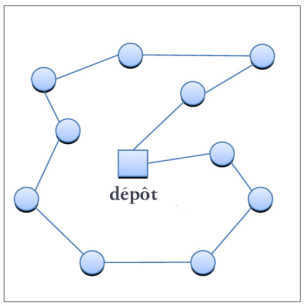
Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis
37200 Tours, France
Tél. +33 (0)2 47 36 14 14
www.polytech.univ-tours.fr

Projet Recherche & Développement
2015-2016

Nouvelles matheuristiques: implémentations et expérimentations

Application aux problèmes de tournée



Entreprise
Laboratoire Informatique

Tuteurs entreprise
Jean-Louis Bouquard

Étudiants
Yan LI (DI5)

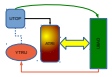
Tuteurs académiques
Jean-Louis Bouquard

26 septembre 2016

Liste des intervenants

Entreprise

Laboratoire Informatique
64 avenue Jean Portalis, 37200 Tours
http://polytech.univ-tours.fr



Nom	Mail	Qualité
Yan LI	yan.li.lynne@gmail.com	Étudiant DI5
Jean-Louis Bouquard	jean-louis.bouquard@univ-tours.fr	Tuteur académique, Département infomatique
Jean-Louis Bouquard	jean-louis.bouquard@univ-tours.fr	Tuteur entreprise

Avertissement

Ce document a été rédigé par Yan LI susnommé l'auteur.

L'entreprise Laboratoire Informatique est représentée par Jean-Louis Bouquard susnommé le tuteur entreprise.

L'école polytechnique de l'université François Rabelais de Tours est représentée par Jean-Louis Bouquard susnommé le tuteur académique.

Par l'utilisation de ce modèle de document, l'ensemble des intervenants du projet acceptent les conditions définies ci-après.

L'auteur reconnait assumer l'entière responsabilité du contenu du document ainsi que toutes suites judiciaires qui pourraient en découler du fait du non respect des lois ou des droits d'auteur.

L'auteur atteste que les propos du document sont sincères et assument l'entière responsabilité de la véracité des propos.

L'auteur atteste ne pas s'approprier le travail d'autrui et que le document ne contient aucun plagiat.

L'auteur atteste que le document ne contient aucun propos diffamatoire ou condamnable devant la loi.

L'auteur reconnait qu'il ne peut diffuser ce document en partie ou en intégralité sous quelque forme que ce soit sans l'accord préalable du tuteur académique et de l'entreprise.

L'auteur autorise l'école polytechnique de l'université François Rabelais de Tours à diffuser tout ou partie de ce document, sous quelque forme que ce soit, y compris après transformation en citant la source. Cette diffusion devra se faire gracieusement et être accompagnée du présent avertissement.

Pour citer ce document :
Yan LI, *Nouvelles matheuristiques: implémentations et expérimentations: Application aux problèmes de tournée*, Projet Recherche & Développement, Ecole Polytechnique de l'Université François Rabelais de Tours, Tours, France, 2015-2016.

```
@mastersthesis{
    author={LI, Yan},
    title={Nouvelles matheuristiques: implémentations et expérimentations: Application aux problèmes
        de tournée},
    type={Projet Recherche \& Développement},
    school={Ecole Polytechnique de l'Université François Rabelais de Tours},
    address={Tours, France},
    year={2015-2016}
}
```

Document mis en forme sous LaTeX
grâce à la classe de document `polytech.cls` V1.0 (2016/02/01)

Table des matières

Table des figures

Liste des tableaux

Introduction

Ceci est le rapport du Projet de Recherche & Développement que j'ai réalisé au cours de ma dernière année dans le cadre du cycle Ingénieur Informatique à Polytech Tours.

Ce projet intitulé "Nouvelles matheuristiques : implémentations et expérimentations" a été encadré par Monsieur Jean-Louis Bouquard.

Les objectifs de ce projet sont les suivants :

- D'abord je dois prendre connaissance de l'état de l'art pour comprendre les méthodes et les algorithmes existants. Cela nécessite la lecture d'un grand nombre d'articles.
- Puis je vais choisir un problème entre pleins de problèmes de tournée pour l'étude plus approfondie et le développement après.
- Ensuite je vais étudier et implémenter certaines variantes de matheuristique sur ce problème choisi en utilisant un solveur.
- Et je vais aussi essayer de proposer une nouvelle matheuristique ou améliorer une matheuristique existante pour ce problème.
- Enfin je dois effectuer des expérimentations numériques pour évaluer et apprécier ces différentes méthodes.

Au premier semestre, nous nous concentrons plutôt sur le Projet de Recherche, donc le travail réalisé porte plutôt sur l'étude bibliographique. Nous avons choisi le problème one-to-one m-PDTSP (*Multi-Commodity One-to-One Pickup-and-Delivery Traveling Salesman Problem*) pour une étude plus approfondie ainsi que pour le Projet de développement suivant.

Au deuxième semestre, nous avons proposé une nouvelle matheuristique avec plusieures variantes pour résoudre ce problème. Nous avons les implémentées et fait des expérimentations pour évaluer ces différentes méthodes.

Ce rapport détaille le travail que j'ai effectué tout au long de l'année 2015-2016, il se compose de deux grandes parties, la première partie présente la recherche, il contient dans un premier temps un état de l'art, pour définir ce qu'est une matheuristique et montrer comment nous pouvons l'utiliser pour résoudre des problèmes de tournée. Puis une étude plus approfondie pour le problème one-to-one m-PDTSP est menée, et nous présentons en détail la description et la modélisation de ce problème, ainsi que deux algorithmes existants pour résoudre le problème one-to-one m-PDTSP.

La deuxième partie porte sur les propositions, les implémentations et les expérimentations. D'abord, nous allons présenter en détail les algorithmes que nous avons proposé, puis comment nous avons fait les expérimentations et les résultats expérimentaux. Il y aura un autre chapitre pour présenter les téchniques et le planning du développement. Enfin, nous finirons avec un chapitre de conclusion.

Première partie

Projet de Recherche

1 État de l'art

1 Matheuristique

Qu'est-ce qu'une heuristique ?

Une heuristique est un terme qui signifie "l'art d'inventer, de faire des découvertes" , elle est approximativement dans le sens qu'elle fournit une bonne solution pour relativement moins d'efforts, mais elle ne garantit pas l'optimalité. Heuristiques sont des critères, méthodes ou principes pour décider qui, parmi plusieurs cours alternatifs d'action, promet d'être le plus efficace afin d'atteindre un objectif.

Qu'est-ce qu'une métaheuristique ?

Le mot métaheuristique est dérivé de la composition de deux mots grecs :

- heuristique qui signifie "trouver"
- meta qui est un suffixe signifiant "au-delà"

Ça veut dire que une métaheuristique est une heuristique dans un niveau supérieur. Une métaheuristique est un processus de génération itérative qui guide une heuristique subordonnée en combinant intelligemment différents concepts pour explorer et exploiter l'espace de recherche. Il existe un grand nombre de métaheuristiques différentes, allant de la simple recherche locale à des algorithmes complexes de recherche globale. Ces méthodes utilisent cependant un haut niveau d'abstraction, leur permettant d'être adaptées à une large gamme de problèmes différents.

Qu'est-ce qu'une matheuristique ?

Le concept de matheuristique est assez récent dans la recherche informatique, il n'existe pas une définition exacte, mais la notion la plus acceptée est : les matheuristiques sont des algorithmes heuristiques fabriqués par l'inter-opérations de métaheuristiques et la programmation mathématique (MP).

Comment faire une matheuristique ?

Certaines approches utilisant la programmation mathématique combinée avec des métaheuristiques ont commencé à apparaître régulièrement dans la littérature. Cette combinaison peut être effectuée de deux façons :

- Améliorer ou concevoir une métaheuristique utilisant la MP.
- Améliorer des techniques de MP connues en utilisant une métaheuristique.

Les matheuristiques sont souvent utilisées pour résoudre des problèmes d'optimisation difficiles (par exemple des problèmes dans les domaines de la recherche opérationnelle, de l'ingénierie ou de l'intelligence artificielle) pour lesquels on ne connaît pas de méthode classique plus efficace. Dans notre cas, nous allons étudier comment les appliquer aux problèmes de tournée de véhicules.

2 Problèmes de tournée

Le problème de tournées de véhicules (VRP) est un nom générique donné à une classe de problèmes comportant la visite des "clients" par les "véhicules". Le nom vient d'un problème pratique de base, où les gens essayent de fournir des clients qui sont géographiquement dispersés avec des marchandises en utilisant un certain nombre de véhicules à partir d'un dépôt commun. Ce problème est une extension classique du problème du voyageur de commerce, et il fait partie de la classe des problèmes NP-complets.

Généralement, le but d'un problème de tournée est de minimiser le coût ou le temps ou la distance de livraison des biens. Mais en fait, le VRP apparaît très fréquemment dans des situations pratiques non directement liées à la livraison des marchandises. Par exemple, la levée du courrier des boîtes aux lettres, le ramassage des enfants par des autobus scolaires, les tours de visite par un médecin, la livraison de linge etc. sont tous des VRP, où l'opération "livraison" peut être un ramassage, un ramassage et une livraison, ou d'autres opérations, et où les "marchandises" et "véhicules" peuvent prendre une variété de formes dont certaines peuvent même ne pas être d'une nature physique.

Compte tenu de l'énorme nombre de situations pratiques qui peuvent être vues comme des VRP, ce n'est pas un surprise de constater qu'un aussi grand nombre de contraintes et objectifs apparaissent dans ces problèmes. Nous avons toutes ces variantes en modifiant certaines contraintes ou l'objectif, dans Figure 1 [1], nous voyons une carte des plusieurs variantes classiques. Le VRP et ses variantes ont été un sujet considérable dans le domaine de recherche au cours de ces dernières années, principalement en raison du grand nombre de contraintes et des objectifs supplémentaires découlant des applications dans le monde réel.

Traditionnellement, les méthodes de résolution pour les VRP ont été classées en trois groupes : méthodes exactes, heuristiques et métaheuristiques. Plus récemment, les matheuristiques sont apparues comme une quatrième option. Dans la section suivante, nous allons choisir certains problèmes VRP classiques pour voir schématiquement comment on peut résoudre ces problèmes avec une matheuristique.

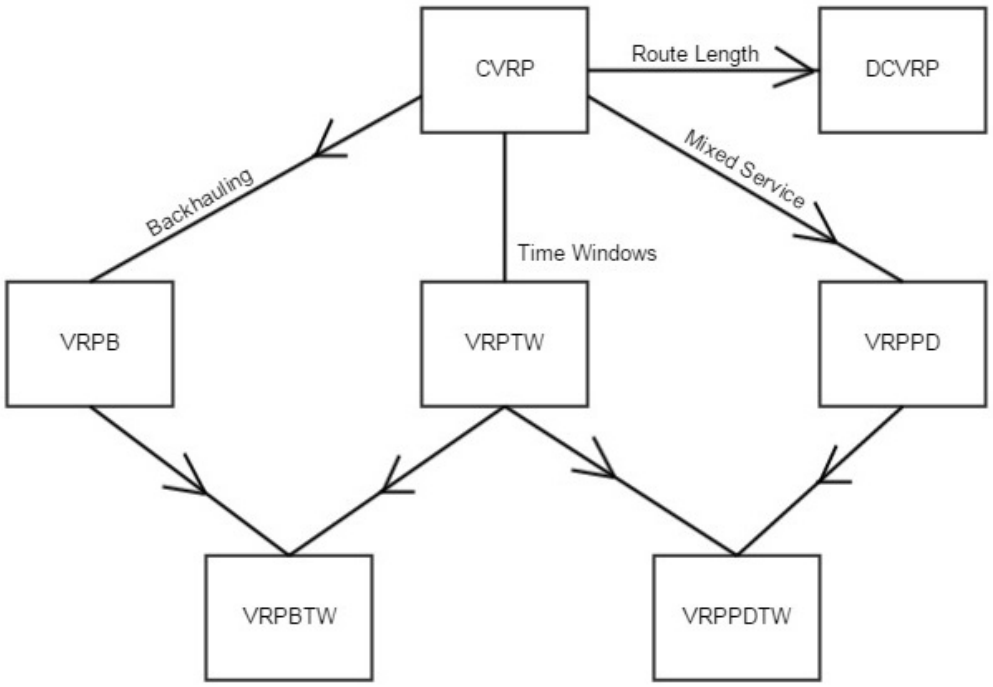


Figure 1 – *Une carte des variantes du VRP*

1. Figure 1 tiré de Wikimedia Commons[WWW1], *free media repository*

CVRP : *Capacitated Vehicle Routing Problem*

DCVRP : *Distance-constrained Capacitated Vehicle Routing Problem*

VRPB : *Vehicle Routing Problem with Backhauls*

VRPTW : *Vehicle Routing Problem with Time Windows*

VRPPD : *Vehicle Routing Problem with Pickup and Delivery*

VRPBTW : *Vehicle Routing Problem with Backhauls with Time Windows*

VRPPDTW : *Vehicle Routing Problem with Pickup and Delivery with Time Windows*

3 Aperçu de matheuristiques proposées pour problème de tournée

En raison de progrès des méthodes exactes et de la technologie du matériel, plusieurs modèles de mixte programmation linéaire en nombres entiers (MILP) peuvent être résolus à l'optimalité ou à proximité de l'optimalité dans une durée raisonnable. Cela a encouragé un nombre de chercheurs à concevoir des heuristiques qui intègrent des phases où des modèles de programmation mathématique sont résolus. C'est ce que nous appelons matheuristiques.

