Dial-A-Ride Problem

Clara Gros
Pauline Vaillant
Clémence David
Camil Zarrouk
Marouane Abia

March 2020

Contents

1 Présentation du problème

1.1 Motivation

Ce rapport vise à présenter un modélisation du problème DARP suivi d'une implémentation qui a pour but de délivrer une solution faisable, e.g. satisfaisant toutes les contraintes, et optimale (vis à vis des objectifs détaillés dans la section 1.3) tout en garantissant un temps de calcul raisonnable.
Pour ce faire, dans un premier temps, nous avons défini notre modèle en terme de PLNE puis, à l'aide d'un solver, nous avons pu obtenir pour une petite instance de données la solution optimale.Notre deuxième challenge a été de mettre au point une heuristique de construction enrichie d'une méta heuristique pour pouvoir répondre à des problèmes de plus grande taille.

1.2 Introduction

Récemment, il y a eu un regain d'intérêt pour les systèmes de partage répondant à la demande, motivé par le souci de l'environnement mais également par les nouveaux développements des technologies qui permettent de nouveaux modes de fonctionnement. Les systèmes de transport public sont toujours confrontés aux objectifs contradictoires d'une exploitation rentable et d'un service de haute qualité - en assurant la livraison des clients depuis/vers leur origine/destination souhaitée au moment voulu. Les services réguliers de bus ou de train peuvent transporter un un grand nombre de passagers (et donc un bon rapport coût-efficacité), mais qui voyagent sur des lignes fixes à des heures régulières vers lesquelles les passagers doivent adapter leurs plans de voyage en conséquence. Les services d'autobus réguliers ne sont souvent pas fournis (ou très rarement) pour les communes rurales parce que le coût de fonctionnement du service ne peut être justifié par la faible demande. Les services de taxi quant à eux, offrent des services de porte à porte sur demande, mais le coût de ce service est élevé, tant en termes monétaires qu'en termes d'impact sur l'environnement.
Ainsi, les services de transport en commun à la demande, qui combinent rentabilité et personnalisation, ont suscité un grand intérêt. Les véhicules de transit à la demande n'ont pas d'itinéraires ou d'horaires fixes, mais sont expédiés en fonction des demandes de transport reçues ; contrairement aux services de taxi, les passagers peuvent partager l'utilisation des véhicules et ne peuvent donc pas être conduits sur l'itinéraire le plus direct de leur origine à leur destination.
Les travaux sur la résolution des DARP sont toujours motivés par des applications réelles. Chacune d'entre elles traite de diverses caractéristiques réalistes qui conduisent à des contraintes ou à des objectifs spécifiques et apporte des informations complémentaires. Une application traditionnelle est celle des services à but non lucratif pour les personnes âgées et handicapées, dont l'objectif est souvent la minimisation des coûts. Les contraintes opérationnelles comprennent le temps de trajet et d'attente, les fenêtres de temps de ramassage et de livraison, la capacité du véhicule et la disposition des équipements dans le véhicule. Certains systèmes utilisent aussi des flottes hétérogènes. D'autres peuvent per-

mettre des transferts d'un véhicule à l'autre. Avec des acteurs différents, ces systèmes ont souvent des objectifs multiples (et parfois contradictoires), ce qui nécessite des modèles multicritères.
La suite de ce chapitre explicite les objectifs et les contraintes spécifiques de notre problème DARP.

1.3 Objectifs

Notre problème vise à trouver une solution optimale vis à vis de deux objectifs $c1$ et $c2$ avec $c1 \succcurlyeq c2$:

(c1) Affecter le plus de courses possibles aux chauffeurs

(c2) Minimiser le temps total de trajet pour les chauffeurs

On se retrouve avec un problème bi-objectif et l'on va se ramener à un problème mono-objectif (voir équation 0) en utilisant la méthode lexicographique sachant que le critère 1 est prioritaire sur le critère 2.

1.4 Contraintes

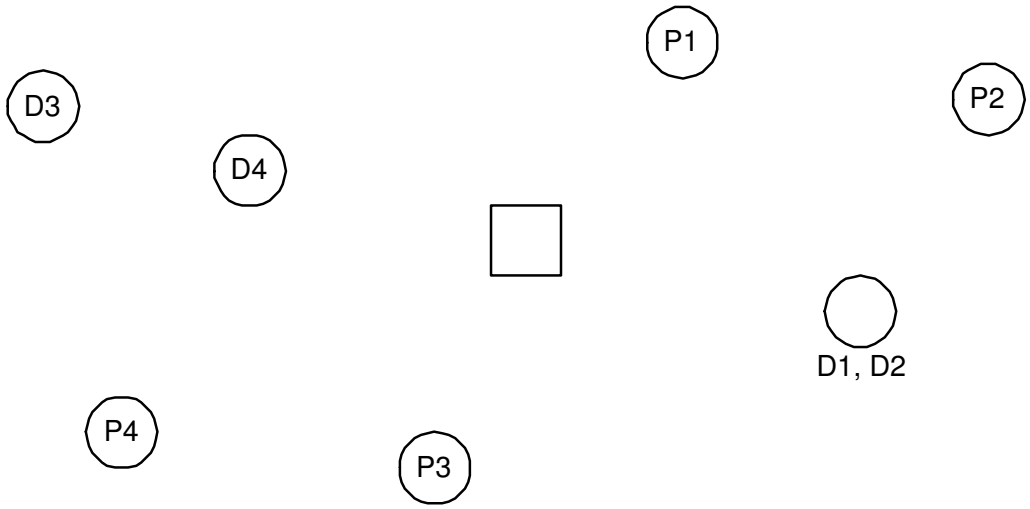
Nous présentons ci-dessous une liste exhaustive des contraintes littérales imposées par le problème :

- Revenu maximal de chaque chauffeur
- Capacité des véhicules
- Durée maximale de chaque course
- Fenêtres de temps pour chaque prise et dépose
- Début et fin de service de chaque chauffeur

