Description détaillée de l'algorithme et de ses composantes

Clément Legrand-Lixon

Today

Ce papier présente l'algorithme mis en œuvre dans le cadre du problème CVRP (Capacitated Vehicle Routing Problem), et s'inspire en grande partie de l'article d'Arnold et Soransen.

Ce rapport, commence par décrire les opérateurs mis en œuvre par la suite et introduit les notations utilisées. L'algorithme commence par initialiser une solution. Il applique ensuite des opérateurs locaux, tant que la condition d'arrêt n'est pas respectée. Lors de certaines itérations, il peut effectuer des opérations spéciales. A la sortie de la boucle, il effectue un raffinement de la solution.

Parcours et exploration des voisinages

L'exploration d'un voisinage de solutions peut être plus ou moins exhaustif selon la condition d'arrêt utilisée. On distingue principalement, deux conditions d'arrêt lorsqu'il s'agit d'explorer des voisinages :

— First improvement (FI) : on parcourt le voisinage jusqu'à trouver un changement qui améliore la solution actuelle (on s'arrête donc à la première amélioration trouvée) ;
— Best improvement (BI) : on parcourt tout le voisinage, et on applique le changement qui va le plus améliorer notre solution actuelle.

Pour explorer un voisinage, on peut le parcourir de différentes manières de sorte à ne pas toujours favoriser les mêmes voisins. On considérera ici 3 parcours différents :

— Dans l'ordre (O) : les voisins sont parcourus dans un ordre naturel (du premier au dernier) ;
— Dans un semi-ordre (SO) : on commence le parcours là où on s'était arrêté au dernier parcours, on parcourt ensuite les voisins dans l'ordre ;
— Aléatoirement (RD) : on tire aléatoirement l'ordre dans lequel on va parcourir les voisins.

On peut remarquer que peut importe le parcours effectué, pour faire une exploration BI, il faudra passer par tous les voisins. On retiendra le tableau récapitulatif suivant :

FI	x	BI
O	Oui	Oui
SO	Non	Oui
RD	Non	Oui

Brique initialisation

Cette brique consiste en la création d'une solution initiale, sur laquelle seront appliquées les modifications.

Cette solution peut être créée de différentes manières :

— Elle peut être générée aléatoirement ;
— Elle peut être obtenue grâce à l'algorithme de Clarke & Wright, avec les paramètres (λ, μ, ν) :
  — Construire autant de tournées que de clients qui passent par le dépôt ;
  — Pour toutes les paires de clients (i, j) calculer le saving $s(i, j)$ via la formule :
  $$s(i,j) = c_{i0} + c_{0j} - \lambda c_{ij} + \mu |c_{i0} - c_{0j}| + \nu \frac{d_i + d_j}{\bar{d}}$$
  où c_{ij} dénote la distance entre les clients i et j (le client 0 étant le dépôt), et d_i dénote la demande du client i.
  — Considérer le couple (i, j) possédant le saving le plus élevée ;
  — Fusionner les tournées auxquelles appartiennent i et j, si possible (tournées différentes, et i et j doivent être premier et dernier clients de leur tournée) ;
  — Mettre $s(i, j) = 0$
  — Recommencer tant que le saving maximal n'est pas négatif.
— Elle peut être calculée grâce à l'intégration de connaissances (objectif du stage).

Il est possible de compléter cette brique avec la brique raffinement vue plus loin.

Condition d'arrêt

La plus grande partie de l'algorithme consiste en l'exécution d'une boucle, tant que la condition d'arrêt n'est pas respectée. Cette condition peut prendre plusieurs formes :

— Un nombre d'itérations à ne pas dépasser ;
— Un certain temps d'itérations à ne pas dépasser (3 minutes dans l'article) ;
— Un nombre d'itérations sans solutions améliorantes à ne pas dépasser (de l'ordre de n^2 dans mon algorithme).

Opérateurs dans la boucle

Différents opérateurs sont appliqués à la solution obtenue après la brique initialisation.

