TP3 - Recherche Operationnelle

Rapport - BELLEC Louison et BOUVARD Alexandre

Table of Contents

Descriptif

Le but du HVRP est de trouver le meilleur agencement possible de plusieurs tournées afin de livrer tous les clients.
On commence par génerer une solution initiale comme on genererait une solution du TSP (sans tenir compte des tournées). On va ensuite transformer cette solution en solution du HVRP en séparant la solution initiale en plusieurs tournées. On affinera par la suite les solutions avec la recherche locale.

Solution initiale

La solution initiale est une tournée géante qui comprend tous les clients. Nous avons 3 méthodes pour la générer.

Plus proches voisins

La méthode des plus proches voisins est la plus simple. Elle va simplement créer le grand tour en prenant le plus proche voisin du dernier point a chaque fois. A la fin on retourne au dépot de telle sorte a creer une boucle.

Plus proche voisins random

La méthode du prus proche voisins random est quasiment similaire seulement elle ajoute une part de hasard en choisisant aléatoirement parmi les 5 voisins les plus proches.
La solution finale obtenu par cette solution est souvent moins bonne que celle des plus proche voisins, mais la part d'aléatoire permet d'avoir une solution plus facile a optimiser car moins linéaire.

Heuristique - Enveloppe convexe

TODO Enveloppe convexe

Split

La méthode split permet de transformer notre solution initiale avec une seule tournée en une solution a plusieurs tournées en respectant les critéres de poids et de capacités des camions.

Recherche locale

La recherche locale permet d'améliorer une solution existante. Elle va faire avec une certaine probabilitée, soit le 2-OPT, soit le 2-OPT inter-tournée, ou soit l'insertion. Cela permet de varier et d'atteindre des solutions plus diverses.

2opt

Le 2-opt va echanger deux segments entre eux. Il permet de "décroiser" deux segments. Pour cela, on va tester 2 a 2 des segments pour voir si une autre disposition donne une plus petite distance. Par exemple, avec les points ABCD, on teste si ACBD a une distance plus courte. Si c'est le cas, on effectue cet echange et on recommence.
Le 2-opt effectue le decroisement qui améliore le plus la solution.

Insertion

L'insertion a pour but de déplacer un point. Pour cela, on teste tout nos points et on essaie de les insérer partout. A la fin, on n'insere que le point qui nous fait le plus gros gain de distance.
L'insertion effectue l'insertion qui amélore le plus la solution.

Description algorithmique

Nous allons décrire en francais de manière très abstraite le fonctionnement des differents algorithmes utilisés.
Les details d'implémentation peuvent être lu directement dans le code.

Plus proche voisins

```
POUR chaque client, en partant du dépot:
    Calculer le plus proche voisins non visité pour ce client
    Marquer ce voisin comme visité
    Ajouter ce voisin dans la tournée
    Incrémenter le cout de la tournée
FPOUR

Ajouter le cout du retour au depot
```

Plus proches voisins randomisees

```
POUR chaque client, en partant du dépot, sans les 5 derniers:

    Calculer les 5 plus proches voisins non visités pour ce client
    Garder un voisin aléatoirement parmi les 5
    Marquer ce voisin comme visité
    Ajouter ce voisin dans la tournée
    Incrementer le coût de la tournée

FPOUR

POUR les 5 derniers clients:
    Ajouter à la tournée de façon aléatoire
FPOUR

Ajouter le cout du retour au depot
```