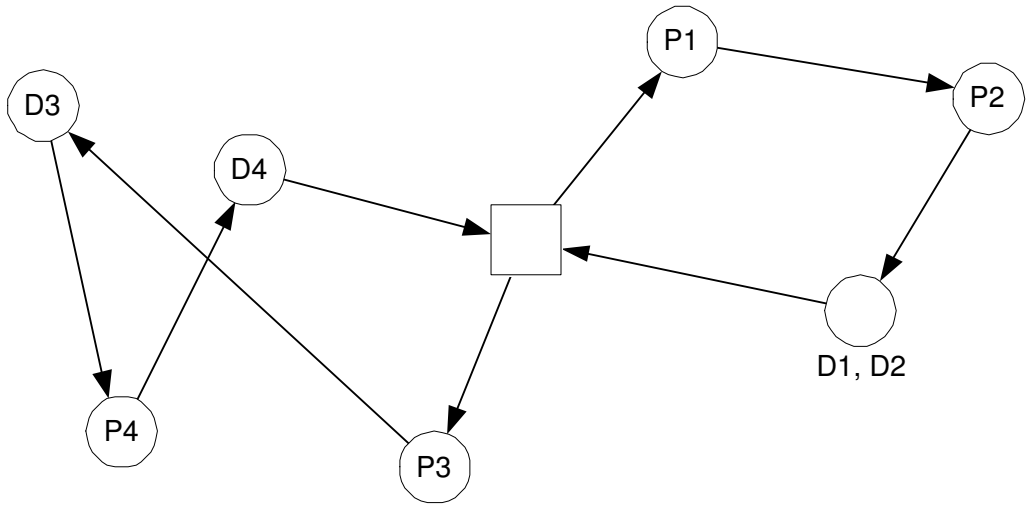
1.5 Illustration du problème

La page suivante présente une petite instance et une solution possible à un problème de DARP constitué d'une flotte de deux véhicules et de quatre courses [4].

DARP example



Possible solution:



2 Modèle mathématique

2.1 Notations

2.1.1 Modélisation par un graphe

Sur une journée, notons $n \in \mathbb{N}$ le nombre total de requêtes. Une requête (course) peut être faite par un ou plusieurs utilisateurs. Le DARP peut donc être défini sur un graphe orienté [2].

$$G = (V, E), \text{ où } V = P \cup D \cup \{0, 2n+1\}, P \in [\![1, n]\!], \text{ et } D \in [\![n+1, 2n]\!]$$

Les sous-ensembles P et D contiennent respectivement les nœuds de ramassage (pick up) et de dépôt (drop off), tandis que les nœuds 0 et $2n+1$ représentent les dépôts d'origine et de destination des véhicules de la flotte (confondus dans notre cas). À chaque utilisateur i est donc associé un nœud d'origine i et un nœud de destination $n+i$. L'ensemble des arcs est défini comme suit : $E = \{(i,j) : i = 0,\ j \in P \text{ ou } (i,j) \in P \cup D,\ i \neq j \text{ et } i \neq n+j \text{ , ou } i \in D,\ j = 2n+1\}$. Concrètement, une arête $(i,j),\ i = 0,\ j \in P$ traduit le fait qu'un bus qui part vide de l'entrepôt passe forcément en premier par une station de prise pour récupérer des gens. De même, une arête $(i,j), i \in D,\ j = 2n+1$ traduit le fait qu'un bus qui rentre à l'entrepôt en fin de tournée ne peut pas passer juste avant par une station de prise pour prendre des gens. Enfin, les arêtes de type $(i,j), (i,j) \in P \cup D,\ i \neq j$ et $i \neq n+j$ représentent les chemins que peut emprunter un bus au cours de sa tournée. La condition $i \neq j$ signifie qu'il n'y a pas d'arête bouclant sur un même noeud : un chemin faisable va d'une station à une autre bien distincte. Quant à la condition $i \neq n+j,\ \forall (i,j) \in P$, elle traduit le fait que, pour une requête i donnée, passer d'abord par la station de dépôt $n+i$ puis par la station de prise i n'a pas de sens.

2.1.2 Paramètres du problème

- M représente le nombre total de chauffeurs (véhicules) de la flotte.
- $\forall (i,j) \in V^2$, t_{ij} représente le temps de trajet entre les stations i et j.
- $\forall i \in V$, e_i représente la durée d'exécution à la station i (e.g le temps d'arrêt à la station pour embarquer ou débarquer des gens). On pose par convention $e_0 = e_{2n+1} = 0$.
- $\forall i \in P$, m_i représente le temps de trajet maximal de la course i.
- $\forall i \in V$, q_i représente le nombre de personnes chargées dans le véhicule au nœud i. Il va de soi que $q_0 = q_{2n+1} = 0$ car aucune personne ne monte ni ne descend à l'entrepôt des véhicules et $\forall i \in P$, $q_i = -q_{n+i}$ car toutes les personnes qui montent dans le véhicule à la station i (requête i) descendent à leur station d'arrivée $n+i$.
- $\forall i \in P$, r_i représente le coût de la course i.

- $\forall k \in M$, C_k représente la capacité maximale du véhicule k.
- $\forall k \in M$, (L_k, U_k) représentent respectivement la date minimale (maximale) de départ (d'arrivée) du véhicule k à l'entrepôt.
- $\forall i \in V$, (l_i, u_i) représente la fenêtre de temps à la station i.
  Les fenêtres de temps à l'entrepôt (stations 0 et $2n+1$) représentent la date la plus tôt à laquelle le véhicule devrait partir de l'entrepôt pour servir les clients et la date la plus tard à laquelle il doit retourner au dépôt.
  Ainsi,
  $$l_0 = min(min_{i \in P}(l_i - t_{0i}), min_{k \in M}(L_k))$$
  $$l_{2n+1} = min(min_{i \in P}(l_i - t_{2n+1,i}), min_{k \in M}(L_k))$$
  signifie que nous prenons la première heure de départ nécessaire comme heure de début de la fenêtre temporelle au dépôt, et de même, on prend :
  $$u_0 = max(max_{j \in D}(u_j - t_{j,0}), max_{k \in M}(U_k))$$
  $$u_{2n+1} = max(max_{j \in D}(u_j - t_{j,2n+1}), max_{k \in M}(U_k))$$
  comme étant le dernier moment de la fenêtre temporelle au dépôt.
- $\forall k \in M$, R_k représente le revenu maximal que le chauffeur k peut toucher en une tournée.

