 2 tournées différentes.

Implémentation

L'implémentation de ces heuristiques est disponible dans le fichier algorithms.cpp via les méthodes :

- *Opt2()*

- *Opt2Etoile()*
- *OrOpt()*
- *OrOptEtoile()*

C - Recherche locale

Présentation

La recherche locale va consister à appeler successivement nos heuristiques d'amélioration sur la solution jusqu'à ce que l'on ne puisse plus améliorer du tout.

Algorithme

On place dans un tableau les heuristiques, de la manière suivante :

Rl[0]	Rl[1]	Rl[2]	Rl[3]	Rl[4]
2Opt*	OrOpt*	2Opt	OrOpt	Cross

Ensuite nous suivons l'algorithme suivant :

```
Entrée : s la solution, Rl le tableau d'heuristiques
k <- 0
Tant que k < 5 Faire
      Appelle la fonction Rl[k] sur la solution s
      Si la solution est ameliorée Alors
            On garde l'amelioration
            k <- 0
      Sinon
            k <- k + 1
      Fin Si
Fin Tant que
```

Implémentation

La recherche locale est implémentée dans le fichier algorithms.cpp avec la méthode *RechLocComplete()*.

Nous avons utilisé une classe par heuristiques dans lesquelles nous avons surchargé l'opérateur () afin de les utiliser aisément à partir de notre tableau. La recherche locale se trouve dans une classe possèdant comme membre un tableau se composant de nos différentes heuristiques

D - Algorithme principal

Présentation

L'algorithme principal va permettre d'approcher la solution optimale. Nous allons tester sur un nombre d'itérations maximum défini plusieurs solutions. Nous commençons par utiliser l'heuristique d'insertion pour avoir la solution de base. Ensuite nous améliorons autant que possible cette solution avec la recherche locale. Puis nous la comparons avec des solutions précédentes. Si nous avons amélioré la distance ou le nombre de tournées, nous conservons cette solution.

Algorithme

```
Entrée : la solution s contenant les données du problème, iterMax le
nombre maximum de solution testées
Creation d'une tournée initiale avec l'insertion et les clients triés
par moyenne de fenêtre de temps
Stocke cette tournée dans bsd et bsn (meilleures en distance et
nombre)
```

```
Pour i de 1 à iterMax Faire
      Heuristique d'insertion sur s
      Recherche locale sur s

      Si distance_totale de s < distance_totale de bsd Alors
            bsd = s
      Fin Si
      Si nombre_routes de s < nombre_routes de bsn Alors
            bsn = s
      Fin Si
Fin Pour

Afficher distance_totale de bsd [Ou afficher les tournées]
Afficher nombre_routes de bsn [Ou afficher les tournées]
```

Implémentation

La méta-heuristique est implémentée dans le fichier algorithms.cpp., via la méthode *MetaHeuristique()*.

III – RESULTATS ET PERFORMANCES

A - Présentation du programme

Nous disposons d'un programme dans lequel il est possible de spécifier un fichier à aller lire pour charger les données d'un problème. On peut le spécifier directement dans la ligne de commande. On peut aussi préciser le nombre maximum d'itérations sur la méta-heuristique en 2ème argument.

Ensuite, le programme va directement lancer la méta-heuristique sur le problème donné, et effectuer le nombre d'itérations spécifié ou utiliser une valeur par défaut.

B – Tests et analyse

Dans le tableau ci-dessous nous testons notre méta-heuristique sur toutes les instances de type rc pour 100 itérations.

INSTANCE	NB ROUTES LITTERATURE	DISTANCE LITTERATURE	NB ROUTES POUR 100 ITERATIONS	DISTANCE POUR 100 ITERATIONS	TEMPS D'EXECUTION EN SECONDES
rc101	14	1696.94	18 (+4)	2011.45 (18.57 %)	0.45
			21 (+7)	1999.40 (17.87 %)	
rc102	12	1554.75	19 (+7)	1843.73 (18.60%)	0.50
			20 (+8)	1824.55 (17.37%)	
rc103	11	1261.67	15 (+4)	1590.40 (26.09%)	0.54
rc104	10	1135.48	13 (+3)	1481.83 (30.48%)	0.62
			14 (+4)	1456.78 (28.28%)	
rc105	13	1629.44	21 (+8)	1929.11 (18.42%)	0.51
			21 (+8)	1884.86 (15.65%)	
rc106	11	1424.73	17 (+6)	1740.77 (22.19%)	0.52
			17 (+6)	1698.10 (19.24%)	
rc107	11	1230.48	15 (+4)	1664.69 (35.28%)	0.57
			17 (+6)	1603.56 (30.33%)	
rc108	10	1139.82	14 (+4)	1563.32 (37.23%)	0.62
			16 (+6)	1502.23 (31.87%)	
rc201	4	1406.94	6 (+2)	1748.34 (24.32%)	0.49
			11 (+7)	1454.84 (3.41%)	
rc202	3	1365.65	6 (+3)	1765.74 (29.30%)	0.61
			10 (+7)	1400.41 (2.56%)	
rc203	3	1049.62	6 (+3)	1479.07 (40.99%)	0.76
			9 (+6)	1181.01 (12.58%)	
rc204	3	798.46	4 (+1)	1127.96 (41.23%)	0.92
			5 (+2)	1075.96 (34.71%)	
rc205	4	1297.65	8 (+4)	1486.59 (14.57%)	0.58
			10 (+6)	1372.27 (5.78%)	
rc206	3	1146.32	6 (+3)	1575.85 (37.43%)	0.55
			8 (+5)	1325.25 (15.62%)	
rc207	3	1061.14	5 (+2)	1392.84 (31.20%)	0.60
			7 (+4)	1221.86 (15.08%)	
rc208	3	828.14	5 (+2)	1196.61 (44.44%)	0.75

Comme on peut le constater dans le tableau précédent, note algorithme trouve en un temps très faible (une demi seconde environ) des résultats très proches des valeurs optimales.

On peut aussi remarquer qu'il est plus efficace sur le nombre de tournées que sur la distance totale, en effet, il a été conçu pour favoriser la réduction du nombre de tournées.

En augmentant le nombre d'itération on peut affiner encore le résultat sans perdre trop de temps. Voici les résultats sur rc101 en fonction du nombre d'itérations :

RC101	1	100	1000	10000
TEMPS (s)	0.01	0.44	4.42	44.60
TOURNEES	18	18	18	18
DISTANCE	2011.45	1999.40	1979.03	1881.40

On trouve très rapidement (une itération) un résultat satisfaisant dès le début grâce à la méthode du tri des clients par moyenne de fenêtre de temps.

Ensuite le résultat s'affine en gardant une complexité proportionnelle au nombre d'itérations.

Conclusion

En conclusion, nous pouvons remarquer qu'il est difficile de trouver la meilleure solution à ce problème. En effet, tout dépend si l'on souhaite minimiser le nombre de véhicules disponible ou la distance totale ou encore les deux à la fois.

Il faut donc dans un premier temps partir d'une solution naïve et réalisable pour le problème, à partir de laquelle nous allons améliorer. Ensuite, à l'aide d'une recherche locale, nous allons pouvoir diminuer le nombre de tournées en échangeant des fins de tournées ou encore en déplaçant des clients dans une autre tournée. Enfin à l'aide d'une méta-heuristique, nous établissons une solution finale qui s'approche au mieux de l'optimalité.

Ceci peut s'effectuer en un temps relativement raisonnable, alors que la détermination de la solution optimale à un problème donné prendrait un temps non envisageable pour certaines entreprises ou industries, pour lesquelles ce genre de problèmes est à maitriser au quotidien.