Projet Stage L3:
Application de mise en relation de transporteurs avec des clients

Responsabilité:
Calcul des tournées des transporteurs

Élève:
Tommy Anfosso

Table des matières

1- Analyse du besoin

1.1- CDC

Objectif du projet :

Proposer un service web permettant de mettre en relation des transporteurs avec des clients souhaitant effectuer un transport d'un point A à un point B, tout en optimisant la tournée des transporteurs selon des critères évalués par un serveur de calcul (pouvoir servir tous les clients et minimiser la distance de parcours globale).
De ce fait, les transporteurs enregistrés se voient proposer des clients nécessitant leur service.

Côté client :
1)Un client effectue une demande de transport auprès du service web.
2)La demande est envoyée au serveur de calcul
3)Le serveur de calcul consulte les données de la base de données et fait tourner un algorithme visant à optimiser certains critères, qui retourne un transporteur (ou une liste triée de transporteurs).
4)Le transporteur en question reçoit une proposition de course pour ce client, et s'il l'accepte, l'état de la demande se met à jour dans la base de données.
//Le client pourra à tout moment consulter l'état de sa demande

Côté transporteur :
1)Un conducteur s'enregistre auprès du service web en indiquant ses horaires, sa zone couverte ainsi que sa géolocalisation actualisée fréquemment.
2)Il peut à présent recevoir des propositions de courses. En acceptant une demande faites par un client, le client sera notifié.
//Le transporteur pourra à tout moment consulter l'historique de ses courses effectuées

Responsabilité de chaque module constituant la solution finale :

-Serveur de calcul : Calculer une output à partir d'un algorithme prenant en compte les infos de la base de données.

-Serveur web : Interface et API permettant à un client d'effectuer une demande de transport ou de s'enregistrer en tant que transporteur. Un client pourra consulter l'état de sa demande.

-Base de données : Stocke les demandes de transport, les transporteurs et les trajets (historique).

Données à stocker :

-Demandes de trajet
    -Coordonnées de départ et d'arrivée
    -Fenêtre de temps

-Transporteurs
    -Zone couverte
    -Horaires

-Géolocalisation

//Correspond à un accord conclu entre un client et un transporteur
-Trajets
-Demande de trajet
-Transporteur
[-Horodatage début/fin course]

1.2- Problème et exemple simple

Description du problème

On a un ensemble de clients qui souhaitent se rendre d'un endroit A à B dans une certaine fenêtre de temps.
Nous connaissons la localisation/position des clients, les coordonnées de départ et d'arrivée.

Objectif :
Servir tous les clients (dans la mesure du possible) tout en minimisant la distance totale parcourue par le groupe de transporteurs.
→ Servir tous les clients > distance totale parcourues

Contraintes :
-Un transporteur ne peut servir qu'un seul client à la fois, il doit finir sa course en cours avant de prendre un prochain client.
-Effectuer l'intégralité de la course dans la fenêtre de temps définie. Cela veut dire que le client doit être pris après le temps min et déposé avant le temps max.