Split

```
depot = ajoute label par défaut

POUR chaque clients en partant du dépot:
    client = client actuel

    POUR chaque label de ce client:
        label = label actuel

        POUR chaque type de camion:
            le label pere = le label actuel
        FPOUR

        TQ il reste des types de camion pouvant faire une tournee:
            curClient = client que on va ajouter
            prevClient = client d'ou on vient

            SI c'est le premier client de la tournee:
                distance = [depot ; client ; depot]
                quantity = quantite de ce client
            SINON
                distance = distance + [prevClient ; client] - [prevClient ; depot] + [client ; depot]
                quantity = quantity + quantite de ce client
            FSI

            TQ on a pas fait tous les camions et que le camion actuel est capable de recevoir la quantite du client:
            (pour chaque type de camion qui peut recevoir la quantite du client suivant, en comptant le state actuel)
                SI le label pere est different de -1
                    SI il reste des camions pour ce type pour ce label:
                        SI c'est le premier client de la tournee:
                            On enleve 1 camion de notre type dans le nombre de camions restant
                        FSI

                        SI le label a ajouter est mieux ou incomparable a tous les autres labels existant dans ce client:
                            ajoute le label
                            on garde en memoire le label pere pour ce type de camion
                        SINON
                            le label pere = -1
                        FSI
                    FSI
                FSI
            FTQ
            On enleve les camions qui n'ont pas pu faire la tournee
        FTQ
    FPOUR
FPOUR

on creer une tournee
on ajoute la distance initiale du client au depot
TQ on est pas au depot
    pere = on recupere le pere
    on ajoute le client courant a la tournee
    SI le courant et le pere ont des nombres de camions restant differents
        on ajoute le retour au depot a la distance
        on ajoute la tournee dans la liste de tournee
        on augmente la cout total avec le cout de la tournee
        on creer une nouvelle tournee
        on ajoute la distance initiale du depot au client
    SINON
        on augmente la distance
    FSI
FTQ
```

Recherche locale

```
POUR nombre d'itération:
    Generer Random entre 0 et 1
    SI random est inferieur a 0.3:
        POUR toutes les tournees:
            2opt
        FPOUR
    SINON SI random est inferieur a 0.6:
        POUR toutes les tournees:
            insertion
        FPOUR
    SINON:
        2optInter
    FSI
FPOUR
```

2 OPT

```
POUR chaque point i de la tournée sauf les 2 derniers:
    POUR chaque autres points j de la tournée non adjacent:
        On calcule la distance originelle, soit l'addition de la distance [i ; i + 1] et la distance [j ; j + 1]
        On calcule la nouvelle distance en cas de swap, soit l'addition de la distance [i ; j] et la distance [i + 1 ; j + 1]
        SI la nouvelle distance est plus faible que l'ancienne:
            La nouvelle différence est l'ancienne distance - la nouvelle
            SI la nouvelle différence est plus grande que la meilleure difference:
                La meilleure différence devient la différence actuelle
                On stocke i et j
            FSI
        FSI
    FPOUR
FPOUR
SI la meilleure diff est superieur a 0:
    On échange i + 1 et j
    On inverse l'ordre de tous les points entre i + 1 et j non compris
    On met a jour la nouvelle distance de la tournée
    On retourne vrai pour indiquer qu'on a pu faire un 2-opt
FSI
On retourne faux
```

Insertion

```
POUR chaque point i en commencant a 1:
    On prend les deux point adjacent a i soit h et j.
    On calcule la distance gagnee en elevant le point i et en reliant directement les deux points h et j soit [H ; I] + [I ; J] - [H
    POUR chaque points adjacents autre que i:
        On calcule la distance perdue en rajoutant notre point
        SI la distance gagnée et superieur  a la distance perdue:
            On stocke la meilleure diffèrence
            On stocke quel point ou doit inserer
            On stocke ou on doit inserer ce point
        FSI
    FPOUR
FPOUR

SI la meilleure diff est superieur a 0:
    On insére i apres le point ou on doit insérer
    On supprime i de l'endroit original
    On met a jour la nouvelle distance de la tournée
    On retourne vrai pour indiquer qu'on a pu faire une insertion
FSI

On retourne faux
```

SwitchBack

```
POUR chaque tournee:
    POUR chaque elements de la tournee:
        On ajoute l'element dans le nouveau tour geant
    FPOUR
FPOUR
```

Meta heuristique - GRASP

```
POUR un nombre d'itérations donné:
    on retransforme la meilleure solution en tour géant
    on génère un nombre donné de voisins(on echange au hasard deux points)
    POUR chacun de ces voisins:
        On split le nouveau tour géant
        On lui applique la recherche locale
        SI la solution est meilleure que le meilleure solution
            la meilleure solution devient la solution actuelle
        FSI
    FPOUR
    SI la meilleure solution est meilleure que la meilleure des meilleures:
        La meilleure des meilleurs devient la meilleur
    FSI
FPOUR
On retourne la meilleure des meilleures solutions
```