2.2 Variables de décision

- $\forall (i,j) \in V^2$, $\forall k \in M$, x_{ij}^k illustre le fait que le véhicule k se déplace ou non du nœud i au nœud j.
  $$x_{ij}^k = \begin{cases} 1 \text{ si le chauffeur } k \text{ passe par } (i,j) \\ 0 \text{ sinon} \end{cases}$$
  Remarque : Si $(i,j) \notin E$, $x_{ij}^k = 0$.
- $\forall (i,k) \in V \times M$, u_i^k représente l'heure à laquelle le chauffeur k arrive à la station i.
- $\forall (i,k) \in P \times M$, m_i^k représente le temps de trajet total de la course i dans le véhicule k.
- $\forall (i,k) \in V \times M$, q_i^k représente la charge du véhicule k en partant de la station i.

2.3 Contraintes

2.3.1 Formulation des contraintes du problème

Chaque demande est réalisée au plus une fois : seulement un véhicule peut arriver à une station de prise donnée et il n'y repart que par un seul chemin.

$$\sum_{k \in M} \sum_{j \in V} x_{ij}^k \leq 1 \ \forall i \in P \tag{1}$$

Un même véhicule qui entre dans un nœud quitte obligatoirement le nœud.

$$\sum_{j \in V} x_{ij}^k - \sum_{j \in V} x_{ji}^k = 0 \ \forall i \in P \cup D, \forall k \in M \tag{2}$$

Les contraintes 3 et 4 garantissent que l'itinéraire de chaque véhicule k commence au dépôt d'origine et se termine au même dépôt, si le véhicule fait au moins une course (dans le cas où le véhicule ne quitte pas l'entrepôt, $x_{0,2n+1}^k = 1$).

$$\sum_{j \in V} x_{0j}^k = 1 \ \forall k \in M \tag{3}$$

$$\sum_{i \in V} x_{i,2n+1}^k = 1 \ \forall k \in M \tag{4}$$

Les nœuds d'origine et de destination sont visités par le même véhicule.

$$\sum_{j \in V} x_{ij}^k - \sum_{j \in V} x_{n+i,j}^k = 0 \ \forall i \in P, \forall k \in M \tag{5}$$

Visit time La cohérence des variables de temps est assurée par la contrainte 6 et la contrainte 7 impose des fenêtres de temps définies par chaque utilisateur.

$$u_j^k \geq (u_i^k + e_i + t_{ij}) x_{ij}^k \ \forall (i,j) \in V^2, \forall k \in M \tag{6}$$

$$l_i \leq u_i^k \leq u_i \ \forall i \in V, \forall k \in M \tag{7}$$

Ride time L' égalité 8 définit le temps de parcours de chaque utilisateur, qui est limité par les contraintes 9.

$$m_i^k = u_{n+i}^k - (u_i^k + e_i) \ \forall i \in P, \forall k \in M \tag{8}$$

$$t_{i,n+i} \leq m_i^k \leq m_i \ \forall i \in P, \forall k \in M \tag{9}$$

Load constraints La cohérence des variables de charge est assurée par la contrainte 10 et la contrainte 11 impose des contraintes de capacités définies pour chaque véhicule.

$$q_j^k \geq (q_i^k + q_j) x_{ij}^k \ \forall (i,j) \in V^2, \forall k \in M \tag{10}$$

$$max(0, q_i) \leq q_i^k \leq min(C_k, C_k + q_i) \ \forall i \in V, \ \forall k \in M \tag{11}$$

Vehicles tour constraints Les contraintes 12 et 13 imposent respectivement une date de début et une date de fin de la tournée d'un véhicule.

$$u_0^k \geq L_k \ \forall k \in M \tag{12}$$

$$u_{2n+1}^k \leq U_k \ \forall k \in M \tag{13}$$

Cost constraint La contrainte 14 limite le revenu maximal de chaque chauffeur.

$$\sum_{j \in V} \sum_{i \in P} r_i x_{ij}^k \leq R_k \ , \forall k \in M \tag{14}$$

Avoid loop on one station :

$$\sum_{k \in M} \sum_{i \in V} x_{ii}^k = 0 \tag{15}$$

2.3.2 Linéarisation des contraintes

La formulation des contraintes ci-dessus est non linéaire en raison des contraintes 6 et 10. En introduisant les constantes M_{ij} et Q_i^k, ces contraintes peuvent toutefois être linéarisées comme suit :

$$u_j^k \geq u_i^k + e_i + t_{ij} - M_{ij}(1 - x_{ij}^k) \ \forall (i,j) \in V^2, \ \forall k \in M \tag{6.1}$$

avec

$$M_{ij} = max(0, e_i + t_{ij} + u_i - l_j) \ \forall (i,j) \in V^2 \tag{6.2}$$

$$q_j^k \geq (q_i^k + q_j) - Q_i^k(1 - x_{ij}^k) \ \forall (i,j) \in V^2, \ \forall k \in M \tag{10.1}$$

avec

$$Q_i^k = min(C_k, C_k + q_i) \ \forall i \in V, \ \forall k \in M \tag{10.2}$$

2.4 Fonction objectif

Pour établir la fonction multi objectif, de la forme

$$\max \sum_{k \in M} \sum_{(i,j) \in P \times V} x_{ij}^k - \epsilon \sum_{k \in M} \sum_{(i,j) \in V^2} t_{ij} x_{ij}^k \tag{0}$$

il faut choisir la valeur du paramètre ϵ afin de garantir que les objectifs sont bien optimisés en priorisant le nombre de courses par rapport au temps de trajet. On choisit ϵ de sorte que $\epsilon \sum_{k \in M} \sum_{(i,j) \in V^2} t_{ij} x_{ij}^k \leq 1$. Pour se faire, on imagine le pire des cas où toutes les routes possibles seraient empruntées par un véhicule : c'est à dire le cas où le graphe est complet. Alors en prenant $\epsilon = 1/\sum_{(i,j) \in V^2} t_{ij}$, on s'assure que la quantité précédente sera toujours inférieure à 1. En effet, on

veut maximiser la quantité $n - \epsilon d$, avec n le nombre de courses et d la distance parcourue par l'ensemble des véhicules de la flotte. Si on note N le nombre de courses maximales et que l'on s'assure que $\epsilon d \leq 1$, alors on est sûr que pour une configuration où $n = N$, on aura $n - \epsilon d > N - 1$ et si $n \leq N - 1$, alors $n - \epsilon d \leq N - 1$. Cela montre bien que la meilleure configuration sera forcément une configuration où $n = N$. On est donc bien sûr de maximiser en priorité sur le nombre de courses. La fonction objectif devient :

$$\max \sum_{k \in M} \sum_{(i,j) \in P \times V} x_{ij}^k - \epsilon \sum_{k \in M} \sum_{(i,j) \in V^2} t_{ij} x_{ij}^k$$
$$\text{with } \left\{ \ \epsilon = 1 / \sum_{(i,j) \in V^2} t_{ij} \right.$$