Nous pouvons classifier les matheuristiques proposées en trois grandes familles en analysant une grande variété d'approches et de problèmes.

1. Approches basées sur Set-Partitioning/Covering

Nous pouvons également dire Approches basées sur la génération de colonnes, elles ont été largement utilisées avec succès pour la résolution des problèmes de tournée. Ces algorithmes utilisent une formulation basée sur un partitionnement ensembliste (set-partitioning), où une variable binaire est associée à chaque circuit possible (colonne). En raison du nombre exponentiel de variables, on fait d'abord une génération de colonnes pour créer un sous-ensemble des circuits possibles, puis on résout le modèle Set-Partitioning sur ce sous-ensemble en utilisant un solveur.

2. Heuristique d'amélioration

Les matheuristiques appartenant à cette classe utilisent des modèles de programmation mathématique pour améliorer une solution trouvée par une approche heuristique. Ils sont très fréquents, car nous pouvons combiner le modèle de programmation mathématique avec toutes sortes de heuristique. Nous voyons souvent les deux utilisations suivantes :

a) Une approche "one-shot" où un modèle MILP est résolu une seule fois dans le but d'améliorer une solution réalisable trouvée par une heuristique.

b) Des approches avec des modèles MILP pour l'optimisation locale où un modèle MILP est utilisé pour l'optimisation locale, ce sont comme des opérateurs de voisinage à l'intérieur d'une procédure de recherche locale.

3. Approches de décomposition

En général, dans une approche de décomposition, un problème est divisé en sous-problèmes plus petits et plus simples, et puis une méthode de résolution spécifique est appliquée à chaque sous-problème. Dans ces matheuristiques, certains ou tous ces sous-problèmes sont résolus grâce à des modèles de programmation mathématique à l'optimalité ou suboptimalité.

La littérature récente est principalement concentrée sur les approches basées sur la génération de colonne et les heuristiques d'amélioration. Dans les deux cas, il existe un grand nombre de contributions couvrant une grande variété de problèmes. Cela prouve que ces approches sont efficaces et peuvent être facilement adaptées aux différents problèmes. Dans la section prochaine, nous allons voir trois exemples d'application, dont le premier exemple *A matheuristic for the truck and trailer routing problem* [19] appartient à la première famille, tandis que la deuxième *A matheuristic approach for the Pollution-Routing Problem* [11] combine les deux car il comprend une procédure d'amélioration utilisant la programmation en nombres entiers (IP) sur une formulation "Set-partitioning".

Les approches de décomposition sont encore appréciées dans les problèmes complexes qui impliquent différents types de décisions, cela est dû au fait qu'il est naturel et simple de décomposer ces problèmes en sous-problèmes. L'exemple *A matheuristic approach for solving a home health care problem* [1] appartient à cette classe.

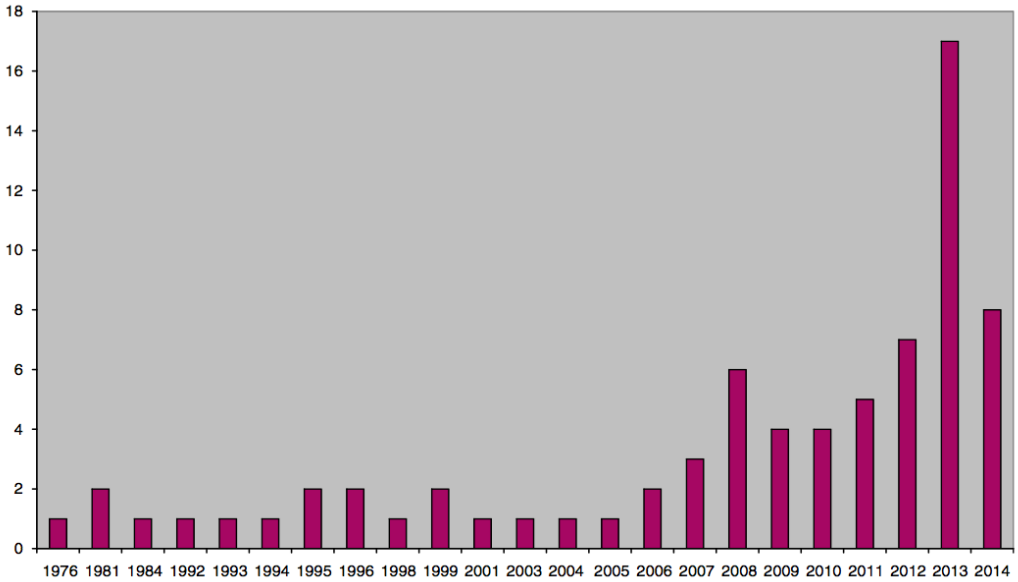


Figure 2 – *Nombre d'articles par an*

La Figure 2 [2] indique le nombre d'articles consacrés aux matheuristiques pour des problèmes de tournée chaque année jusqu'à 2014. Nous avons également compté les articles utilisant une méthode similaire, mais avant l'apparition du concept de matheuristique. Nous pouvons voir que l'intérêt pour cette technique a eu une croissance remarquable au cours des dernières années.

2. Données tirées de l'article [2]

4 Exemples d'utilisation de matheuristiques

4.1 A matheuristic for the truck and trailer routing problem

Dans le problème TTRP(*truck and trailer routing problem*) [19], une flotte de camions et remorques sert un ensemble de clients. Certains clients avec des contraintes d'accessibilité doivent être servis simplement par un camion, tandis que d'autres peuvent être servis par camion ou par un véhicule complet (un camion tirant une remorque). Le classique TTRP initialement introduit par Chao [6].

Les applications du monde réel de la TTRP apparaissent dans les opérations de distribution et de collecte dans les régions rurales et les villes. Par exemple, dans certains pays européens, la collecte du lait est effectuée par un petit camion avec remorque de grande capacité. Certaines fermes ne sont pas accessibles par les gros véhicules, de sorte que la remorque doit être détachée sur les routes principales avant de visiter ces fermes. Gerdessen [9] a signalé deux autres applications de TTRP aux Pays-Bas. La première application concerne la distribution d'alimentation animale dans les régions rurales, tandis que la seconde émerge dans le contexte de la distribution de produits laitiers.

4.1.1 Définition du problème

Les données :

m_t : le nombre de camions

m_r : le nombre de remorque (Nous supposons $m_r < m_t$)

$N = \{1,...n\}$: les sommets, c'est l'ensemble des clients

0 : le sommet 0 représente le dépôt

q_i : chaque client $i \in N$ a une demande $q_i > 0$

Q_t : la capacité de camion

Q_r : la capacité de remorque

c_{ij} : la distance entre les deux sommets $i, j \in N \cup 0$

N_t : sous-ensemble des clients qui sont accessibles uniquement par un camion

N_v : sous-ensemble des clients qui sont accessibles par un camion ou par un véhicule complet

Un trait distinctif de l'TTRP est que les emplacements des véhicule-clients (les clients dans N_v) peuvent être utilisées pour garer la caravane avant de servir les camion-clients (les clients dans N_t). Cette possibilité donne lieu à des itinéraires complexes avec un trajet principal de premier niveau et un ou plusieurs trajets de second niveau comme indiqué dans la Figure 3 [3].

L'objectif :

En somme, l'objectif du TTRP est de trouver un ensemble de chemins de distance totale minimale de sorte que chaque client est visité par un véhicule compatible ; la demande totale des clients visités sur une route ne dépasse pas la capacité du véhicule attribué ; et le nombre de camions et de remorque nécessaires ne dépasse pas m_t et m_r respectivement.

3. Figure 3 tiré de l'article [9]

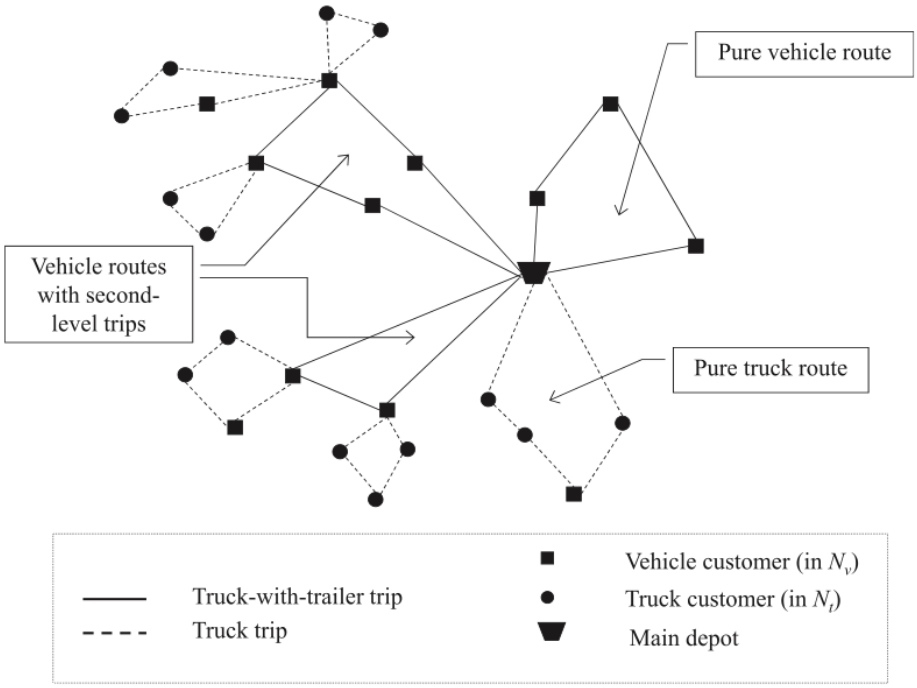


Figure 3 – *Types de routes dans une solution de TTRP*

4.1.2 Une formulation set-partitioning du TTRP

Comme nous pouvons voir dans la Figure 3, il y a trois types de route dans une solution de TTRP. Nous définissons :

Ω : ensemble de circuits faisables dans un TTRP

$\Gamma \subseteq \Omega$: ensemble de circuits camions purs

$\Psi \subseteq \Omega$: ensemble de circuits véhicules purs

$\Pi \subseteq \Omega$: ensemble de circuits à deux niveaux

Les variables :

a_{ij} : un paramètre binaire, il prend la valeur 1 si le client $i \in N$ est visité sur un circuit camion pur $j \in \Gamma$, sinon il prend la valeur 0

b_{ik} : égale 1 si le client $i \in N_v$ est visité sur un circuit véhicule pur , 0 sinon

e_{il} : égale 1 si le client $i \in N_v$ est visité sur un circuit à deux niveaux, 0 sinon

x_j : variable binaire, il prend la valeur 1 si le circuit $j \in \Gamma$ est sélectionnée dans la solution du problème, sinon il prend la valeur 0

y_k : égale 1 si le circuit $k \in \Psi$ est sélectionnée dans la solution, 0 sinon

z_l : égale 1 si le circuit $l \in \Pi$ est sélectionnée dans la solution, 0 sinon

c_r : représente le coût (e.i. la distance totale) de tous les circuits $r \in \Omega (= \Gamma \cup \Psi \cup \Pi)$

Donc nous avons modélisé ce problème par un MILP, vu comme un problème de partitionnement. La fonction objective est :

$$z = \sum_{j \in \Gamma} c_j x_j + \sum_{k \in \Psi} c_k y_k + \sum_{l \in \Pi} c_l Z_l \tag{1}$$

Avec des contraintes :

$$\sum_{j\in\Gamma} a_{ij}x_j + \sum_{k\in\Psi} b_{ik}y_k + \sum_{l\in\Pi} e_{il}Z_l = 1, \quad \forall i \in N_v \tag{2}$$

$$\sum_{j\in\Gamma} a_{ij}x_j + \sum_{l\in\Pi} e_{il}Z_l = 1, \quad \forall i \in N_t \tag{3}$$

$$\sum_{j\in\Gamma} x_j + \sum_{k\in\Psi} y_k + \sum_{l\in\Pi} Z_l \leqslant m_t \tag{4}$$

$$\sum_{k\in\Psi} y_k + \sum_{l\in\Pi} Z_l \leqslant m_r \tag{5}$$

La fonction objective Equation 1 minimise la distance totale de la solution. Equation 2 assure que chaque véhicule-client est visité exactement une fois ; Equation 3 assure que chaque camion-client est visité exactement une fois soit par une pure camion route soit par une route avec un deuxième niveau. Equation 4 et Equation 5 imposent une limite supérieure sur le nombre de camions et de remorques utilisées.

4.1.3 Matheuristique proposée

En fait, pour les petites instances, cette formulation peut être résolue directement par un solveur à l'optimalité, mais pour les plus grandes instances, nous avons un nombre exponentiel de circuits possibles, c'est pas efficace de résoudre le modèle directement par un solveur. Donc ils proposent une matheuristique en deux phases. Dans la premiere phase, il génère un sous-ensembles de circuits, et dans la deuxième phase, au lieu de résoudre la formulation sur tous les circuits possibles, on résout cette formulation sur ce sous-ensemble.