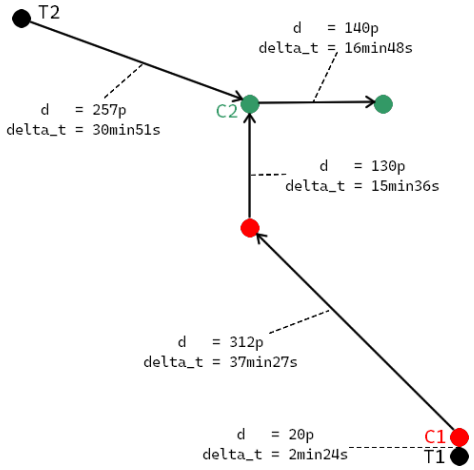
Un exemple simple

Soit une zone de 500 points par 500 (repère orthonormé), inscrit dans les points (1;1), (1;500), (500;1), (500;500).
Soit 2 transporteurs T1 et T2 situés respectivement sur (480;480) et (20;20).
Soit 2 clients C1 et C2 avec comme trajet respectif ((480;460),(260;240)) et ((260;110), (400;110)).
Soit t0 = 12h00 (temps actuel).
La course de C1 doit se déroulé dans la fenêtre de temps [12h30;13h30[.
La course de C2 doit de déroulé dans [13h45;14h10[.
On suppose que tous les trajets entre deux points sont parcourus linéairement.
On suppose que les véhicules se déplacent à 500 points à l'heure.
→ 50 points en 6 min, 125 points en 15 min, etc..
On suppose que les 2 transporteurs couvrent l'intégralité de la zone (ils peuvent se déplacer n'importe où dans le carré).

Une première solution pourrait être de dire que T1 serve C1 et T2 serve C2, les fenêtre de temps seraient en effet respectés et la distance total parcourue serait de 332+397=**729** points.

Hors, la meilleure solution est de confier les deux clients à T1, il respectera les fenêtres de temps et la distance total parcourue sera de **602** points. Cette solution minimise la distance totale parcourue.

T2 ne peut pas servir les deux clients dans les fenêtre de temps, donc cela n'est pas envisageable.



Pour résoudre ce problème avec des données simples, j'ai calculé chaque combinaison possible et comparé les distances totales parcourues par les transporteurs pour chaque tournée.

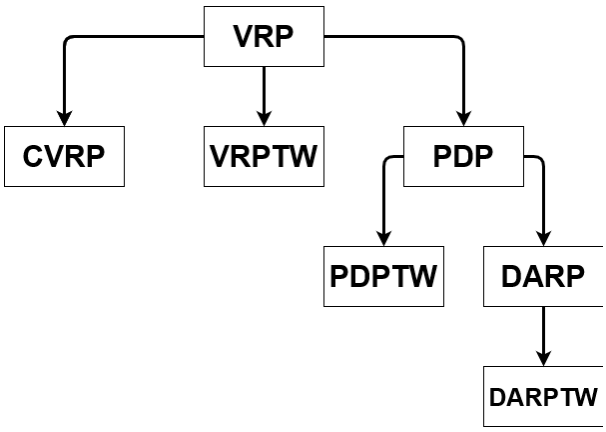
2- Recherche sur les VRP (Vehicle Routing Problems)

2.1- Catégories de VRP et modèle choisi

Catégories de VRP

Le VRP (Vehicule Routing Problem) est un nom générique donné à toute une classe de problèmes concernant l'optimisation de chemins à emprunter par une flotte de véhicules pour servir un ensemble de clients. Le VRP est une généralisation du TSP (Travelling Salesman Problem).

Basé sur le livre '*Smart Delivery Systems*' page 60, les trois variantes du VRP qui semble avoir le plus d'applications dans l'industrie sont le CVRP (Capacitated Vehicle Routing Problem), le VRPTW (Vehicle Routing Problem with Time Windows) et le GPDP (General Pickup and Delivery Problem) aussi appelé PDP.



-VRP :
Le chargement de marchandise s'effectue dans un même point de dépôt connu.

-CVRP :
On suppose que chaque véhicule possède une capacité maximum donnée (généralement la même pour tous les véhicules).

-VRPTW :
Des fenêtre de temps doivent être respectées pour les chargements et livraisons.

-GPDP ou PDP :
On considère deux types d'emplacement : les origines et les destinations. Chaque véhicule dans une flotte possède une capacité donnée, un point de départ et un point d'arrivée. Chaque chargement doit être transporté par un véhicule d'un ensemble d'origines à un ensemble de destinations sans transbordement à d'autres emplacements.

-GPDPTW ou PDPTW :
Des fenêtre de temps doivent être respectées pour les chargements et livraisons.

-DARP (Dial-A-Ride Problem) :
PDP se charge du transport de biens tandis que DARP se charge du transport de personnes.
Cette différence est essentiellement exprimée en termes de contraintes ou d'objectifs additionnels liés au confort de l'utilisateur (tel que le temps maximal de course, ..).

-DARPTW :
L'intégralité d'une course s'effectue dans une fenêtre de temps prédéfinie.