2.5 Programme linéaire

$$\max \sum_{k \in M} \sum_{(i,j) \in P \times V} x_{ij}^k - \epsilon \sum_{k \in M} \sum_{(i,j) \in V^2} t_{ij} x_{ij}^k$$

$$\text{s.t.} \left\{ \begin{array}{ll}
\sum_{k \in M} \sum_{j \in V} x_{ij}^k \leq 1 & \forall i \in P \\
\sum_{j \in V} x_{ij}^k - \sum_{j \in V} x_{ji}^k = 0 & \forall i \in P \cup D, \forall k \in M \\
\sum_{j \in V} x_{0j}^k = 1 & \forall k \in M \\
\sum_{i \in V} x_{i,2n+1}^k = 1 & \forall k \in M \\
\sum_{j \in V} x_{ij}^k - \sum_{j \in V} x_{n+i,j}^k = 0 & \forall i \in P, \forall k \in M \\
u_j^k \geq u_i^k + e_i + t_{ij} - M_{ij}(1 - x_{ij}^k) \forall (i,j) \in V^2, & \forall k \in M \\
l_i \leq u_i^k \leq u_i & \forall i \in V, \forall k \in M \\
m_i^k = u_{n+i}^k - (u_i^k + e_i) & \forall i \in P, \forall k \in M \\
t_{i,n+i} \leq m_i^k \leq m_i & \forall i \in P, \forall k \in M \\
q_j^k \geq (q_i^k + q_j) - Q_i^k (1 - x_{ij}^k) & \forall (i,j) \in V^2, \forall k \in M \\
max(0, q_i) \leq q_i^k \leq min(C_k, C_k + q_i) & \forall i \in V, \forall k \in M \\
u_0^k \geq L_k & \forall k \in M \\
u_{2n+1}^k \leq U_k & \forall k \in M \\
\sum_{j \in V} \sum_{i \in P} r_i x_{ij}^k \leq R_k & \forall k \in M \\
\sum_{k \in M} \sum_{i \in V} x_{ii}^k = 0 & \\
x_{ij}^k \in \{0,1\} & \forall (i,j) \in V^2, \forall k \in M
\end{array} \right.$$

Remarque : Notre programme linéaire peut être encore simplifié en éliminant certaines variables de décision [2] ce qui permettrait de diminuer drastiquement le nombre de variables intervenant dans le programme informatique et donc le temps de calcul d'une solution exacte.

3 Revue des méthodes de résolution

Le problème DARP fait partie de la classe des problèmes dit NP-difficile. Pour ce type de problème, on distingue généralement deux types de méthodes de résolution :

- Les méthodes exactes permettant d'obtenir une solution optimale au problème posé
- Les méthodes approchées permettant d'obtenir une solution au problème posé sans pouvoir affirmer qu'elle est optimale.

La section 3.1 présente les résultats obtenus sur une micro instance. Cependant, étant donné que ce problème est NP-difficile, l'exploration de l'espace de recherche pour la résolution du problème croît de manière exponentielle avec la taille des instances. Il n'existe pas, à ce jour, d'algorithme polynomial pour la résolution de ces problèmes. C'est pourquoi l'utilisation d'algorithmes permettant de trouver une solution approchée en un temps raisonnable est courante. De tels algorithmes sont rapidement présentés dans la section 3.2.

3.1 Résolution exacte sur une petite instance

3.1.1 Etat de l'art

Les algorithmes exacts pour les DARP sont développés principalement sur la base du concept de "branch-and-bound" (BB). Les problèmes de programmation sont difficiles à résoudre pour des cas réalistes de DARP et les méthodes exactes ne sont pas capables, pour des instances d'une certaine taille, de fournir une solution optimale en un temps raisonnable. Le tableau 1 résume les cas les plus importants qui ont été résolus de manière optimale par des méthodes exactes. Ces derniers vont jusqu'à 8 véhicules et 96 demandes pour le DARP de base. Pour les autres variantes plus compliquées, les dimensions sont plus petites.

Type de problème	Algorithme et référence	Taille instance	CPU(s)
DARP	B&P&C (Gschwind and Irnich, 2015)	8/96	898.8
HDARP	B&C (Braekers et al., 2014)	8/96	Not reported
MD-HDARP	B&C (Braekers et al., 2014)	8/64	1331
DC-HDARP	B&C (Braekers et Kovacs, 2016)	4/40	578

Table 1: Les plus grandes instances résolues par des méthodes exactes

3.1.2 Notre solution

Pour cet exercice, nous disposions d'un petit jeu de données *day_data.json* constitué de 25 courses et de 8 chauffeurs. Pour trouver une solution exacte, nous avons utilisé un logiciel open-source basé sur Python appelé **Pyomo** qui supporte un ensemble diversifié de capacités pour la formulation et l'analyse de modèles d'optimisation. Conjointement, nous nous sommes servis du solver **Gurobi** pour pouvoir déterminer une solution optimale à notre problème avec notre petite instance. Le tableau 2 récapitule le résultat obtenu en termes de courses affectées, de distance totale parcourue par la flotte, et de temps de calcul nécessaire au solver pour déterminer la solution.

Instance size	Assigned bookings	Route cost	CPU (s)
8/25	25/25	66525	7653

Table 2: Caractéristiques de la solution optimale

Encore une fois, nous gardons un regard critique sur notre temps de calcul qui peut être drastiquement réduit en limitant le nombre de variables de décision dans le modèle mathématique.

3.2 Autres méthodes possibles de résolution

Contrairement aux méthodes de résolution exactes, les méthodes heuristiques nous permettent de trouver des solutions approchées pour des problèmes d'optimisation en des temps raisonnables. Ces solutions ne sont pas forcément optimales mais peuvent être obtenues dans un temps relativement correct. En général, ces méthodes permettent de réaliser un bon compromis entre le temps de calcul et la qualité de la solution. Pour les problèmes de type VRP (Vehicle Routing Problem), il existe un certain nombre d'heuristiques classiques évoquées dans la figure ci-dessous [1].