Phase 1

Nous utilisons une méthode GRASP × ILS pour remplir un sous-ensemble de circuits (colonnes) $\overline{\Omega}$.

1. GRASP(*Greedy randomized adaptive search procedure*), une tournée géante T qui visite tous les clients est calculée en utilisant une version randomisée de la bien connue heuristique plus proche voisin pour le problème du voyageur de commerce. A partir de l'entrepôt, à chaque itération, nous ajoutons un client après la dernière noeud i de la tournée émergeante.

2. Une solution du problème est dérivée de T par une procédure de séparation de tournée *Split*. (Nous devons séparer la tournée T en plusieurs circuits parce que nous avons plusieurs camions.)

3. Améliorer cette solution en utilisant ILS (*Iterated Local Search*). L'ILS proposé tente d'améliorer une solution S en effectuant itérativement quatre étapes :

- Utiliser la procédure *Concat* pour obtenir une tournée géante T de la solution S
- Échanger au hasard p paires de clients de T pour obtenir une nouvelle tournée géante T'
- Dériver une nouvelle solution S' en appliquant le procédure de séparation (*plit*) à T'
- Appliquer une VND (*variable neighborhood descent*) à S'

4. Mettre les circuits améliorés de la solution dans un sous-ensemble de circuits

Le processus **1-4** est répété pour créer plus de circuit.

Phase 2

Nous résolvons le MILP Equation 1 – Equation 5 sur ce sous-ensemble $\overline{\Omega}$ au lieu de sur tous les circuits dans le graphe. Le solveur sera beaucoup plus efficace avec cette plus petite instance.

4.2 A matheuristic approach for the Pollution-Routing Problem

Cet article [11] présente le problème "Pollution-Routing" (PRP). C'est un VRP avec des considérations environnementales, récemment introduit dans la littérature par Bektas et Laporte (2011) dans leur article [Transportation Research Part B : Methodological 45 (8), 1232–1250] [3]. L'objectif est de réduire les coûts opérationnels et environnementaux tout en respectant les contraintes de capacité et des fenêtres de temps de service.

4.2.1 Description du problème

Les données :

G = (V, A) : un graphe complet et orienté

V = {0, 1, 2, ..., n} : l'ensemble des sommets

A = $\left\{(i,j) \in V^2, i \neq j\right\}$: l'ensemble des arcs

0 : représente l'entrepôt où il y a une flotte de m véhicules identiques avec une capacité de Q

V – {0} : des clients

q_i : un client i a une demande q_i non-négative pour un seul produit

τ_i : durée de service τ_i et une fenêtre du temps spécifiée $[a_i, b_i]$ pour le service

Nous supposons que $q_0 = 0$ et $\tau_0 = 0$.

d_{ij} : chaque arc $(i,j) \in$ A représente une possibilité de voyage du noeud i à j pour une distance d_{ij}

R : l'ensemble de tous les circuits possibles

$\sigma \in \mathbf{R}, \sigma = (\sigma_1, ..., \sigma_{|\sigma|})$: un circuit commence et se termine à l'entrepôt, i.e. $\sigma_1 = 0$ et $\sigma_{|\sigma|} = 0$.

$f_{\sigma_i\sigma_{i+1}}$: le chargement du véhicule sur le circuit σ lorsque il voyage sur l'arc (σ_i, σ_{i+1})

$[v_{MIN}, v_{MAX}]$: les vitesses possibles pour les véhicules

Les variables :

v_{ij} : la vitesse sur chaque arc (i,j), elle est entre $[v_{MIN}, v_{MAX}]$

R' $\subset$ **R** : un sous-ensemble des circuits

L'objectif :

Le PRP vise à trouver une matrice de vitesse $(v)_{ij}$ et un sous-ensemble de circuits R' (tel que $|R'| \leq m$, m : le nombre de véhicules) pour servir tous les clients, tout en minimisant les coûts environnementaux et opérationnels.

La fonction de la consommation de fuel :

$$F^F_{\sigma_i\sigma_{i+1}}(v_{\sigma_i\sigma_{i+1}}) = d_{\sigma_i\sigma_{i+1}}\left(\frac{w_1}{v_{\sigma_i\sigma_{i+1}}} + w_2 + w_3 f_{\sigma_i\sigma_{i+1}} + w_4 v^2_{\sigma_i\sigma_{i+1}}\right) \tag{6}$$

w_1, w_2, w_3, w_4 sont les paramètres basés sur des propriétés du fuel, les véhicules et les caractéristiques du réseau.

La consommation de fuel est convexe, le minimum v^*_F sur cette fonction, i.e. la valeur de vitesse qui minimise les coûts de fuel est :

$$\frac{dF^F_{\sigma_i\sigma_{i+1}}}{dv_{\sigma_i\sigma_{i+1}}}(v^*_F) = 0 \Rightarrow v^*_F = \left(\frac{w_1}{2w_4}\right)^{1/3} \tag{7}$$

La fonction de la consommation de fuel et de salaire pour les conducteurs :

$$\mathrm{F}^{\mathrm{FD}}_{\sigma_i\sigma_{i+1}}(v_{\sigma_i\sigma_{i+1}}) = \omega_{\mathrm{FC}} d_{\sigma_i\sigma_{i+1}} \left(\frac{w_1}{v_{\sigma_i\sigma_{i+1}}} + w_2 + w_3 f_{\sigma_i\sigma_{i+1}} + w_4 v^2_{\sigma_i\sigma_{i+1}} \right) + \omega_{\mathrm{FD}} \frac{d_{\sigma_i\sigma_{i+1}}}{v_{\sigma_i\sigma_{i+1}}} \tag{8}$$

le minimum v^*_{FD} sur cette fonction, i.e. la valeur de vitesse qui minimise les coûts de fuel est :

$$\frac{d\mathrm{F}^{\mathrm{FD}}_{\sigma_i\sigma_{i+1}}}{dv_{\sigma_i\sigma_{i+1}}}(v^*_{\mathrm{FD}}) = 0 \Rightarrow v^*_{\mathrm{FD}} = \left(\frac{\frac{\omega_{\mathrm{FD}}}{\omega_{\mathrm{FC}}} + w_1}{2w_4} \right)^{1/3} \tag{9}$$

ω_{FC} : Les coûts de fuel et de CO2 émission par litre(£)

ω_{FD} : Salaire Driver(£/s)

v^*_{F} et v^*_{FD} sont indépendants.

Considérant à la fois la consommation de fuel et les coûts de conduite, fonction objective du PRP est :

$$\mathrm{Z}_{\mathrm{PRP}}(\mathrm{R}, v) = \sum_{\sigma \in \mathrm{R}} \left(\omega_{\mathrm{FC}} \sum_{i=1}^{|\sigma|-1} \mathrm{F}^{\mathrm{F}}_{\sigma_i\sigma_{i+1}}(v_{\sigma_i\sigma_{i+1}}) + \omega_{\mathrm{FD}} t_{\sigma_{|\sigma|}} \right) \tag{10}$$

4.2.2 ILS-SP-SOA matheuristic

L'algorithme proposé, appelé ILS-SP-SOA(i.e. *Iterated Local Search-Set Partitioning-Speed Optimization Algorithm*), combine une recherche locale itérative avec les procédures d'optimisation de la vitesse et l'optimisation de la programmation en nombres entiers sur une set-partitioning formulation. Pour être plus clair, nous pouvons diviser cet algorithme en deux étapes :

Première étape :

la construction de la solution initiale et la recherche locale visent à minimiser le coût en tenant compte des décisions de routage mais sans changer les vitesses. Ce sous-problème peut être considéré comme un VRPTW (*Vehicle Routing Problem with Time Windows*) avec la fonction objective Equation 10.

La recherche est complétée par une procédure récursive d'optimisation de vitesse, qui est appliquée sur chaque optimale solution locale de la "routage" recherche locale. Nouvelles décisions de vitesse sont incluses dans une matrice de vitesse dynamique, qui est à son tour utilisée dans les sous-problèmes VRPTW ultérieures.

Deuxième étape :

Enfin, les routes associées aux solutions optimales locaux sont stockées dans un sous-ensemble, et utilisées par une procédure d'optimisation de la programmation en nombres entiers sur une set-partitioning formulation pour générer des meilleures solutions possibles. Dans ce processus, chaque circuit peut être associée à une matrice de vitesse différente, et donc cette procédure exacte ajoute un niveau supplémentaire de l'intégration entre les circuits et la vitesse des décisions.

Aperçu de l'ILS-SP-SOA est présenté dans Algorithme 1.

Input : $n_R, n_{ILS}, n_{SP}, n_{POOL}, T_{MIP}, seed$
Output : $S_{BEST-ALL}$
Meilleure solution $S_{BEST-ALL} \leftarrow \emptyset$;
Cost de meilleure solution $f(S_{BEST-ALL}) \leftarrow \infty$;
La piscine de routes $P_{PERM} \leftarrow \emptyset$;
Compteur d'itérations $i_R \leftarrow 0$;
while $i_R < n_R$ **do**
  $i_R \leftarrow i_R + 1$;
  $S_{BEST} \leftarrow \emptyset$;
  $f(S_{BEST}) \leftarrow \infty$;
  $P_{TEMP} \leftarrow \emptyset$;
  $i_{ILS} \leftarrow 0$;
  $v \leftarrow InitializeSpeedMatrix(v_{MAX})$;
  $S \leftarrow SpeedOptimisation(LocalSearch(GenInitSol(seed)))$;
  $v \leftarrow UpdateSpeedMatrix(S)$;
  while $i_{ILS} < n_{ILS}$ **do**
    $i_{ILS} \leftarrow i_{ILS} + 1$;
    $S \leftarrow SpeedOptimisation(LocalSearch(Perturbation(S_{BEST}, seed)))$;
    $v \leftarrow UpdateSpeedMatrix(S)$;
    $P_{TEMP} \leftarrow P_{TEMP} \bigcup (routesfaisablesdeS)$;
    if $f(S) < f(S_{BEST})$ **then**
      $S_{BEST} \leftarrow S$;
      $i_{ILS} \leftarrow 0$;
    end
    if $i_{ILS} \geqslant n_{ILS}/2$ **then**
      $v \leftarrow ReinitializeSpeedMatrix(v_{MAX}, S_{BEST})$;
    end
  end
  if $(n \geqslant n_{SP} \wedge i_R = n_R - 1) \vee n > n_{SP}$ **then**
    $S_{BEST} \leftarrow MIPSolver(S_{BEST}, P_{TEMP}, PPERM, n_{ILS}, T_{MIP})$;
  end
  if $f(S_{BEST}) < f(S_{BEST-ALL})$ **then**
    $S_{BEST-ALL} \leftarrow S_{BEST}$;
  end
  $P_{PERM} \leftarrow P_{PERM} \bigcup (routesfaisablesde S_{BEST})$;
  if *numbre d'itération consécutive* $\geqslant n_{POOL}$ **then**
    $P_{TEMP} \leftarrow \emptyset$;
  end
end
return $S_{BEST-ALL}$;

Algorithme 1 : ILS-SP-SOA matheuristic

4.3 A matheuristic approach for solving a home health care problem

Il s'agit de *Home Health Care Problem (HHCP)* [1], l'objectif consiste à construire des circuits et des listes de personnel optimales pour soins de santé. La difficulté est la combinaison de problème de tournées et problème de "personnel rostering" qui sont deux problèmes d'optimisation difficiles bien connus. L'objet est de minimiser le nombre de staff.

Ils proposent une formulation de programmation linéaire en nombres entiers (ILP). Ce modèle peut être utilisé directement pour résoudre des petites instances en utilisant un solveur, ici ils ont utilisé le solveur Cplex-IBM.

Pour les instances plus grandes, ils développent une matheuristique basée sur la décomposition de la formulation ILP en deux problèmes. Le premier est un problème modélisé comme un *partitioning-like problem* et il représente la partie ordonnancement de personnel. Le second problème consiste à la partie de tournée de véhicule. Ce dernier est équivalent à un problème *Traveling Salesman Multi-dépôt avec Windows Time (MTSPTW)*.

L'algorithme fonctionne comme suit :

– D'abord nous faisons la pré-affectation de personnel à des fenêtres de temps du travail posté et nous regroupons l'ensemble des services en plusieurs clusters.

– Ensuite, nous résolvons le TSPTW pour chaque cluster en utilisant une méthode de programmation dynamique proposée par Righini [16]. Le résultat est un ensemble de circuits faisables qui va être utilisés comme colonnes du *rostering problem*. Si il y a au moins un service non couvert, nous ajoutons des nouveaux clusters pour ces services et nous résolvons le TSPTW sur les nouveaux clusters.