Modèle adopté

Notre problème ne correspond pas au VRPTW ou CVRP car dans ceux-ci chaque transporteur doit revenir à un point d'origine (dépôt) pour récupérer la 'marchandise' à transporter.

Le PDP avec fenêtre de temps permet de résoudre notre problème car désormais les points d'origines et de destinations varient en fonction de la livraison. De plus, un transporteur est obligé de finir sa livraison avant de passer à une autre.

Mais le modèle correspondant le plus à notre problème est le DARP avec fenêtre de temps car adapté au transport de personnes. Elle ajoute des contraintes ou objectifs liés au confort de la personne transportée (tel que le temps maximal de course).

J'opterai donc pour le DARPTW dynamique avec plusieurs véhicules.
Notre modèle est dynamique car nous souhaitons pouvoir traiter les demandes de transport au fur et à mesure qu'elles arrivent.

2.2- Techniques de résolution

Les méthodes de résolution exactes n'étant pas adaptées aux cas du vrai monde (à cause du nombre élevé de données à prendre en compte), nous résoudrons le problème avec des méta-heuristiques qui nous permettrons de trouver des solutions plus rapidement, sans trop négliger la qualité de celles-ci.

Technique 1, algorithme de recherche tabou (DARP statique) :

Cordeau et Laporte (*2003b*) proposent un algorithme de recherche tabou (Tabu Search) pour le DARPTW statique avec véhicules multiples. Les fenêtres de temps sont considérés à l'origine ou à la destination en fonction du type de demande (entrante ou sortante). Le voisinage pris en compte dans la recherche tabou est défini en déplaçant une demande vers un autre itinéraire. Le meilleur déplacement possible sert à générer une nouvelle solution en place. Les déplacements inverses sont déclarés tabous.
Cependant, un critère d'aspiration est défini, de telle sorte que les déplacements tabous qui fournissent une meilleure solution, tout en respectant les autres solutions déjà construites par ce même mouvement, puissent constituer une nouvelle solution en place. La solution initiale est construite aléatoirement, en respectant néanmoins les contraintes de précédence.

Des solutions pour des instances allant jusqu'à 295 demandes ont été trouvées.

Technique 2, algorithme génétique (DARP statique) :

Bergvinsdottir propose un algorithme génétique, il sépare la partie de clustering et de cheminement. Le clustering de clients est résolu à l'aide d'un algorithme génétique tandis que le cheminement est résolu par une heuristique modifiée de voisin le plus proche dans l'espace-temps.

Les instances résolues sont les mêmes que Cordeau et Laporte (*2003b*).

Technique 3, recherches tabous parallèles (DARP dynamique) :

Attanasio approche le problème avec pour objectif d'accepter autant de demandes que possible tout en satisfaisant les contraintes du problème. Pour résoudre le problème, il utilise une stratégie parallèle de l'heuristique de recherche tabou précédemment utilisée par Cordeau et Laporte (2003b) dans le cas statique du problème.

Différentes stratégies de parallélisation sont testées avec des instances allant jusqu'à 144 clients.

Référence :
Cordeau JF, Laporte G (*2003b*) *A tabu search heuristic for the static multi-vehicle dial-a-ride problem*. Transport Res B-Meth 37:579–594

3- Modèle linéaire statique

Dans un but pédagogique, nous élaborerons dans un premier temps un modèle linéaire et statique.

Formulation mathématique du modèle :

Ce modèle est une adaptation du PDP (Pickup and Delivery Problem) avec fenêtres de temps, il s'inspire du DARP (Dial A Ride Problem) et prend pour base la formulation du VRPTW proposé par Cordeau.

Ce problème est modélisé par un graphe G = (V,E) où V représente un ensemble de sommets et E l'ensemble des arcs (i,j) entre chaque couple de sommets appartenant à V.

L'ensemble V comprend les sous-ensembles S, pour les points de départ des véhicules, P, pour les points de chargement (origines), et D, pour les points de déchargement (destinations). Chaque origine est appairée à une destination pour former une course (transport d'un client d'un point de départ à un point d'arrivée).