Dans la première grande famille des heuristiques de résolution, les heuristiques de construction - dites gloutonnes - permettent de générer rapidement une première solution à un problème de tournées de véhicules. Deux heuristiques connues sont : l'heuristique avec calcul d'économies (savings) et l'heuristique d'insertion présenté dans la section 4. Concernant les heuristiques à deux-phases, la construction d'une première solution est divisée en deux étapes consistant à regrouper des noeuds au sein de clusters et à calculer un trajet entre les noeuds d'un même cluster. Les méthodes de descente partent d'une solution initiale et tentent d'améliorer cette première solution en explorant son voisinage. La deuxième famille des méthodes approchées comprend les méta heuristiques. Les méta heuristiques tirent en particulier leur intérêt dans leur capacité à éviter les optima locaux, soit en acceptant une dégradation de la fonction objectif au cours de leur progression, soit en utilisant une population de solutions. Elles sont souvent inspirées d'analogies avec la réalité (physique, biologie, éthologie, ...). Un grand nombre de méta-heuristiques existent. Par exemple, le recuit simulé, qui est une méthode à solution unique que nous mettrons en place par la suite dans la section 5, est inspiré d'un processus utilisé en métallurgie.

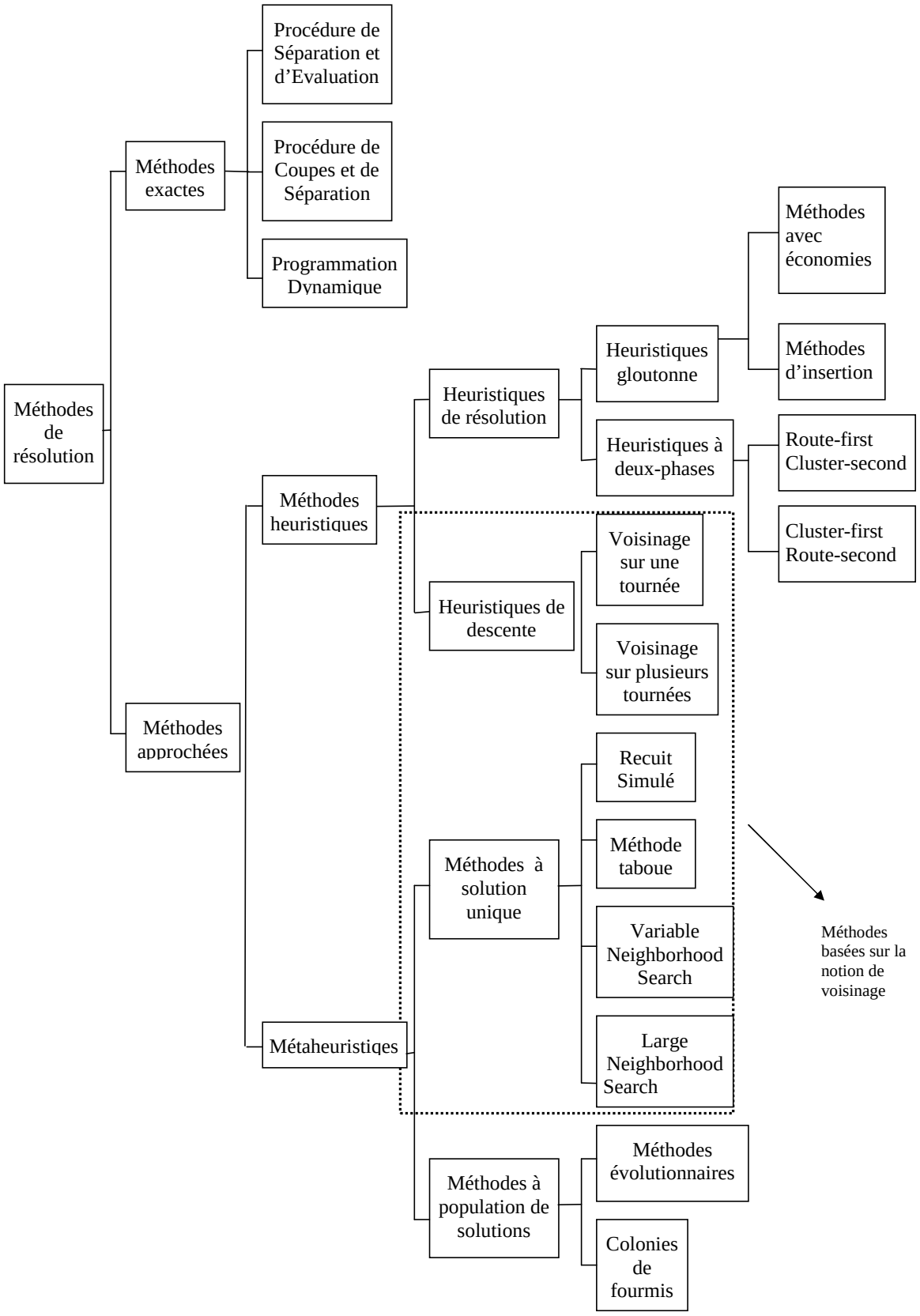


FIG. 1.6 – Récapitulatif des méthodes de résolution

4 Heuristique d'insertion

4.1 Principe général

Afin de trouver rapidement une solution et donc de pouvoir résoudre des problèmes avec des instances de plus grande taille, nous avons mis en place une heuristique d'insertion adaptée aux spécificités de notre problème. L'algorithme que nous proposons traite séquentiellement les requêtes, en insérant, si possible, chaque requête dans le planning d'un véhicule, de façon à minimiser une fonction de coût d'insertion, jusqu'à ce que toutes les requêtes aient été traitées. Les éléments centraux de l'algorithme sont les suivants : une recherche d'insertion de requête dans le planning de chaque véhicule et une étape d'optimisation pour trouver la meilleure insertion possible parmi toutes les insertions faisables dans chacun des véhicules [3].

L'insertion d'un client dans le planning d'un véhicule n'est possible que si elle ne viole aucune contrainte concernant le client en question mais aussi le chauffeur et les autres clients déjà assignés à ce véhicule. L'étape d'optimisation consiste à minimiser le "coût" supplémentaire dû à l'insertion du client dans le planning d'un véhicule.