– Enfin, nous résolvons le *rostering problem* en utilisant l'ensemble des circuits générés. Nous définissons :

R : un sous-ensemble de circuits faisables

R_i : représente l'ensemble de circuits passant par le service $i \in S$

R_k : représente l'ensemble de circuits passant par la personne $k \in P$

y_l : égale 1 si circuit $l \in R$ est dans la solution, 0 sinon

Nous pouvons écrire la formulation suivante :

$$\min \sum_{l \in R} y_l \tag{11}$$

$$\sum_{l \in R_i} y_l = 1, \quad \forall i \in S \tag{12}$$

$$\sum_{l \in R_k} y_l \leqslant 1, \quad \forall k \in P \tag{13}$$

$$y_l \in \{0,1\}, \quad \forall l \in R \tag{14}$$

La fonction objective Equation 11 est de minimiser le nombre de circuits (i.e. le nombre de personnel). Nous résolvons cette formulation avec Cplex IBM.

Remarque : J'ai appris beaucoup d'autres exemples d'application, mais je vais pas présenter toutes en détail ici, vous pouvez les trouver dans les comptes rendus hebdomadaires.

2 Étude approfondie du problème one-to-one m-PDTSP

Dans ce chapitre, nous allons nous concentrer sur une variante du problème de tournée de véhicule appelée *Multi-Commodity One-to-One Pickup-and-Delivery Traveling Salesman Problem* (one-to-one m-PDTSP). Ce problème a été initialement introduit par Hipólito Hernández-Pérez et Juan-José Salazar-González en 2009 [10], il généralise plusieurs problèmes ramassage-et-livraison reconnus.

1 Description du problème

Un véhicule d'une capacité limitée sert un groupe de clients qui peuvent fournir ou commander certains produits. Le véhicule doit transporter ces produits de leurs origines à leurs destinations (ce sont des clients). Chaque produit a un poids connu, et il a exactement une origine et une destination.

Tous les clients doivent être visités exactement une fois par le véhicule, même si certains sont ni l'origine, ni la destination de la marchandise. Donc l'objectif est de trouver une tournée Hamiltonian de longueur minimal qui satisfait toutes les demandes sans jamais violer la contrainte de capacité du véhicule.

Les données :

0 et $n+1$: le dépôt

$i \in \{1, 2, ..., n\}$: ensemble des clients

G = (V, A) : un graphe orienté complet

V = $\{0, 1, 2, ..., n, n+1\}$: le dépôt et les clients

A = $\{(i, j), i \in V, j \in V\}$: des arcs entre les clients

c_{ij} : le coût de voyage de (i, j)

$k \in K = \{1, ..., m\}$: on a m différents produits

q_k : chaque produit a un poids q_k

$s_k \in \{0, 1, ..., n\}$: l'origine de produit k

$d_k \in \{0, 1, ..., n\}$: la destination de produit k

Nous supposons $s_k \neq d_k$ and $q_k > 0$

Q : la capacité de véhicule

2 Modélisation du problème

2.1 Définition et analyse

Nous utilisons un modèle de mixte programmation linéaire en nombre entiers (MILP) pour résoudre ce problème. D'abord, nous définissons :

P $\subset$ A : une route (composé par un certain nombre des arcs) dans le graphe

$\mathbf{P}^k$: l'ensemble de toutes les routes possibles dans G de s_k à d_k

– Pour tous les sous-ensembles S $\subset$ V, on dit :

$\delta^+(S) = \{(i,j) \in A : i \in S, j \notin S\}$: l'ensemble des arcs qui partent de S

$\delta^-(S) = \{(i,j) \in A : i \notin S, j \in S\}$: l'ensemble des arcs qui arrivent à S

– Pour chaque sommet $i \in V$, on dit :

$K_i^+ = \{k \in K : s_k = i\}$: tous les produits qui partent de i, i.e. i est ses origines

$K_i^- = \{k \in K : d_k = i\}$: tous les produits qui arrivent à i, i.e. i est la destination

$q_i^+ = \sum \left\{q_k : k \in K_i^+\right\}$

$q_i^- = \sum \left\{q_k : k \in K_i^-\right\}$

Nous devons trouver une route qui correspond toutes les conditions :

$\theta(0) = 0$: la route du véhicule commence au dépôt

$\theta(n+1) = n+1$: la route du véhicule termine au dépôt

$\theta(s_k) < \theta(d_k)$ pour chaque $k \in K$: l'origine doit être visitée avant la destination

$\sum_{k \in K : \theta(s_k) \leqslant i < \theta(d_k)} q_k \leqslant Q$ pour chaque $i \in V \backslash n+1$: la capacité du véhicule est une limite supérieure de la charge du véhicule à travers tout le parcours.

La source et la destination de chaque produit implique une contrainte de précédence entre certaines paires de sommets dans V. Parce que chaque sommet dans le graphe G doit être visité exactement une fois, il est fondamental pour l'existence d'une solution de one-to-one m-PDTSP que les contraintes de précédence induisent un sous-graphe acyclique de G.

Si chaque sommet est la source d'au plus un produit et/ou la destination d'au plus un produit et les contraintes de précédence induisent un sous-graphe acyclique de G, alors la condition

$$Q \geqslant \max\left\{q_i^+, q_i^- : i \in V\right\}$$

est une condition nécessaire et suffisante pour l'existence d'une solution de one-to-one m-PDTSP.

2.2 Le modèle

Ci-dessous, nous présentons le modèle pour résoudre le one-to-one m-PDTSP. Ce modèle fait usage de la classique 2-indice formulation de la TSP, adaptés pour calculer le plus court chemin Hamiltonien de 0 à $n+1$ dans G.

Les variables mathématiques de cette formulation :

$$x_a = \begin{cases} 1 & \text{si a est choisi} \\ 0 & \text{sinon} \end{cases} \qquad \text{Pour tous les } a \in A$$

Remarque : Si A' $\subset$ A et S',S'' $\subset$ V, nous écrivons $x(A')$ au lieu de $\sum_{a \in A'} x_a$, et $x(S',S'')$ au lieu de $\sum_{(i,j) \in A : i \in S', j \in S''} x_{(i,j)}$.

La fonction objective :

$$\sum_{a \in \mathrm{A}} c_a x_a \tag{1}$$

soumis aux :

$$x(\delta^-(\{i\})) = 1 \quad \text{pour tout } i \in \mathrm{V} \setminus \{0\} \tag{2}$$

$$x(\delta^+(\{i\})) = 1 \quad \text{pour tout } i \in \mathrm{V} \setminus \{n+1\} \tag{3}$$

$$x(\delta^+(\mathrm{S})) \geqslant 1 \quad \text{pour tout } \mathrm{S} \subset \mathrm{V} : 0 \in \mathrm{S}, n+1 \notin \mathrm{S} \tag{4}$$

$$x_a \in \{0,1\} \quad \text{pour tout } a \in \mathrm{A} \tag{5}$$

La fonction objective Equation 1 consiste à minimiser le coût total des arcs sur la route. Equation 2 et Equation 3 assurent que le véhicule visite chaque sommet exactement une fois. Equation 4 assure que le chemin connecte tous les sommets en une seule route.

Nous pouvons voir que Equation 2 – Equation 5 assurent que nous obtenons un chemin Hamiltonien, mais quelques chemins Hamiltonian peuvent être infeasibles pour le one-to-one m-PDTSP, comme nous avons aussi les contraintes de précédence et la contrainte de capacité du véhicule.

Donc nous allons ajouter certains nouvelles contraintes pour créer un modèle modifié pour one-to-one m-PDTSP, ce modèle peut éviter les tournées infaisables. Pour faire ça, nous avons deux choix :

Formulation de chemin

Nous associons le chemin Hamiltonian avec un chemin $\mathrm{P} \in \mathbf{P}^k$ pour tout $k \in \mathrm{K}$. Les poids au long de tous ces chemins doivent être inférieurs ou éguals à la limitation de la capacité du véhicule. On va formuler ces conditions d'une manière linéaire.

$$\lambda_{\mathrm{P}} = \begin{cases} 1 & \text{si la véhicule va de } s_k \text{ à } d_k \text{ par chemin P} \\ 0 & \text{sinon} \end{cases} \qquad \text{Pour tout } k \in \mathrm{K} \text{ et tout } \mathrm{P} \in \mathbf{P}^k.$$

Et la formulation est Equation 1-Equation 5 avec :

$$\sum_{k \in \mathrm{K}} q_k \sum_{a \in \mathrm{P} \in \mathbf{P}^k} \lambda_{\mathrm{P}} \leqslant \mathrm{Q} x_a \quad \text{pour tout } a \in \mathrm{A} \tag{6}$$

$$\sum_{\mathrm{P} \in \mathbf{P}^k} \lambda_{\mathrm{P}} = 1 \quad \text{pour tout } k \in \mathrm{K} \tag{7}$$

$$\lambda_{\mathrm{P}} \geqslant 0 \quad \text{pour tout } \mathrm{P} \in \mathbf{P}^k \text{ et tout } k \in \mathrm{K} \tag{8}$$

Evidemment, la condition Equation 5 en variable x_a fait la condition Equation 8 inutile.

Formulation de flux

Un modèle alternatif pour one-to-one m-PDTSP est basé sur le flux de produit.

Pour chaque arc $a \in A$ et chaque produit $k \in K$, nous définissons la variable f_a^k qui représente le poids de produit k dans la véhicule quand la véhicule passe par l'arc k. Donc nous avons :

$$\sum_{a \in \delta^+(\{i\})} f_a^k - \sum_{a \in \delta^-(\{i\})} f_a^k = \begin{cases} q_k & \text{si } i = s_k \\ -q_k & \text{si } i = d_k \\ 0 & \text{autrement} \end{cases} \tag{9}$$

$$f_a^k \geqslant 0 \quad \text{pour tout } a \in A \text{ et tout } k \in K \tag{10}$$

$$\sum_{k \in K} f_a^k \leqslant Q x_a \quad \text{pour tout } a \in A \tag{11}$$

Contraintes Equation 9 et Equation 10 imposent que, pour chaque produit k, il devrait y avoir un chemin de s_k à d_k qui envoye q_k unités de k. L'inégalité Equation 11 assure la limitation de la capacité du véhicule à chaque arc.

Il est bien connu que les bornes inférieures et supérieures serrés sur les variables de flux ont un impact crucial sur la valeur optimale de l'ensemble du modèle. Par exemple, c'est mieux de remplacer Equation 10 par Equation 12

$$\sum f_a^k - \sum_{a \in \delta^-(\{i\})} f_a^k = \begin{cases} q_k x_a & \text{si } a \in \delta^+(\{s_k\}) \text{ ou } a \in \delta^-(\{d_k\}) \\ 0 & \text{si } a \in \delta^+(\{0\}) \text{ et } s_k \neq 0 \\ 0 & \text{si } a \in \delta^-(\{n+1\}) \text{ et } d_k \neq n+1 \\ 0 & \text{si } a \in \delta^+(\{d_k\}) \\ 0 & \text{si } a \in \delta^-(\{s_k\}) \end{cases} \tag{12}$$

Nous pouvons également renforcer la contrainte Equation 11 par la suivante :

$$\sum_{k \in K} f_{(i,j)}^k \leqslant x_{(i,j)} (Q - \max\left\{0, q_i^- - q_i^+, q_j^+ - q_j^-\right\}) \quad \text{pour tout } (i,j) \in A \tag{13}$$

Notez que $q_j^+ - q_j^-$ est le poids net collecté par le véhicule du client j, et $q_i^- - q_i^+$ est le poids net obtenu du véhicule par le client i.

Ces deux modèles ont deux perspectives différentes, mais la méthode pour les résoudre sont assez similaires, donc nous allons utiliser le deuxième modèle dans les algorithmes suivants.

3 Algorithmes existants pour one-to-one m-PDTSP

3.1 Single Benders Décomposition

Hipólito Hernández-Pérez et Juan-José Salazar-González qui ont soulevé ce problème ont proposé un algorithme exact appelé *Single Benders Decomposition* [10] implémenté dans un branch-and-cut cadre,.

Comme nous pouvons voir que le modèle Equation 1 – Equation 5, Equation 9 – Equation 11 a un nombre polynôme de variables continues, nous peuvons développer une technique de décomposition où un problème de maître (*master problem*) fonctionne uniquement avec les variables x_a.

La décomposition : Nous résolvons d'abord un problème simplifié appelé *relaxed master problem* qui considére plutôt les contraintes dans Equation 1 – Equation 5. Ensuite, nous vérifions la faisabilité par un *subproblem*, dans ce problème, nous allons déterminer une contrainte qui n'a pas été respectée dans la solution précédente, nous l'ajoutons au master problème et nous recommencons la entière procédure. La procédure est répétée jusqu'à le *relaxed master problem* est infaisable, ou zéro est la solution optimale du *subproblem*.

Le branch and cut : Dans ce dernier cas où la solution optimale est zéro, nous remplons la variable x_a avec une variable $0 < x_a < 1$, et répétons les procédures de cut-generation dans chaque nœud de l'arbre de décision de branche dans un cadre branch-and-cut.

Cet algorithme peut juste résoudre les instances jusqu'à 24 clients et 15 produits.

En 2011, I. Rodríguez-Martín et Juan-José Salazar-González [17] ont proposé une matheuristique appelée *MIP-based GRASP* pour le one-to-one m-PDTSP qui peuvent résoudre des instances avec un maximum de 300 clients et 600 produits. Et ils ont comparé les résultats des deux algorithmes sur les instances pas très grandes.