Pire arête et pénalisation

A chaque tour de boucle, on calcule l'arête (i, j) qui maximise la fonction suivante (donnée dans l'article) :

$$b(i,j) = \frac{[\lambda_w w(i,j) + \lambda_c c(i,j)][\frac{d(i,j)}{max_{k,l} d(k,l)}]^{\frac{\lambda_d}{2}}}{1+p(i,j)}$$

où :

— $p(i, j)$ est la pénalisation de l'arête (i, j) (nombre de fois où l'arête a maximisé b) ;
— $w(i, j)$ est la largeur de l'arête (i, j) ;
— $c(i, j)$ est le coût de l'arête (i, j) ($c(i, j) = c_{ij}(1 + \lambda p(i, j)$, avec $\lambda = 0.1$ dans l'article) ;
— $d(i, j)$ est la profondeur de l'arête (i, j) (max de c_{0i} et c_{0j}) ;
— les paramètres $\lambda_w, \lambda_c, \lambda_d$, prennent comme valeurs 0 ou 1, selon les caractéristiques que l'on veut considérer. Il y a ainsi 6 fonctions de pénalisation différentes, que l'on peut choisir au cours de l'exécution.

C'est autour de l'arête calculée ici que vont s'orienter les recherches des opérateurs locaux qui suivent.

Ejection-Chain

Cet opérateur va essayer de déplacer au plus l clients sur des tournées plus adaptées. Dans l'article $l = 2$.

Cet opérateur s'exécute de la manière suivante :
— Déterminer une arête à éliminer ;
— Considérer un des deux clients ;
— Regarder parmi les pp-voisins pour trouver une deuxième tournée ;
— Déplacer le client sur cette tournée (après le voisin trouvée) ;
— Essayer de déplacer un client de cette tournée sur une autre tournée (en répétant les étapes précédentes) ;
— Recommencer l'étape précédente l fois ;

Cross-Exchange

Cet opérateur essaie d'échanger deux séquences de clients successifs entre deux tournées. Il est possible de limiter le nombre de clients par séquence échangée (non implémenté).

Cet opérateur s'exécute de la manière suivante :
— Choisir une arête (c_1, c_2) à éliminer ;
— Trouver une autre tournée grâce aux pp-voisins de c_1 ;
— On note c_4 le voisin considéré, et c_3 le client précédent c_4 sur la nouvelle tournée ;
— On échange c_1 et c_3 ;
— On essaie d'échanger deux autres clients entre les deux tournées.

Différentes variantes existent :
— First improvement (FI) : on considère à chaque fois le premier couple de client qui va produire un saving positif ;

— Best improvement (BI) : on teste tous les couples de clients entre les deux tournées et on considère celui qui maximise le saving.

On peut également choisir de parcourir les clients sur une tournée de différentes façons :
— Dans l'ordre (O) : les couples sont parcourus dans l'ordre à partir des clients échangés ;
— Dans un semi-ordre (SO) : on reprend le parcours des couples là où on l'avait interrompu à la dernière itération ;
— Aléatoirement (RD) : on tire aléatoirement l'ordre dans lequel on va parcourir les couples de clients.

Ces variantes et parcours sont aussi applicables en ce qui concerne le choix du voisin.

Lin-Kernighan

Utilisé en général pour résoudre le problème du voyageur de commerce (TSP). Il effectue une ptimisation intra-tournée (c'est-à-dire que la tournée considérée est améliorée indépendamment des autres). Cela consiste en une réorganisation des clients sur la tournée. On choisit k tel que LK ne dépasse pas k-opt au cours de son exécution. D'après l'article on peut prendre $k = 2$.

LK va donc s'exécuter de la manière suivante :
— On considère la tournée à améliorer ;
— On applique 2-opt sur la tournée (on échange deux clients sur la tournée) ;
— Tant qu'on a des améliorations on applique 2-opt sur la tournée ;
— On renvoie la tournée améliorée.

Différentes variantes existent :
— First improvement (FI) : on considère à chaque fois le premier couple de client qui va produire un saving positif ;
— Best improvement (BI) : on teste tous les couples de clients sur la tournée et on considère celui qui maximise le saving.

On peut également choisir de parcourir les clients sur une tournée de différentes façons :
— Dans l'ordre (O) : les couples sont parcourus dans l'ordre en partant du dépôt ;
— Dans un semi-ordre (SO) : on reprend le parcours des couples là où on l'avait interrompu à la dernière itération ;
— Aléatoirement (RD) : on tire aléatoirement l'ordre dans lequel on va parcourir les couples de clients.

Opérations spéciales

Il peut souvent arriver que la solution actuelle soit bloquée sur un optimum local.