Algorithm 1: Principe général de l'heuristique d'insertion

Data: N requêtes avec des fenêtres de temps
n véhicules avec des fenêtres de temps
une fonction de coût $COST$
Result: Une solution au problème DARP se rapprochant de l'optimal

```
begin
    Requêtes par PickUpTimeWindowBeginDate croissants ← REQ;
    for request i in REQ do
        for vehicle j in vehicles do
            Trouver tous les façons possibles d'insérer la requête i dans le
             planning du véhicule j;
            if feasible then
                Trouver la configuration qui minimise la fonction COST_j;
            else
                continue;
            end
        end
        if insérer i dans n'importe quel véhicule est impossible then
            i ← requête rejetée
        else
            j* ← j s.t COST_{j*} ≤ COST_j ∀j ∈ [[0, nb_vehicles]];
            assigner la requête i au véhicule j*;
            mise à jour du nouveau planning du véhicule j* ;
        end
    end
end
```

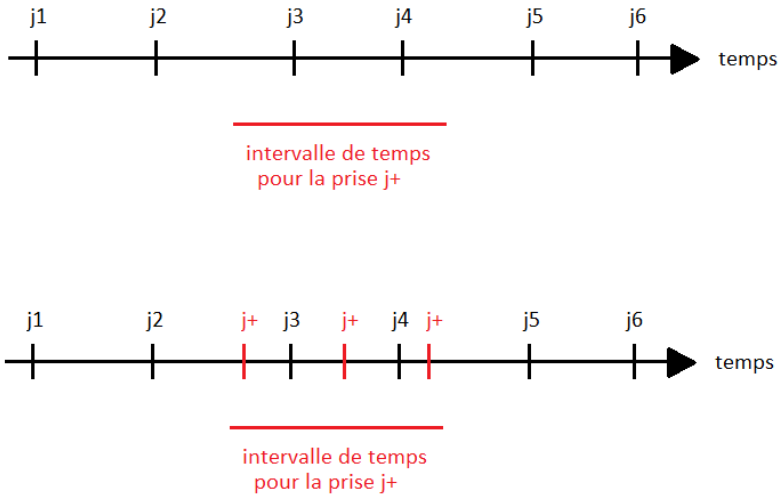
4.2 Faisabilité d'une insertion

Supposons qu'on ait un véhicule ayant un trajet déjà planifié constitué de plusieurs "jobs" $\{j_0, j_1, ...j_m\}$, un job étant soit une prise soit une dépose : pour chaque job on connaît aussi l'heure à laquelle il est effectué $\{h_0, h_1, ...h_m\}$.
On s'intéresse à l'insertion d'un client $j*$. On note $\{j_+, j_-\}$ la prise et la dépose de ce client.
On garde les notations précédentes :

- l_k : Date à partir de laquelle le véhicule peut effectuer un job j_k
- u_k : Date limite après laquelle le véhicule ne peut plus effectuer un job j_k
- e_k : durée d'exécution du job j_k
- $t_{k,l}$: durée du trajet entre la station du job j_k et du job j_l

Il faut d'abord choisir à quelle place on insère la prise j_+ dans le planning du véhicule par rapport aux autres jobs déjà prévus. Pour cela il suffit de comparer

l'heure à laquelle un job est effectué et l'intervalle de temps dans lequel la prise j_+ peut être faite. **Pour pouvoir insérer j_+ entre deux jobs, il faut que l'heure d'exécution d'au moins un des deux soit dans l'intervalle de temps de j_+** Cela se vérifie assez intuitivement avec une exemple : ici, on peut choisir d'insérer j_+ entre j_2 et j_3 , ou j_3 et j_4, ou j_4 et j_5.



Supposons qu'on insère j_+ entre j_p et j_{p+1}. Il faut ensuite choisir l'heure à laquelle cette prise est effectuée. Nous avons décidé de toujours choisir cette heure la plus tôt possible, donc nous supposons que le véhicule part de la station de j_p juste après le job j_p effectué. Si jamais le véhicule arrive trop tôt a la station de j_+, il attendra jusqu'à l_+ pour effectuer la prise de j_+. L'expression de l'heure à laquelle la prise j_+ est effectuée est donc :

$$h_+ = max(l_+, h_p + e_p + t_{p,+})$$

Il faut ensuite explorer les conséquences pour les autres clients, en calculant le retard induit par la prise j_+. L'expression de ce retard est :

$$\Delta = t_{p,+} + t_{+,p+1} - t_{p,p+1} + e_+$$

Ce retard risque d'impacter tous les jobs effectués après j_+ : cependant il est aussi possible que le véhicule fasse des pauses à certains moment de son planning, lorsqu'il arrive à une station avant que le client soit prêt. Si une

pause est plus grande que le retard, alors la pause sera diminuée et le retard sera annulé. Si le retard est plus grand que la pause, la pause sera annulé et le retard sera diminué du temps de la pause. On parcourt donc l'ensemble des jobs j_k suivant j_+ en mettant le retard à jour :

$$\Delta = max(0, \Delta - pause_k)$$

où $pause_k$ est la durée de la pause juste avant le job j_k.
Puis on prend en compte le retard :

$$h_k = h_k + \Delta$$

Il suffit ensuite ensuite de refaire exactement la même chose pour la dépose, en prenant en compte que celle ci aura forcement lieu après la prise. On obtient alors le nouveau planning après l'insertion du client $j*$.

Version 2 Dans une deuxième version de l'algorithme, nous avons opté pour une vision un peu différente pour vérifier si l'insertion respectait les contraintes d'intervalles de temps. Au lieu de calculer le retard causé par l'insertion de $j+$ ou de $j-$, nous recalculons l'ensemble des horaires à partir de $l+$. Cela peut sembler lourd en calcul mais cela permet d'être sûr que l'on n'oublie aucune marge.

Nous avons donc deux versions de l'algorithme : $insertion\_v1$, basée sur le calcul de temps de pause et de retard, et $insertion\_v2$ qui recalcule tout l'itinéraire. Cette deuxième version est légèrement meilleure que la première, nous avions donc dû oublier certains cas de figure dans $insertion\_v1$.

La dernière étape est alors de vérifier que toutes les contraintes sont bien vérifiée, à savoir :

- Contraintes d'intervalle de temps pour les jobs
- Contraintes de durées maximales pour les clients
- Contraintes de charge maximale,salaire maximal et retour au dépôt pour le véhicule

Si toutes ces contraintes sont vérifiées, alors l'insertion est faisable.