3.2 MIP-based GRASP

MIP-based greedy randomized adaptive search procedure (GRASP), cette méthode combine les techniques de programmation mathématique avec des ingrédients typiques de métaheuristiques et il peut donc être considérée comme une matheuristique

MIP(Mixed integer programming) est la programmation linéaire où certaines variables sont contraintes d'être entiers.

GRASP est un multi-démarrage algorithme métaheuristique, où chaque itération se compose de deux phases :

Phase 1 – la construction d'une solution faisable. Pour générer une solution initiale, nous appliquons une heuristique gloutonne randomisés comme l'heuristique "les plus proches voisins" pour le TSP. C'est un chemin simple où c'est pas obligatoire de visiter l'origine avant de la destination d'un produit. Si aucunne solution faisable est trouvée, la procédure recommence à partir du dépôt.

Phase 2 – l'amélioration. La phase d'amélioration commence à partir d'une solution faisable donnée et essaie de l'améliorer en utilisant une modification de l'algorithme branch-and-cut.

La modification consiste à fixer les variables pour obtenir un modèle plus facile à résoudre. Plus précisément, un pourcentage donné de variables binaires, correspondant à des arcs de la solution initiale, sont fixés à 1. Ensuite, le modèle MIP détendue est résolu en utilisant l'approche branch-and-cut avec une limite de temps.

Le processus qui combine de la fixation de variables avec la résolution du modèle MIP correspondant est répété jusqu'à ce que il n'y a aucune amélioration ou la limite de temps est atteinte.

Les deux phases sont répétées jusqu'à ce qu'un critère d'arrêt soit satisfait. Dans notre cas, le critère d'arrêt est une limite de temps.

En outre, chaque solution trouvée, soit par l'algorithme de construction ou par l'algorithme branch-and-cut, est soumis à des "edge-exchange" procédures comme 2-opt et 3-opt pour le standard TSP. Ces procédures d'échange ont été modifiées afin de maintenir la faisabilité des solutions.

```
repeat
    s_0 ← GenerateInitialSol();
    s_0 ← 2-and-3-opt(s_0);
    s_best ← s_0;
    s_local_best ← s_0;
    while Il y a une solution améliorée et LS_timeLimit pas encore arrivé do
        Fix_variables(s_local_best);
        s' ← solveMIP();
        s' ← 2-and-3-opt(s');
        if s' est mieux que s_local_best then
            s_local_best ← s';
        end
    end
    if s_local_best est mieux que s_best then
        s_best ← s_local_best;
    end
until timeLimit arrivé;
return s_best;
```

Algorithme 2 : MIP-based GRASP for the m-PDTSP

Deuxième partie

Projet de Développement

3 Nouvelles matheuristiques

Après toutes les études du premier semestre, nous voulons créer nos propres matheuristiques en mettant en oeuvre les méthodes et surtout les principes que nous avons appris. Dans ce chapitre, nous présentons les algorithmes que nous avons proposés pour résoudre le problème one-to-one m-PDTSP présenté dans Chapitre 2.

1 Aperçu de l'algorithme

En utilisant le modèle matheuristique présent dans Section 2 (Chapitre 2), nous pouvons résoudre les petites instances à optimalité directement par un solveur, par exemple Cplex, mais malheureusement,le solveur ne peut pas résoudre des plus grandes instances qui sont avec plus que 1000 constraintes.

Pour explorer un algorithme efficace pour les grandes instances, nous avons fait beaucoup d'essayes. Mais l'idée principale de nos algorithmes est de trouver une solution initiale utilisant une heuristique et puis utiliser le modèle MLP et un solveur pour l'améliorer. Les améliorations sont faites par déduire le problème en plus petit problème et le résoudre à l'optimalité ou à proximité de l'optimalité. Ça veut dire que le modèle mathématique ici sert à l'optimisation locale, donc nos matheuristiques appartiennent à la deuxième grande familles de matheuristique "Heuristique d'amélioration" présentée dans Section 3 (Chapitre 1).

Globalement, nos matheuristiques sont en trois phrases :

- **Phase 1** Génération d'une solution initiale
- **Phase 2** Déduction du problème original
- **Phase 3** Résolution du modèle avec un solveur

Nous répétons la **Phase 2** et la **Phase 3** jusqu'à certains critères sont attendus.

Comme nous avons essayé de différentes façons pour réaliser ces trois phrases, nous avons beaucoup de versions de l'algorithme, certains marchent bien mais certains sont moins bien. Nous avons gardé les deux meilleures versions pour la comparaison des résultats.

Dans la section suivante, nous allons vous présenter en détail les déroulements des trois phases, y compris les deux différentes vesions de matheuristique que nous allons faire comparer avec les résultats de Hipólito Hernández-Pérez et Juan-José Salazar-González dans leur article [10].

2 Déroulement de l'algorithme

Pour faciliter la compréhension de l'algorithme, nous nous faisons rappeler ici les significations des symboles dans le problème :

0 et $n+1$: le dépôt

$i \in \{1, 2, ..., n\}$: ensemble des clients

G = (V, A) : un graphe orienté complet

V = $\{0, 1, 2, ..., n, n+1\}$: le dépôt et les clients

A = $\{(i, j), i \in V, j \in V\}$: des arcs entre les clients

$k \in K = \{1, ..., m\}$: on a m différents produits

$s_k \in \{0, 1, ..., n\}$: l'origine de produit k

$d_k \in \{0, 1, ..., n\}$: la destination de produit k

2.1 Génération d'une solution initiale

Nous avons utilisé trois méthodes pour générer une solution initiale :

1. **Génération aléatoire** Nous créons un circuit qui part du dépôt et se termine au dépôt, et tous les clients à visiter, nous les mettons au hasard dans ce circuit.
2. **Le plus proche voisin classique** Pour trouver un circuit concernant le dépôt et tous les clients, nous partons du dépôt, et chaque fois nous choisissons le client qui est le plus proche de la position actuelle pour visiter. Après nous avons visité tous les clients, nous revenons au dépôt.
3. **Le plus proche voisin considérant les contraintes de précédence** En respectant les contraintes de précédence, c'est-à-dire pour tous les produits, son origine doit être visitée avant sa destinations, nous choisissons le client qui est le plus proche de la position actuelle pour visiter. Identiquement, nous départons et arrivons au dépôt.

Par conséquent, la troisième méthode est la meilleure, non seulement d'améliorer la qualité des résultats, mais aussi gagner de temps pour le calcul ultérieur.

Le pseudo code de la troisième méthode :

```
s[0] ← 0 ;
O ← les k origines des produits;
D ← les k destinations des produits;
A_dispo ← A − D;
for each i ∈ [1, n] do
    (s[i − 1], j) ← le plus court chemin entre s[i − 1] et les sommets dans A_dispo;
    s[i] ← j;
    A_dispo ← A_dispo − j;
    while s[i] est l'origine d'un produit k do
        Supprimer la destination d_k du produit k dans D;
        if il n'y a aucun de d_k dans D then
            Ajouter d_k dans A_dispo;
        end
    end
end
return s;
```

Algorithme 3 : Génération d'une solution initiale

De cette façon, nous obtenons une solution qui respecte les contrainte de précédence mais qui ne respecte pas forcément la contrainte de la capacité du véhicule, du coup, c'est une solution qui ne garantit pas la faisabilité.

2.2 Déduction du problème original

Pour déduire le problème original, nous comptons fixer des arcs à partir de la solution initiale. Après fixer des arcs, nous allons calculer les données pour obtenir une nouvelle instance, mais plus petite que l'original et l'adaptons à notre modèle mathématique. Puis nous allons lancer le solveur et résoudre ce modèle.

A partir de sa solution qui est forcément faisable à local, nous pouvons obtenir une nouvelle solution pour le problème original en intégrant la solution initiale. Comme la solution initiale respecte toutes les contraintes sauf la contrainte de capacité du véhicule, nous devons revérifier cette contrainte sur la nouvelle solution pour assurer sa faisabilité. Alors, nous ajoutons une étape de vérification après l'intégration de la solution locale et la solution initiale, et nous gardons juste des solutions bien faisables. De cette façon, nous avons la possibilité d'avoir des améliorations locales et d'avoir des bonnes solutions.

Nous avons proposé deux méthodes pour fixer des arcs et c'est selon ces méthodes nous distinguons nos deux différentes matheuristiques. Pour chaque méthode, nous appliquons différentes stratégies pour sélectionner les arcs à fixer et différentes façons correspondante pour recalculer les données dans la nouvelle instance, bien sûr que les critères de l'arrêt de l'algorithme sont différents. Nous allons vous présenter séparément comment marchent ces deux méthodes de réduction.

2.2.1 La première matheuristique

Sélection des arcs

Dans cette méthode, nous prenons m sommets consécutifs ($m < n + 2$) dans la solution initiale pour créer une nouvelle instance et fixer les autres sommets. Comme vous pouvez voir dans la Figure 1, premièrement, nous prenons la nouvelle instance 1 et puis la nouvelle instance 2 jusqu'à la nouvelle instance $n - m + 2$, nous appelons ça un "tour". Nous allons faire plusieurs tours en fait.

Calcul des données

1. Recalculer la matrice de distances :

Dans cet algorithme, comme nous fixons les sommets consécutifs et nous créons pas des nouveaux sommets, du coup, nous changons pas les informations sur un sommet, des distances entre les sommets sélectionnés ne changent pas par rapport au graphe original.

2. Recalculer la matrice de demandes

Dans le graphe original, le premier sommet $0 \in A$ et le dernier sommet $n + 1 \in A$ sont le dépôt, et ils n'ont pas d'offres ou demandes des produits. Mais dans le modèle modifié, comme nons considérons juste un certain nombre de sommets consécutifs A' dans la solution initiales, dont sommets $h \in A'$ et $q \in A'$ sont le premier et le dernier, nous perdrons des informations sur les demandes des produits.

Alors, pour chaque produit, nous allons prendre la somme des offres de tous les sommets dans $A - A'$ qui sont fixés ainsi que l'offre de h comme l'offre de ce produit du sommet h, et la somme des demandes de tous les sommets dans $A - A'$ ainsi que la demande de q comme la demande de ce produit du dernier sommet q dans A'. Les demandes ou offres des produits sur les autres sommets séléctionnés $A' \setminus \{q, h\}$ ne changent pas.

Critères de l'arrêt

Chaque fois nous obtenons une nouvelle solution à partir d'une résoulution locale, nous vérifions sa faisabilité et décidons si nous la gardons ou non. Après que nous avons fait un "tour" complet sur la solution initiale, les ordres de certains sommets seront changés, si nous commençons une autre fois à partir du dépôt, peut-être que nous pouvons avoir des nouvelles améliorations. Alors, quand nous devons

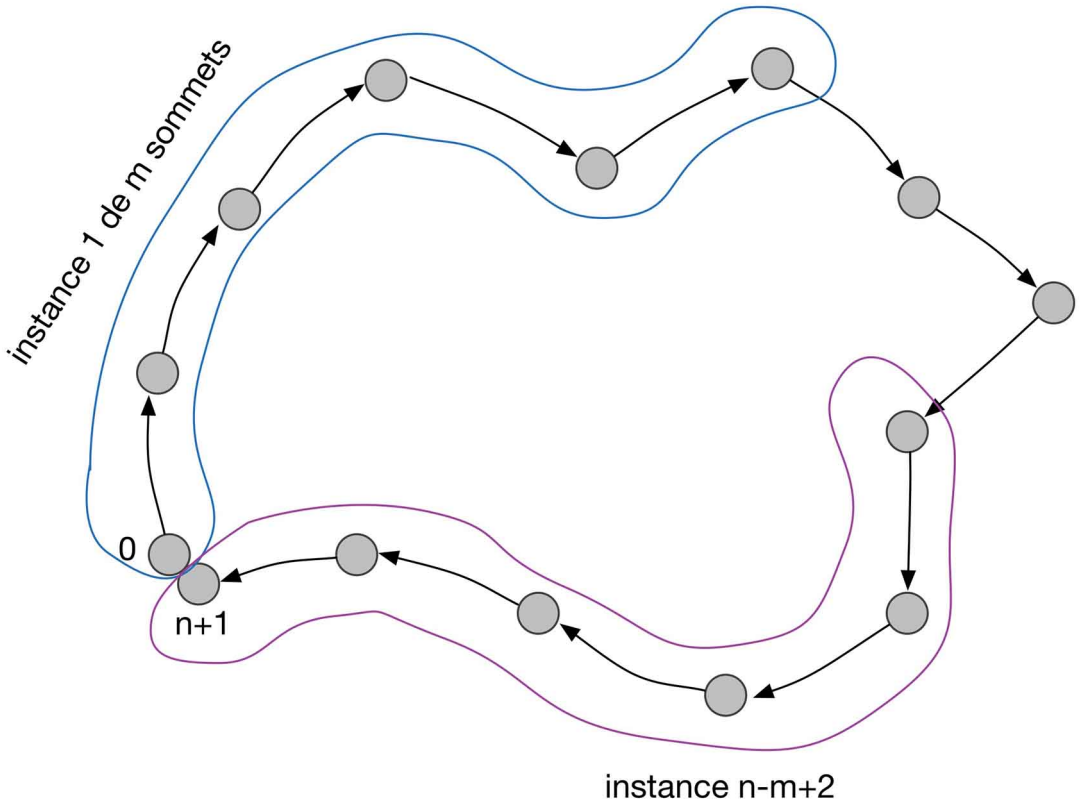


Figure 1 – *Première méthode de sélection des arcs*

arrêter l'algorithme ? Comme nous utilisons les mêmes façons de sélection des sommets pour créer des nouvelles instances, si il n'y aucune amélioration dans un "tour" ou il n'y aucune solution faisable, il n'y aura pas de amélioration dans les "tour" après ni de solution faisable, du coup, nous arrêtons de rechercher dans cette situation.