K est un ensemble de véhicules à disposition pour effectuer des courses, il contient m éléments.
L'ensemble S contient m éléments (autant que de véhicules).
L'ensemble P et D contiennent chacun n éléments.
L'ensemble V contient N éléments, soit m+2*n éléments.

L'ensemble V est indicé de telle sorte :

$$\left\{ \underbrace{v_1, \dots, v_m}_{\substack{départ\ véhicule : s_i, \\ s \in S}}, \underbrace{v_{m+1}, \dots, v_{m+n}}_{\substack{origine : p_i, \\ p \in P}}, \underbrace{v_{m+n+1}, \dots, v_N}_{\substack{destination : d_i, \\ d \in D}} \right\}$$

c_{ij} est la matrice contenant la **distance** en mètres séparant le sommet i du sommet j.

t_{ij} est la matrice contenant la **durée** en minutes nécessaire pour aller du sommet i au sommet j.

e_i est le tableau contenant les temps **au plus tôt** pour commencer le service au sommet i.

l_i est le tableau contenant les temps **au plus tard** pour commencer le service au sommet i.

B_i^k est la matrice contenant les temps de début de service au sommet i par le véhicule k.

Les valeurs de temps sont exprimées en minutes passées après minuit, cette valeur est comprise entre 0 et 2879.
(Ex : 00h00 → 0 ; 12h00 → 720 ; 23h59 → 1439 ; 02h00 le lendemain → 1560)

x_{ij}^k est la matrice tridimensionnelle binaire indiquant si oui ou non le véhicule k parcourt l'arc (i,j).

Les entrées (constantes) sont :

- m : nombre de véhicules
- n : nombre de clients à servir
- c : matrice des distances entre chaque paire de sommets
- t : matrice des durées de parcours entre chaque paire de sommets
- e : tableau des temps de début de service au plus tôt pour chaque sommet

- l : tableau des temps de début de service au plus tard pour chaque sommet

Les sorties (variables du modèle) sont :

- x : tournée de chaque véhicule
- B : temps de début de service pour chaque sommet, véhicule (optionnel)

Fonction objective :

$$min \sum_{k \in K} \sum_{i \in V} \sum_{j \in V} c_{ij} x_{ij}^{k} \qquad (01)$$

Contraintes :

$$\sum_{k \in K} \sum_{j=i+n} x_{ij}^{k} = 1 \qquad \forall i \in P \qquad (02)$$

$$\sum_{k \in K} \sum_{i \in V} x_{ij}^{k} = 1 \qquad \forall j \in P \cup D \qquad (03)$$

$$x_{ij}^{k}(B_i^k + t_{ij}) \leqslant B_j^k \qquad \forall i, j \in P \cup D, k \in K \qquad (04)$$

$$(\sum_{k \in K} \sum_{i \in V} x_{ij}^{k}) - (\sum_{k \in K} x_{j,j+n}^{k}) = 0 \qquad \forall j \in P \qquad (05)$$

$$B_i^k \leqslant B_{i+n}^k \qquad \forall i \in P, k \in K \qquad (06)$$

$$e_i \leqslant B_i^k \leqslant l_i \qquad \forall i \in P \cup D, k \in K \qquad (07)$$

01 : Minimiser la distance totale parcourue par les véhicules.
02 : Chaque paire (origine,destination) doit être traversée 1 fois par 1 véhicule.
03 : Pour chaque paire (origine,destination), il y a 1 véhicule entrant.
(Permet de prendre en compte les trajets allant d'une destination à la prochaine origine et un point de départ véhicule vers une première origine)
04 : Le temps d'arrivée au sommet j doit être inférieur à la somme du temps de départ au sommet i et de la durée de parcours de (i,j). **expression quadratique**
05 : Le véhicule ayant la responsabilité de prendre un client à un sommet d'origine doit également faire le trajet de ce sommet d'origine vers le sommet destination correspondant.
06 : Le temps prévu pour l'origine doit être inférieur au temps prévu pour la destination.
07 : Les fenêtres de temps doivent être respectées.