4.3 Insertion optimale d'un nouveau client

En effectuant la démarche présentée dans la section 4.2, on peut ainsi déterminer dans un premier temps, toutes les façons faisables d'insérer un client dans un véhicule. Puis dans un deuxième temps, on élargit ceci à tous les véhicules de la flotte.
Le critère d'optimisation pour pouvoir enfin sélectionner la meilleure insertion possible se base sur la distance totale parcourue par la flotte. En effet, on considère que le nombre de courses assignées à un véhicule est maximisé par la façon dont on trie initialement la liste de clients et par le choix de toujours choisir l'heure de la prise d'un client la plus tôt possible.

4.4 Résultats

Le tableau 3 présente les caractéristiques de la solution obtenue par les deux algorithmes d'insertion (versions 1 et 2) pour les jeux de données *day_data*, *week_data* et *week_data_v*2.

	Instance size	Assigned bookings	Route cost	CPU (s)
Version 1 : Day data	8/25	22/25	31888	0.0040
Version 2 : Day data	8/25	23/25	31496	0.0043
Version 1 : Week data	40/147	145/147	197021	0.1790
Version 2 : Week data	40/147	147/147	208095	0.7797
Version 1 : Week data v2	40/147	125/147	176005	0.1398
Version 2 : Week data v2	40/147	131/147	190508	0.1135

Table 3: Caractéristiques d'une solution obtenue par l'algorithme d'insertion

Pour le jeu de données *week_data_v*2, la solution obtenue avec la version 2 amélioré de notre algorithme d'insertion permet de servir 131 clients sur 147 pour les données de la semaine. Ce résultat est très satisfaisant, puisque le temps de calcul est de l'ordre de la demi-seconde, c'est-à-dire un temps suffisamment cours pour l'utilisation que l'on souhaite faire de l'algorithme. De plus, cette solution va maintenant pouvoir être améliorée par une heuristique d'amélioration, décrite dans la partie suivante 5.

5 Heuristique d'amélioration : Simulated Annealing ou Recuit Simulé

Après avoir généré rapidement une solution faisable au problème DARP, il s'agit maintenant d'améliorer celle-ci en des temps de calcul raisonnables en cherchant à minimiser la fonction de coût. Pour cela, nous avons décidé d'implémenter une méta-heuristique de recuit simulé.

5.1 Principe général du recuit simulé

L'algorithme du recuit simulé s'inspire d'un processus physique dans lequel on chauffe, puis on refroidit des métaux afin de leur permettre de se rapprocher le plus possible de leur structure cristalline la plus stable, et donc, par cela de minimiser leur énergie. Ce processus rend le métal plus résistant.

L'algorithme du recuit simulé reprend cette idée générale. Dans l'algorithme, l'énergie s'apparente à la fonction à minimiser. Ce sera pour nous la fonction objectif définie plus haut dans ce document. Le principe de l'algorithme est donc de construire de nouvelles solutions en perturbant la solution existante. Une nouvelle solution qui permet de diminuer l'énergie sera toujours acceptée, en revanche, une solution qui augmente l'énergie de $E1$ vers $E2$ et donc qui

nous éloigne d'un optimum ne sera acceptée qu'avec une probabilité p. Le fait d'accepter des solutions moins bonnes avec une certaine probabilité a pour but d'échapper aux optima locaux et d'explorer l'espace des solutions le mieux possible. Dans la littérature, on trouve :

$$p = \begin{cases} 1 & \text{si} \quad E1 \geq E2 \\ e^{-\frac{E2-E1}{T}} & \text{si} \quad E1 \leq E2 \end{cases}$$

Cette probabilité dépend d'un paramètre T : la température. La température est un paramètre choisi par le décideur tout au long de l'algorithme. Initialement, la température est fixée à une valeur relativement élevée, ce qui traduit le fait que la probabilité d'accepter une solution moins bonne est plutôt grande. Cela permet au système de pouvoir explorer une vaste région de l'espace des solutions. On diminue ensuite la température, ce qui réduit la probabilité d'accepter une solution moins bonne et permet donc de converger vers un minimum local.

Dans l'application du recuit simulé au problème DARP, à chaque itération, l'algorithme doit réaliser une modification élémentaire de la solution. Pour cela, nous avons défini plusieurs façon de modifier un solution : La première méthode consiste à prendre au hasard un client parmi tous les clients (ceux qui ont été attribués, mais également ceux qui n'ont pas encore été attribués) et l'affecter à un véhicule au hasard. Il faudra alors vérifier, comme dans l'algorithme d'insertion, que ce client peut être ajouté à cette voiture, en respectant les contraintes. Pour cela, nous utiliserons les fonction déjà définie pour l'algorithme d'insertion. Cependant si les véhicules ont tous des emplois du temps très chargés, on risque de ne pas pouvoir ajouter de client en plus. La deuxième méthode évite ce problème, car elle consiste à échanger deux clients de deux véhicules différents, ou, un client dans un véhicule avec un client qui n'a pas été affecté. Cependant cette méthode ne permet pas d'augmenter le nombre de clients servis, qui est pourtant un critère important. Nous avons donc défini une troisième méthode, qui consiste a insérer un par un tous les clients non servis jusqu'à ce qu'il y en ait éventuellement un que l'on arrive à servir.

Pour définir l'énergie du système à tout instant, on utilisera l'opposé de la fonction objectif définie dans le programme linéaire. Ainsi pour une solution S,

$$E(S) = -\sum_{k \in M} \sum_{(i,j) \in P \times V} x_{ij}^k - \epsilon \sum_{k \in M} \sum_{(i,j) \in V^2} t_{ij} x_{ij}^k$$

$$\text{with } \epsilon = 1/\sum_{(i,j) \in V^2} t_{ij}$$

Algorithm 2: Principe général du recuit simulé

Data: S = So un S0 l'état initial donné par l'algorithme d'insertion
E(S) la fonction objectif qui calcule l'énergie
To la température intiale
Une loi de calcul de la température
iterMax un nombre d'itérations maximal
iter = 0 le nombre d'itérations
Result: Une solution au problème DARP se rapprochant de l'optimal

```
begin
    while Tmin ≤ T do
        S ← fluctuations(S);
        E1 ← E2;
        E2 ← E(S);
        if E1 ≥ E2 then
            p ← 1;
        else
            p ← e^{-(E2-E1)/T};
        end
        T mise à jour;
    end
end
```

L'avantage du recuit simulé, comparé à d'autres méta-heuristiques, est donc d'explorer une grande partie de l'espace des solutions afin d'éviter de tomber à la fin sur un minimum local qui serait beaucoup moins bon que le minimum global. L'inconvénient de cet algorithme est le nombre de paramètres à choisir. En effet, en plus de l'état initial, il faut également définir la température à chaque itération.