Le pseudo code en global de cet algorithme :

```
s_0 ← Génération de solution initiale;
s_best ← une très mauvaise solution;
isImprove ← true;
while isImprove do
    isImprove ← false ;
    A' ← Sélectionner un certain nombre de sommet consécutifs dans s_best;
    Supprimer les arcs associant les sommets dans A', fixer les autres arcs de s_best ;
    Recalculer le matrice de distances et la matrice de demandes sur A' ;
    Adapter le modèle sur A' ;
    s_local ←Résoudre le modèle avec Cplex ;
    Obtenir s' depuis s_local et s_0;
    if s' est mieux que s_best AND s' est faisable then
        isImprove ← true ;
        s_best ← s';
    end
end
return s_best;
```

Algorithme 4 : Première matheuristique pour le problème m-PDTSP

2.2.2 La deuxième matheuristique

Sélection des arcs

Dans la deuxième méthode, au lieu d'utiliser la façon déterminée pour sélecionner les sommets à fixer, nous allons les sélectionner par hasard. Ça veut dire que dans une solution initiale, nous allons choisir certaines paires des sommets adjacents, et fixer l'arc entre eux. Du coup, c'est plutôt de fixer les arcs dans la solution initiale. Ça peut être deux arcs adjacents comme $x2$ dans Figure 2, ou bien trois ou plus arcs adjacents dans les plus grandes instances. Nous pouvons imaginons comme les sommets adjacents que nous avons fixé devient un seul sommet, nous diminuons la dimension du problème de cette façon.

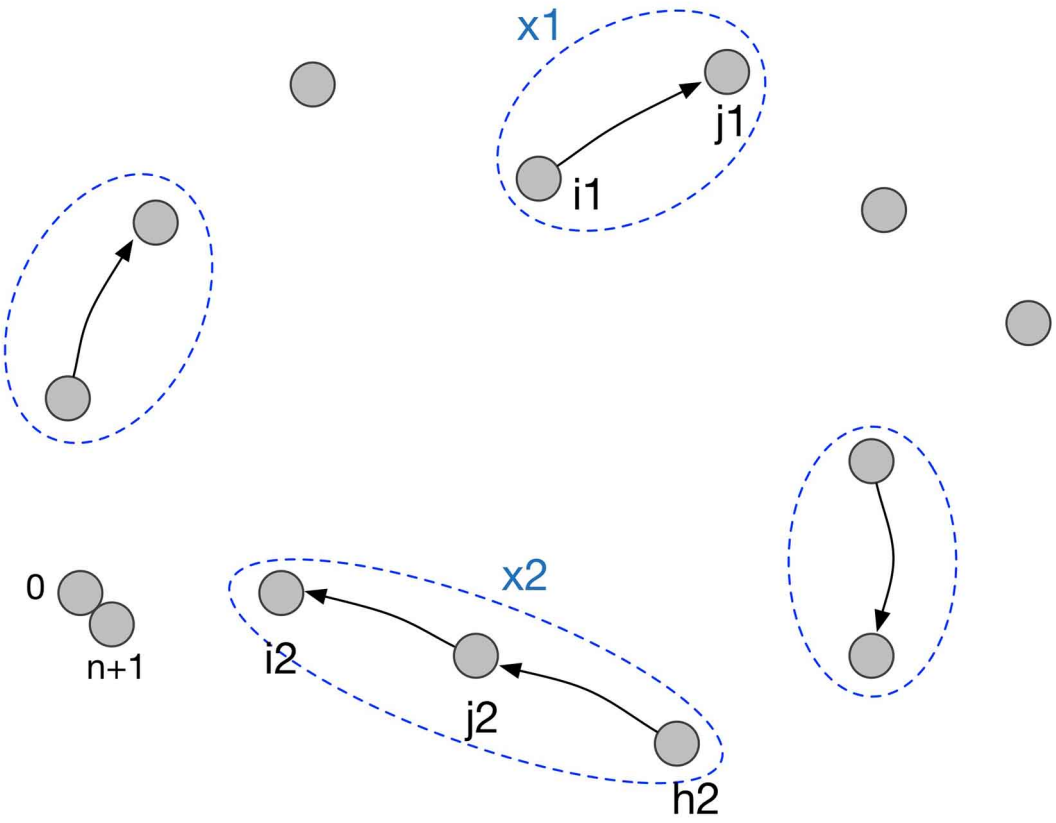


Figure 2 – *Deuxième méthode de sélection des arcs*

Calcul des données

1. Recalculer la matrice de distances :

Voyons dans la Figure 2, supposons que nous avons un nouveau sommet $x1$ qui était un arc $(i1, j1)$ entre sommets $i1$ et $j1$ dans la solution initiale. Quand nous voulons calculer la distance d'un autre sommet n à sommet $x1$, ça sera la distance entre sommet n et $i1$ dans le graphe original, et quand nous voulons calculer la distance de $x1$ à un autre sommet m, ça sera la distance entre $j1$ et m dans le graphe original.

Pour un petit groupe de sommets, par exemple $x2$, de même façon, la distance entre n et $x2$ sera la distance de n à $h2$ et la distance entre $x2$ et m sera égale à la distance $(i2, m)$.

Nous devons recalculer les distances entre tous ces nouveaux sommet pour créer la nouvelle matrice de instances. Evidemment, nous avons plus de calculs à faire par rapport à la première façon de fixation.

2. Recalculer la matrice de demandes

Les calculs de matrice de demandes sont plus simples. La demande ou l'offre sur certain produit de sommet x seront la somme des demandes ou l'offres des tous les sommets qui sont contenus dans x sur ce produit.

Par exemple, si le sommet i est la source de produit $k1$ et le sommet j est la source de produit $k2$, donc le nouveau sommet x seront les sources de produit $k1$ et $k2$, et si i est la source de produit k et j est la destination de k, alors la demande et l'offre du nouveau sommet x sur produit k sont mise à zéro.

Critères de l'arrêt

Dans cette méthode, il n'y a pas de situation où nous pouvons être sur qu'il n'y aura plus d'amélioration, alors comment nous définissons la critère d'arrêt ? Nous allons compter la fois de lancements du solveur où nous obtenons des solutions faisables et nous n'obtenons pas d'amélioration, si nous arrivons à n fois consécutives sans amélioration, alors nous allons arrêter de rechercher. n est un paramètre très important pour la performance de cette méthode, si il est trop petit, nous pouvons manquer une mieux solution, mais il est trop petit, ça sera un perdu du temps.

Selon nos expérimentations, pour les instances ayant 10 sommets, nous le fixons à 50, pour les instances de 15 sommets, nous le fixons à 100, il égale à 300 pour les instances avec 20 sommets et 10 produits, et il égale à 400 pour les plus grandes instances.

Le pseudo code en global de cet algorithme :

```
s_0 ← Génération de solution initiale;
s_best ← une très mauvaise solution;
failConsecu ← 0;
while failConsecu < 50 do
    A' ← Sélectionner un certain nombre de paire de sommet dans s_best;
    Recalculer le matrice de distances et la matrice de demandes sur A' ;
    Adapter le modèle sur A' ;
    s_local ←Résoudre le modèle avec Cplex ;
    Obtenir s' depuis s_local et s_best ;
    if s' est faisable then
        if s' est mieux que s_best then
            s_best ← s';
            failConsecu ← 0;
        else
            failConsecu ← failConsecu +1;
        end
    end
end
return s_best;
```

Algorithme 5 : Deuxième matheuristique pour le problème m-PDTSP

2.3 Résolution avec Cplex

2.3.1 Le modèle

Dans cette partie, nous allons présenter le modèle mathématique dans un premier temps. Nous avons utilisé le modèle de flux proposé par Hipólito Hernández-Pérez et Juan-José Salazar-Gonzélez, et la présentation de ce modèle se trouve dans subsection 2.2 (Chapitre 2). Avec la fonction objective Equation 1 (Chapitre 2), nous ajoutons des contraintes Equation 2 (Chapitre 2) – Equation 5 (Chapitre 2) et Equation 9 (Chapitre 2) – Equation 11 (Chapitre 2).

Au cours de notre recherche, mon encadrant a trouvé un erreur du modèle de Hipólito Hernández-Pérez et Juan-José Salazar-Gonzélez, cet erreur se trouve dans la contrainte Equation 4 (Chapitre 2) qui est écrit comme :

$$x(\delta^+(S)) \geqslant 1 \quad \text{pour tout } S \subset V : 0 \in S, n+1 \notin S \tag{1}$$

Cette contrainte est destinée à forcer l'ensemble des arcs choisis à former un chemin hamiltonien. Cette contrainte évite de constituer un chemin de 0 à $n+1$ avec certains sommets mais pas tous, les autres sommets étant dans des circuits.

Mais avec ses conditions : pour tout $S \subset V : 0 \in S, n+1 \notin S$, cette contrainte ne marche pas comme il faut, nous prenons une instance de 6 sommets comme un exemple, le sommet 0 et 5 sont le dépôt, supposons que la matrice X des résultats sont comme suivant, $X[0][1] = 1$ implique la sélection de l'arc $(0,1)$.

	1	2	3	4	5
0	1	0	0	0	0
1	0	0	0	0	1
2	0	0	1	0	0
3	0	0	0	1	0
4	0	1	0	0	0

Nous pouvons voir que matrice X respecte tous les autres contraintes, et la Equation 1 est également vérifiée : tout sous-ensemble contenant 0 et pas $n+1$ a au moins un arc sortant. Mais en fait, avec cette matrice de résultats, les arcs correspondants forment le chemin $(0,1,5)$ et le circuit $(2,3,4)$, c'est pas du tout un chemin hamiltonien.

Il faut donc modifier la Equation 1 en la nouvelle :

$$x(\delta^+(S)) \geqslant 1 \quad \text{pour tout } S \subset V : 0 \notin S, n+1 \notin S \tag{2}$$

2.3.2 Mise en oeuvre

Pour mettre en oeuvre ce modèle avec Cplex, nous avons choisi la langue Java en ses deux packages *ilog.concert* et *ilog.cplex*.

Pour gagner du temps, nous ne créons le modèle qu'une seule fois, chaque fois quand nous voulons l'adapter sur une nouvelle instance, nous utilisons les données calculées (la matrice de distances et la matrice de demandes) pour modifier les coeffients des contraintes et de la fonction objective dans le modèle, et puis lance Cplex pour le résoudre.

Comme nous savons, le modèle est difficile à résoudre, principalement à cause de la Equation 2, parce que il a une croissance exponentielle, du coup, nous devons faire de notre mieux pour réduire le nombre de variables x_a. Premièrement, nous créons $(n+1)\times(n+1)$ variables x_a au lieu de $(n+2)\times(n+2)$, parce qu'il n'y pas de arc qui part du sommet $n+1$, et il n'y a pas de arc qui arrive au sommet 0. Nous pouvons

enlever $n+3$ variables de cette façon.

Après la création du modèle, il y a aussi certains fixations des variables à faire pour diminuer le temps de calcul de Cplex.

1. Fixer les variables représentant les arcs (i,j) où $i=j$ à 0, parce que nous pouvons pas partir d'un sommet et arriver au même sommet.

2. Chaque produit induit une contrainte de précédence : $s_k \prec d_k$. On peut donc fixer à 0 toutes les variables $x_{i,j}$ telles que $i=d_k$ et $j=s_k$.

3. Si le sommet i est la destination du produit k, alors $f_a^k=0$ pour tous les $a \in \delta^+(\{i\})$, ça veut dire qu'il n'y pas de produit k part de i. Au contraire, si i est la source du produit k, alors il n'y aura pas de produit k qui arrive à i, $f_a^k=0$ pour tous les $a \in \delta^-(\{i\})$.