Pour étendre le problème au DARP, nous pourrions ajoutons comme contrainte un temps de course maximum (ici représenté par L_i) :

$$B_{i+n}^k - (B_i^k + d_i) \leq L_i \qquad \forall i \in P$$

4- Codage du modèle avec julia

4.1- Packages utilisés

-JuMP v0.21 : Le langage de modélisation JuMP nous permettra d'implémenter notre modèle mathématique.

-GLPK v0.13 : Nous utiliserons l'optimisateur de GLPK, qui est libre, pour résoudre le problème linéaire.

4.2- Implémentation du modèle avec JuMP

Soit N le nombre total de sommets :

```
N = 2*n+m
```

```
@variable(model, x[1:N,1:N,1:m], Bin)
@variable(model, 0 <= B[1:N,1:m] <= 2879, Int)
```

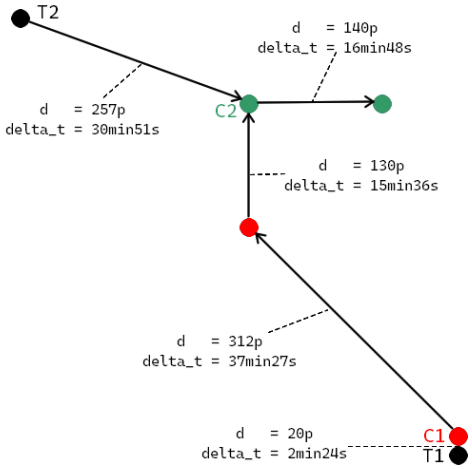
```
@objective(model, Min, sum(c[i,j]*x[i,j,k] for i in 1:N, j in 1:N, k in 1:m))
```

```
@constraint(model, [i in m+1:m+n], sum(x[i,j,k] for k in 1:m, j = i + n) == 1)
@constraint(model, [j in m+1:N], sum(x[i,j,k] for i in 1:N, k in 1:m) == 1)
@constraint(model, [j in m+1:m+n],
      sum(x[i,j,k] for i in 1:N, k in 1:m) - sum(x[j,j+n,k] for k in 1:m) == 0)
@constraint(model, [i in m+1:m+n, k in 1:m], B[i,k] <= B[i+n,k])
@constraint(model, [i in m+1:N, k in 1:m], e[i] <= B[i,k] <= l[i])
```

4.3- Résolution de l'exemple simple avec GLPK

Rappel de l'exemple simple

Soit C1 et C2 les clients à servir et T1 et T2 les transporteurs disponibles, nous avions calculé à la main que la meilleure tournée était de laisser le transporteur T1 se charger des deux clients, pour une distance totale de 602.



Résoudre avec *julia*

Données prises de l'exemple précédent, certaines valeurs sont volontairement grande pour remplir les trous.

```
m = 2
n = 2

c = [10000 10000 20 10000 10000 10000;
  10000 10000 10000 257 10000 10000;
  20 10000 10000 312 312 10000;
  10000 257 10000 10000 130 140;
  10000 10000 312 130 10000 10000;
  10000 10000 10000 140 10000 10000]

t = [1440 1440 2 1440 1440 1440;
  1440 1440 1440 31 1440 1440;
  2 1440 1440 37 37 1440;
  1440 31 1440 1440 16 17;
  1440 1440 37 16 1440 1440;
  1440 1440 1440 17 1440 1440]

e = [0, 0, 750, 825, 0, 0]
l = [2879, 2879, 765, 840, 809, 849]
```