5.2 Détermination des paramètres

La performance du recuit simulé méta-heuristique présenté ci-dessus, dépend d'un certain nombre de paramètres qui doivent être fixés à l'avance. Ce jeu de paramètres constitué des trois variables suivantes :

- Température initiale $T0$
- Température minimale $Tmin$
- Taux de décroissance de la température λ

Détermination des bornes des paramètres Afin de pouvoir par la suite faire des analyses de sensibilité pour trouver des valeurs de température initiale et de taux de refroidissement optimal, nous devons trouver des intervalles dans

lequel faire évoluer nos valeurs de paramètres. Il convient de déterminer les températures T_{min}, T_{max} telles que $T0 \in [T_{min}, T_{max}]$. Pour se faire, nous sommes partis du principe que la probabilité d'accepter une solution moins performante que la solution initiale dans la première itération de l'algorithme, correspond à

$$e^{-\frac{E2-E1}{T0}} \text{ si } E1 \leq E2$$

$$\text{with } E2 - E1 = -\sum_{k \in M} \sum_{(i,j) \in P \times V} x_{ij}^{k(2)} - x_{ij}^{k(1)} + \epsilon \sum_{k \in M} \sum_{(i,j) \in V^2} t_{ij}(x_{ij}^{k(2)} - x_{ij}^{k(1)})$$

Or, la solution $S2$ (d'énergie $E2$) est obtenue à partir de la solution $S1$ (d'énergie $E1$) à partir d'une fluctuation élémentaire qui consiste soit à insérer un nouveau client, soit à échanger de véhicule un client déjà affecté. Ainsi $E2 - E1$ équivaut environ à ϵ. Donc si, lors de la première itération, on veut fixer une probabilité p d'accepter la solution lorsque elle est en réalité moins bonne que la solution, on peut poser :

$$T0 \simeq -\epsilon / ln(p)$$

En faisant varier la probabilité $p \in [0.01, 0.99]$ on obtient donc bien un intervalle de température, $T0 \in [T_{min}, T_{max}]$.
λ sera lui choisi arbitrairement à 0.99

5.3 Résultats

Après avoir testé les différentes méthodes de modification d'une solution, ainsi que des configurations ou plusieurs méthodes étaient utilisée en même temps, (en choisissant aléatoirement l'une d'entre elle à chaque étape), il s'est avéré que la méthode 1 (celle où on change un client de voiture) est la plus efficace pour améliorer la solution. La méthode 3 (voir si un client non servis peut être servi) peut aussi être utilisée en même temps que la méthode 1, mais comme elle prend plus de temps à réaliser, il vaut mieux ne pas la choisir trop souvent (on peut par exemple la sélectionner à chaque étape avec une probabilité de 0.1)

Le recuit simulé que nous avons implémenter améliore significativement notre solution fournie par l'algorithme d'insertion comme le montre le tableau 4. En effet, par exemple, pour le jeu de donnée *week_data_v2* le nombre de courses effectuées passe de 126 à 139.

	Instance size	Assigned bookings	Route cost	CPU (s)
Insertion (day)	8/25	23/25	31496	0.0043
SA (day)	8/25	25/25	28940	0.3235
Insertion (week)	40/147	147/147	208095	0.7797
SA (week)	40/147	147/147	166973	1.6089
Insertion (week2)	40/147	131/147	190508	0.1135
SA (week2)	40/147	135/147	176673	2.7432

Table 4: Comparaisons des solutions avec ou sans recuit simulé

6 Conclusion

Rappelons d'abord que l'objectif de ce projet était de mettre en place un modèle mathématique décrivant les objectifs ainsi que les contraintes qui se posent dans un problème de véhicule à la demande (DARP). A l'aide de ce modèle, il s'agit ensuite de construire une stratégie pour obtenir une solution la plus proche de l'optimal possible sans pour autant avoir des temps de calcul pharamineux. Pour ce faire, notre stratégie a d'abord été de mettre au point une heuristique gloutonne d'insertion séquentielle des bookings dans le planning de chaque véhicule pour pouvoir construire une solution viable. Dès l'implémentation de cet algorithme, nous avons fait des choix de tri de liste par exemple, pour obtenir une première solution déjà convaincante. Ensuite, afin de l'améliorer encore, nous avons mis au point une métaheuristique de recuit simulé ayant pour objectif de s'écarter des potentiels optima locaux en générant une solution à partir de la solution fournie par l'algorithme d'insertion. Le tableau 4 résume nos différentes solution en terme d'objectifs à atteindre et de temps de calcul.
En conclusion, si nous poursuivions le projet, il existe plusieurs voies d'amélioration. D'une part, comme expliqué plus haut, le modèle mathématique pourrait être simplifié en terme de variables de décision pour permettre de trouver des solutions exactes de petites instances en un temps moindre. D'autre part, concernant l'algorithme de recuit simulé, nous avions déjà créé une version améliorée basée sur le choix d'un opérateur plus efficace (qui consiste à créer une nouvelle solution en inversant deux clients). Cependant, il est aussi envisageable d'insérer un nouveau paramètre correspondant au nombre de clients à inverser pour construire la nouvelle solution.
Enfin, pour aller plus loin et répondre davantage à une situation concrète à laquelle est confrontée Yuso, il parait pertinent de considérer un cas stochastique où il existe des aléas sur les temps de trajet. Nous pourrions également nous placer dans le cas "dynamique" du problème en imaginant que les requêtes ne sont pas toutes disponibles en même temps mais "s'affichent" en temps réel, au compte goutte. Il faudrait alors essayer de construire une solution progressivement tout en essayant de rester proche de l'optimal.

References

[1] Frédérique Baniel : Prise en compte d'objectifs de stabilité pour l'organisation de collectes de déchets. Mémoire de D.E.A., Université de Toulouse, 11 2009. Section 1.6 : Résolution des problèmes VRP.

[2] Cordeau : A branch-and-cut algorithm for the dial-a-ride problem. *Operations Research*, 54(3):573–586, 2006.

[3] Harilaos N. Nigel H. Wilson Jang-Jei Jaw, Amedeo R. Odoni : A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Psaraftis of Cambridge*, pages 243–257, March 1984.

[4] Stefan Røpke : The dial a ride problem (darp). February 2005. CAOS.