4. Nous pouvons aussi fixer $x_{i,j}$ à 0 dans les 3 cas suivants :
   — $\sum_{s_k=i \text{ or } d_k=j} q_k > \mathrm{Q}$
   — $\sum_{d_k=i \text{ or } (d_k=j \text{ and } s_k\neq i)} q_k > \mathrm{Q}$
   — $\sum_{s_k=j \text{ or } (s_k=i \text{ and } d_k\neq j)} q_k > \mathrm{Q}$

5. Si un des 4 cas suivants est vrai, alors les 2 variables $x_{i,j}$ et $x_{j,l}$ ne peuvent pas être égales simultanément à 1 et on peut poser la contrainte supplémentaire : $x_{i,j}+x_{j,l} \leq 1$.
   — $\sum_{s_k=i \text{ or } d_k=j \text{ or } (d_k=l \text{ and } s_k\neq j)} q_k > \mathrm{Q}$
   — $\sum_{s_k=j \text{ or } d_k=l \text{ or } (s_k=i \text{ and } d_k\neq j)} q_k > \mathrm{Q}$
   — $\sum_{d_k=i \text{ or } (d_k=j \text{ and } s_k\neq i) \text{ or } (d_k=l \text{ and } s_k\neq i) \text{ and } s_k\neq j)} q_k > \mathrm{Q}$
   — $\sum_{s_k=l \text{ or } (s_k=j \text{ and } d_k\neq l) \text{ or } (s_k=i \text{ and } d_k\neq j \text{ and } d_k\neq l)} q_k > \mathrm{Q}$

4 Expérimentations

1 Les instances

Pour les expérimentations, nous utilisons les instances de deux familles qui sont présentées par Hipólito Hernández-Pérez et Juan José Salazar-González dans leurs articles [10] en 2009. Comme il n'y a pas de instances de référence dans la littérature pour la m-PDTSP, ils ont généré trois familles de instances et ce que nous utilisons sont les deux dernières familles : Classe 2 et Classe 3.

Pour créer les instances dans la Classe 2, ils ont généré n points aléatoirement dans le carré $[-500, 500] \times [-500, 500]$, le dépôt est situé au point $(0, 0)$. Le coût de voyage c_{ij} entre les points i et j a été calculé comme la distance euclidienne entre les deux points de localisation, donc la matrice de distance de chaque instance de la Classe 2 est symétrique. Pour les produits, ils choisissent au hasard des arcs et définissent un objet k pour chaque arc choisi sans créer un cycle avec ceux précédemment choisis. Le nombre de produits m a été pris en compte dans 5, 10, 15. Pour chaque valeur de m, on prend également différentes capacités Q.

Les instances de la Classe 3 ont été créées comme les instances dans la Classe 2, mais dans la Classe 3 chaque point est l'origine ou la destination d'un objet. Plus précisément, le nombre de produits de l'instance est $n/2$ et pour chaque $k = 1, ..., n/2$, $s_k = 2k - 1$ et $d_k = 2k$. Les quantités q_k sont générés aléatoirement entiers dans $[1, 5]$. Différentes capacités Q ont été envisagées.

Pour conduire des expérimentations, nous avons écrit aux auteurs Hipólito Hernández-Pérez et Juan José Salazar-González, ils nous ont répondu gentiment et nous ont envoyé toutes les instances générées, donc nous utilisons exactement les mêmes instances comme ce qu'ils ont utilisées pour l'article [10], dans ce cas, la comparaison des résultats sera plus efficace et fiable.

2 Les résultats

Les résultats des expérimentaion des intances dans la Classe 2 sont listés dans la Table 1 et les résultats des expérimentation des instances dans la Classe 3 sont listés dans la Table 2. Voici les significations dans les tableaux :

n : le nombre de clients + un dépôt

m : le nombre de produit

Q : la capacité du véhicule

Algo1 : la première matheuristique proposée Algorithme 4 (Chapitre 3)

Algo2 : la deuxième matheuristique proposée Algorithme 5 (Chapitre 3)

SBD : Single Benders Décomposition proposé par Hernández-Pérez et Salazar-González

Table 1 – *Résultats des instances dans Classe 2*

n	m	Q	Algo1	Algo2	SBD	n	m	Q	Algo1	Algo2	SBD
9	5	10	3190.68	3190.68	3192	14	10	10	6229.04	5045.00	5044
9	5	15	3530.74	3530.74	3531	14	10	15	5550.82	4405.71	4406
9	5	20	3190.68	3190.68	3192	14	10	20	4896.88	4643.38	4643
9	5	500	3530.74	3530.74	3531	14	10	25	4896.88	4106.08	4106
9	10	10	3978.68	3978.68	3978	14	10	30	6482.39	5470.47	5472
9	10	15	4420.20	4340.44	4343	14	10	500	5737.39	4985.51	4986
9	10	20	4665.89	4665.89	4580	14	15	15	7169.60	5880.04	5881
9	10	500	4420.20	4340.44	4343	14	15	20	7075.40	5780.44	5781
9	15	15	4214.94	4214.94	4216	14	15	30	5980.96	5652.18	5653
9	15	20	5813.49	5813.49	5814	14	15	500	4992.32	4985.51	4986
9	15	25	3978.68	3978.68	3978	19	10	10	–	5936.62	5937
9	15	30	4932.81	4648.24	4648	19	10	15	–	5716.06	5174
9	15	500	4484.53	3796.00	3797	19	10	20	5617.49	4843.94	4836
19	15	15	–	6962.84	7120	19	10	25	–	4229.84	4228
19	15	20	–	6223.43	7543	19	10	30	5617.49	4837.37	4836
19	15	25	6701.63	6047.55	6048	19	10	500	4957.10	4229.84	4228
19	15	30	6319.19	4765.50	4722						
19	15	500	7809.68	6489.14	6487						

Comme nous pouvons voir dans les tableaux, la deuxième matheuristique que nous proposons dans subsubsection 2.2.2 (Chapitre 3) est bien efficace, il peut toujours trouver la meilleure solution comme Hernández-Pérez et Salazar-González ont trouvée, et pour certains instances très grandes, il même améliore leur résultat, mais en plus de temps bien sûr.

Mais la première matheuristique que nous avons proposons dans subsubsection 2.2.1 (Chapitre 3) ne marche pas bien pour les grandes instances. Pour moi, c'est parce que nous utilisons une façon déterminée pour générer la solution initiale, ainsi que une façon déterminée pour attirer les nouvelle petites instances de la solution initiale, donc l'espace de rechercher est trop petit. Si la solution initiale est loin de la meilleure solution, alors on est difficile à arriver. De toute façon, elle est efficace pour les instances de moyenne taille.

Table 2 – *Résultats des instances dans Classe 3*

n	m	Q	Algo1	Algo2	SBD	n	m	Q	Algo1	Algo2	SBD
10	5	10	3986.96	3773.93	3773	20	10	10	–	4955.01	4955
10	5	15	3295.00	3295.00	3295	20	10	15	4675.99	4419.54	4420
10	5	20	4006.48	4006.48	4007	20	10	20	5624.35	4807.33	4808
10	5	25	3312.58	3292.36	3293	20	10	25	6187.41	4848.18	4848
10	5	30	3312.58	3292.36	3293	20	10	30	5283.64	4741.66	4741
10	5	500	3986.96	3773.93	3773	20	10	500	5283.64	4741.66	4741

5 Gestion de projet

Ce Projet Recherche & Développement se compose de deux parties principales, la première partie se concentre sur l'étude de la littérature, la deuxième partie contient les propositions et les expérimentations.

Dans la section suivante, je vais lister toutes les tâches réalisées séparément pendant le premier semestre et pendant le deuxième semestre ainsi qu'une planifications des tâches pour chaque semestre.

1 La planification du projet

Afin de mener à bien ce projet, j'ai découpé le projet entier en tâches au début, et j'ai aussi estimé le temps de réalisation afin de planifier les tâches. Cette planification a presque été bien respectée pendant le permier semestre.

Voici une représentation de la liste des tâches que j'ai réalisées et un diagramme de Gantt associé à cette liste.

Titre	Début	Fin
1) Recherche et bibliographie	16/09/2015	04/01/2016
1.1) Etude du contexte du projet	16/09/2015	17/09/2015
1.2) Etude de l'état de l'art	17/09/2015	04/12/2015
1.3) Choisir un problème	15/10/2015	18/11/2015
1.4) Etude approfondie	19/11/2015	02/01/2016
1.4.1) Etude des modèles	25/11/2015	10/12/2015
1.4.2) Etude des algorithmes	02/12/2015	02/01/2016
1.5) Rédaction du rapport	04/12/2015	04/01/2016

Figure 1 – *Les tâches réaliées*

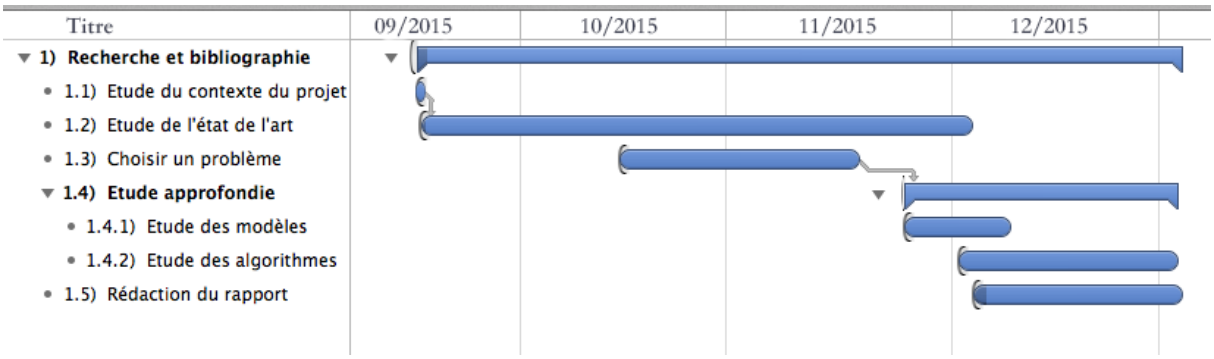


Figure 2 – *Gantt des tâches réaliées*

La liste des tâches réalisées au deuxième semestre. En fait, j'ai commencé à faire la deuxième partie du projet un mois avant la fin du premier semestre.

Titre	Début	Fin
1) Implémentation et Expérimentation	10/12/2015	01/04/2016
1.1) Mise en place des outils	10/12/2015	22/02/2015
1.2) Conception de nouvelles matheuristique	03/01/2016	12/01/2016
1.3) Implémentation	12/01/2016	28/02/2016
1.3.1) Implémentation de nouvelles matheuristique	12/01/2016	20/02/2016
1.3.2) Débugage	03/02/2016	28/02/2016
1.4) Expérimentation numériques	29/02/2016	23/03/2016
1.4.1) Débugage	29/02/2016	23/03/2016
2) Rapport et Soutenance	15/03/2016	01/04/2016
2.1) Rédaction du rapport	15/03/2016	23/03/2016
2.2) Préparation pour le soutenance	25/03/2016	01/04/2016

Figure 3 – *Les tâches à réaliser*

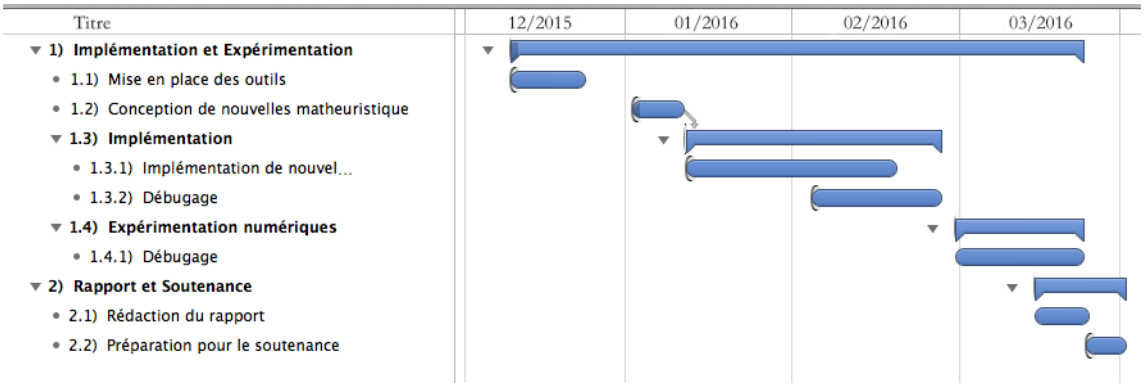


Figure 4 – *Gantt des tâches à réaliser*

2 Méthodologies et outils

La méthodologie adoptée pour l'implémentation des algorithmes c'est une méthode Agile. Pour la gestion du projet, deux outils seront été utilisés : Trello, un site web de gestion des tâches et GIT un gestionnaire de versions du code.

2.1 Environnement de développement

Les algorithmes sont développés en Java et Cplex. J'ai utilisé Eclipse comme éditeur. Les expérimentation sont menées dans un système d'exploitation Macintosh sur un MacBook Air individuel.

2.2 Les outils

CPLEX est un outil informatique d'optimisation commercialisé par IBM depuis son acquisition de l'entreprise française ILOG en 2009. Son nom fait référence au langage C et à l'algorithme du simplexe. Il est composé d'un exécutable (CPLEX interactif) et d'une bibliothèque de fonctions pouvant s'interfacer avec différents langages de programmation : C, C++, C#, Java et Python.