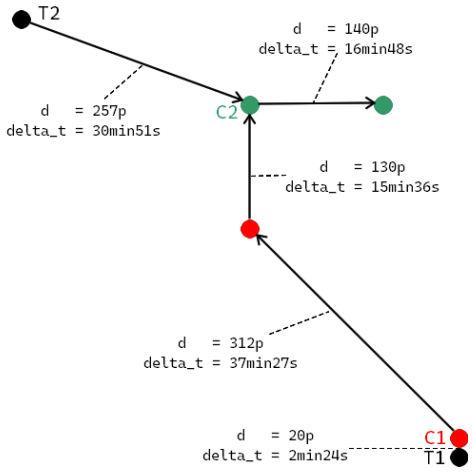
Résultat :

obj value = 602
obj times = Int16[0 0; 0 0; 750 750; 825 825; 750 750; 825 825] à ne pas considérer pour l'instant
res = Bool[0 0 1 0 0 0; 0 0 0 0 0 0; 0 0 0 0 1 0; 0 0 0 0 0 1; 0 0 0 1 0 0; 0 0 0 0 0 0]
Bool[0 0 0 0 0 0; 0 0 0 0 0 0; 0 0 0 0 0 0; 0 0 0 0 0 0; 0 0 0 0 0 0; 0 0 0 0 0 0]

Interprétation résultat :

Pour une distance totale parcourue de 602, le véhicule k_1 va emprunter les arcs (1,3), (3,5), (4,6) et (5,4).

Son chemin sera donc : (1→ 3→ 5→ 4→ 6),
soit (T1→ C1→ C1.2→ C2→ C2.2) si on se réfère au dessin.



5- Interfaçage avec les autres modules

Afin que l'application de calcul puisse être employée au sein de la plateforme web, il était nécessaire de trouver un format dans lequel les données en entrée et en sortie puissent être implicitement convertibles en types Javascript. C'est pourquoi ont été mises en place les classes entités 'Input' et 'Output' ainsi qu'un parseur capable de convertir les types primitifs julia en Javascript (format JSON) et inversement.

Pour ce qui est de la communication, j'ai choisi de faire un serveur TCP en julia, indépendant du serveur web, avec lequel il est possible d'envoyer des requêtes contenant une 'Input'. Le serveur traite alors les données reçues et renvoie une 'Output' au format JSON.
Le protocole TCP étant très répandu, il est très facile de trouver une implémentation de client TCP en Javascript. Pour ce qui est de la mise en forme des données, Javascript est nativement capable de convertir un objet en chaîne JSON et inversement.

Le serveur est capable de traiter plusieurs demandes simultanément et n'est contrôlable que depuis la session de terminal l'ayant démarré.

6- Amélioration des performances de julia

Une sysimage peut être considéré comme une session julia sérialisée, où certaines fonctions ont été pré-compilées.
Lors de l'ouverture d'une session julia, une sysimage par défaut est automatiquement utilisée permettant l'accès rapide aux fonctions et libraries de la STL.

Afin de réduire le temps de compilation des libraries GLPK et JuMP (ainsi que des fonctions que nous avons écrites), nous créons une sysimage personnalisée que nous utilisons à la place de celle par défaut lors du lancement d'une session julia.

Le gain de temps au lancement du serveur est très grand, il nous fallait autrefois attendre 40 secondes à chaque reboot mais désormais cela ne prend plus qu'une seconde. Le fichier de 200Mo peut être généré à l'aide du script 'sysimage/build.sh' qui suit les instructions de pré-compilation de 'sysimage/precompile.jl'.

Source : https://julialang.github.io/PackageCompiler.jl/dev/devdocs/sysimages_part_1/

7- Bilan

7.1- Travaux non aboutis

-Reformuler la contrainte quadratique en linéaire pour pouvoir l'intégrer au modèle JuMP

-Faire en sorte que le modèle puisse retourner une solution même dans le cas où les clients ne peuvent pas tous être servis (maximiser le nombre de clients servis en plus de la distance totale parcourue)

-Compléter les séries de tests dans le répertoire 'test' pour pouvoir évaluer la qualité de l'algorithme et localiser d'éventuels problèmes.

7.2- Pistes d'amélioration

-Faire une image Docker basée sur l'image officielle julia en remplaçant la sysimage par défaut pour tirer les meilleures performances possibles de julia et offrir une solution « cross-platform ».

-Travailler sur les méta-heuristiques telles que les recherches tabous pour pouvoir résoudre des instances plus grandes et de manière plus efficace.