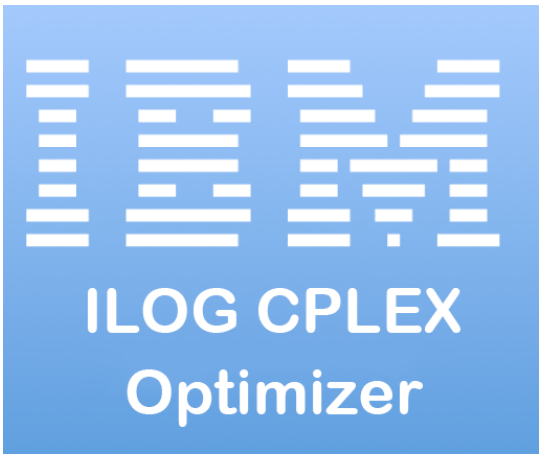


Figure 5 – *Logo de CPlex*

Eclipse est un projet, décliné et organisé en un ensemble de sous-projets de développements logiciels, de la Fondation Eclipse visant à développer un environnement de production de logiciels libre qui soit extensible, universel et polyvalent, en s'appuyant principalement sur Java.

Figure 6 – *Logo de Eclipse*

Git est un logiciel de gestion de versions décentralisé. C'est un logiciel libre créé par Linus Torvalds, auteur du noyau Linux, et distribué selon les termes de la licence publique générale GNU version 2.

Figure 7 – *Logo de Git*

Trello est un outil de gestion de projet en ligne, lancé en septembre 2011, et inspiré par la méthode Kanban de Toyota. Il est basé sur une organisation des projets en planches listant des cartes, chacune représentant des tâches. Les cartes sont assignables à des utilisateurs et sont mobiles d'une planche à l'autre, traduisant leur avancement.

Figure 8 – *Logo de Trello*

Conclusion

C'est la première fois que je travaille sur un projet de recherche et il m'a permi d'apprendre énormément de choses et de développer plusieurs capacités importantes. Premièrement, la matheuristique est une domaine très large et très riche, il existe plein d'articles dans la littérature. Donc la première capacité que j'ai obtenue est de chercher et comprendre des articles de recherche, ainsi que je dois trouver rapidement des informations utiles dans un article, pour ne pas perdre du temps sur les choses peu importantes.

Deuxièmement, c'est la capacité de comprendre des algorithmes complexes. Un algorithme complexe peut combiner de nombreuses autres méthodes existantes et utiliser des notions qui sont pas expliquées dans l'article. Ça sera difficile à comprendre si nous sommes pas familière avec des concepts et des méthodes dans cette domaine.

Sur la partie de développement, dans un premier temps, ce sont des difficultés de passer d'un algorithme qui fonctionne sur le papier, à un programme exécutable. Et puis pour proposer un nouveau algorithme, il faut vraiment beaucoup de essayes. Avec l'aide de mon encadrant, qui discute les détails de l'algorithme chaque semaine avec moi et qui propose beaucoup d'idées importantes pour moi, nous avons conçu deux nouvelles matheuristiques qui fonctionnent bien et atteint les objectifs fixés au début.

Enfin, je suis très contente d'avoir pu travailler sur un projet qui m'a permis de beaucoup progresser.

Comptes rendus hebdomadaires

Compte rendu n°1 du 16/09/2015

C'est le début du projet, j'ai lu deux articles que j'ai choisis pour me familier avec des notions basiques et comprendre le sujet du projet. Et j'ai écrit des petits résumés comme vous m'a conseillé.

Matheuristics : Optimization, Simulation and Control [4]

Dans cet article, il nous donne la définition de Matheuristique. Bref, matheuristiques sont des algorithmes heuristique fabriqués par l'inter-opérations de métaheuristiques et la programmation mathématique (MP).

Le mot métaheuristique est dérivé de la composition de deux mots grecs : heuristique qui signifie "trouver" et meta qui est un suffixe signifiant "au-delà". Ça veut dire que une métaheuristique est une heuristique dans un niveau supérieur. Une métaheuristique est un algorithme d'optimisation visant à résoudre des problèmes d'optimisation difficile (souvent issus des domaines de la recherche opérationnelle, de l'ingénierie ou de l'intelligence artificielle) pour lesquels on ne connaît pas de méthode classique plus efficace.

Une métaheuristique est formellement définie comme un processus de génération itérative qui guide une heuristique subordonnée en combinant intelligemment différents concepts pour explorer et exploiter l'espace de recherche.

Et puis, cet article présente deux façon de la combinaison de métaheuristique et MP :

1. Améliorer ou concevoir une métaheuristique utilisant MP.
2. Améliorer des techniques de MP connus utilisant métaheuristique.

Bien sûr, l'objet de métaheuristique est de faire des optimisations, et cet article aussi nous présente un concept important : simulation, qui a des relations étroites avec optimisations. Pour moi, la simulation ressemble beaucoup au processus de modéliser des problèmes.

Contrôle de processus est "la tâche de planification et de régulation d'un processus, avec l'objectif de l'accomplir de manière efficace, effective et cohérente". Trouver une manière plus efficace et effective, c'est aussi une problème de l'optimisation, donc matheuristiques peuvent aussi contribuer à des problèmes de contrôle de processus.

Matheuristics : Hybridizing Metaheuristics and Mathematical Programming [13]

C'est un livre plutôt. Et ce livre montre comment les deux : métaheuristiques et MP peuvent influencer sur l'autre. Mais comme il est vraiment trop longue, j'ai lu juste le première chapitre. Il s'agit des concepts de base comme heuristique, heuristique glouton, métaheuristique et la recherche locale.

1. Une heuristique est approximativement dans le sens qu'elle fournit une bonne solution pour relativement moins d'efforts, mais il ne garantit pas l'optimalité. Heuristiques sont des critères, méthodes ou principes pour décider qui, parmi plusieurs cours alternatifs d'action, promet d'être le plus efficace afin d'atteindre un objectif.
2. Heuristiques gloutonnes sont des approches itératives simples disponibles pour tout type de problème d'optimisation. Une bonne caractérisation est leur comportement myope.
3. Une métaheuristique est une stratégie de haut niveau qui guide une heuristique sous-jacente résolvant un problème donné.
4. Le principe de base de la recherche locale est de modifier les solutions successivement et localement.

Dans ce livre, il y a aussi des problèmes spécifiques utilisant algorithme matheuristique, y compris "The problem of balancine transfer lines", "Mixed intègre non-linear programming" etc... Mais comme on s'intéresse plutôt les problèmes de tournée, j'ai pas pris du temps pour lire tous ça.

Compte rendu n°2 du 01/10/2015

Dans cette semaine, j'ai commencé l'étude bibliographique, je cherche des articles sur matheuristique et problème de tournée sur Google Scolar, et mon encadrant aussi m'envoye des articles. Je fait mes choix dans un grande nombre de l'article.

The Vehicle Routing Problem : An overview of exact and approximate algorithms [12]

Le problème de tournées de véhicules est une classe de problèmes de recherche opérationnelle et d'optimisation combinatoire. Le but est de minimiser le coût de livraison des biens. Ce problème est une extension classique du problème du voyageur de commerce, et fait partie de la classe des problèmes NP-complet.

Il présente une définition générale du problème de tournée et puis il liste les algorithmes exacts et heuristiques :

Algorithmes exacts :

The assignment lower bound and a related branch-and-bound algorithm

The k-degree center tree and a related algorithm

Dynamic programming

Set partitioning and column generation

A three-index vehicle flow formulation

Algorithmes heuristiques :

The Clarke and Wright algorithm

The sweep algorithm

The Christofides-Mingozzi-Toch two-phase algorithm

A tabu search algorithm

The Vehicle Routing Problem [7]

Dans cet article, il présente en général le problème de tournée, et puis la structure de solutions. Il y a aussi des méthodes de solutions exactes ou heuristiques comme l'article dernier.

A matheuristic for the truck and trailer routing problem [19]

Remarque : pour les articles que j'ai presenté dans mon rapport, je vais pas répéter ici.

Compte rendu n°3 du 08/10/2015

Je continue de lire des articles cette semaine. Et dans cette semaine, j'apprends rédiger mon compte rendu hebdomadaire en LaTex, j'ai jamais utilisé LaTex avant, donc c'est pas très facile à commencer. J'utilise l'éditeur TexShop sur Mac.

A matheuristic for the school bus routing problem [18]

Dans cet article, ils proposent une formulation de MIP pour "school bus routing problem" dans lequel la sélection d'arrêts d'un ensemble d'arrêts potentiels et répartition des étudiants à des arrêts sont des décisions supplémentaires variables. Ils proposent une GRASP + VND matheuristique qui utilise une procédure de programmation linéaire exacte pour résoudre le sous-problème d'affectation des étudiants aux arrêts.

La matheuristique se compose de deux phases. La phase de construction utilisant l'idées de GRASP. Et la phase d'amélioration est une "variable neighborhood descent method" (VND).

Une attention particulière est posée sur la sélection d'arrêt, qui distingue ce problème de les autres problèmes de tournée traditionnels.

Il n'utilise pas un solveur dans son algorithme en fait, mais il compare ses résultats avec des résultats d'un MIP solveur.

A variable neighborhood search based matheuristic for nurse rostering problems [8]

A matheuristic approach is introduced, based on a set of neighborhoods iteratively searched by a commercial integer programming solver within a defined global time limit, relying on a starting solution generated by the solver running on the general integer programming formulation of the problem.

C'est un "nurse rostering problems", qui fait partie de problème "Home Health Care Problem".

A matheuristic approach for solving a home health care problem [1] dans le rapport.

Compte rendu n°4 du 15/10/2015

Discussion sur le Pollution-Routing Problem [11] avec mon encadrant.

A survey on pickup and delivery problems [14]

Cet article est organisé comme suit : tout d'abord, les formulations de modèle pour l'unique ainsi que plusieurs véhicules sont présentés afin de définir clairement les sous-classes de "pickup and delivery problem". Ensuite, pour chaque classe de problème un aperçu des différentes méthodes de résolution proposés est donnée. Enfin, il y a une subsection de conclusion qui fournit quelques conseils sur les meilleures approches actuellement pour chaque catégorie de problème et aussi donne des directions pour la recherche future.

Dans cet article, il nous donne un modèle plutôt général pour les différents versions de problèmes "Pickup and Delivery". Il liste les populaires solutions de différents problèmes et leurs efficacité mais sans détails des algorithmes.

Compte rendu n°5 du 22/10/2015

Depuis la semaine dernière, j'ai commencé de choisir un problème pour l'étude suivante. Mon encadrant veut que c'est un problème utilise intelligemment, pas trop cherché et pas trop compliqué.

matheuristic for the liner shipping network design problem [5]

The liner shipping network design problem (LSNDP) est de construire un ensemble de services cycliques pour former un réseau du transport de marchandises conteneurisées. La conception du réseau maximise les revenus du transport de conteneurs compte tenu du coût des navires déployés aux services, la consommation totale de carburant, des coûts de port et des frais de manutention.

Le matheuristic développé pour le LSNDP est une composition des quatre procédures suivantes :

1. heuristique de construction avec plusieurs redémarrages.
2. Amélioration heuristique.
3. Réinsertion heuristique.
4. Perturbation heuristique.

L'amélioration heuristique utilise un solveur.

A matheuristic for aggregate production–distribution planning with mould sharing [15]

Cet article étudie le problème de production-distribution globale pour un fabricant de produits en plastique qui sont produites avec le moulage par injection.

Il nous présente deux modèle mathématique possible et les résout avec un solveur MILP.

Pour les plus large instances, il les décomposés par détendant les constraintes. Bien que ces problèmes décomposés sont encore NP-difficile, les tailles sont suffisamment petits pour être résolu par un solveur.

Compte rendu n°6 du 05/11/2015

Discussion sur le truck and trailer routing problem [19] et le The Multi-Commodity One-to-One Pickup-and-Delivery Traveling Salesman Problem [17] avec mon encadrant.

Compte rendu n°7 du 19/11/2015

Pendant cette semaine, nous avons décidé de choisir le The Multi-Commodity One-to-One Pickup-and-Delivery Traveling Salesman Problem pour l'étude suivante, donc je cherche plus de articles sur ce sujet et je les lis avec plus de soin.

Compte rendu n°8 du 03/12/2015

Pendant cette semaine, j'ai mise en place le modèle du rapport. Comme je suis pas familière avec Latex, j'ai rencontré plein de problèmes. Avec l'aide de Monsieur Aupetit et mes camarades, j'ai résolu certains problèmes de compilation, mais le problème de la bibliographie est toujours pas résolu. Enfin je décide de utiliser un éditeur sur ligne Overleaf pour rédiger mon rapport.

Compte rendu n°9 du 10/12/2015

J'ai lu des documents de l'introduction du solveur et décidé d'utiliser IBM Cplex comme le solveur pour le développement. Mon encadrant m'envoye plusieurs Cplex Tutorial et les exemples qu'il a fait pour d'autres problèmes.

Ensuite j'ai mise en place IBM Cplex dans mon mac et lu des tutorials pour comprendre comment il fonctionne, et j'ai aussi essayé de executer des exemples simples.

Compte rendu n°10 du 17/12/2015

Ces jours-ci, Je prends du temps pour rédiger mon rapport utilisant le modèle que Monsieur Aupetit a nous envoyé. Il y a encore des difficultés mais ça avance quand même.

Pour les vacances de Noël : D'abord, je vais finir ma première version du rapport. Ensuite, je vais réaliser au moins un algorithme avec C++ et Cplex.

Ci-joint la structure préliminaire de mon rapport, elle indique juste des sections et sub-sections, je vais le détailler en train de rédiger. Et les sections en rouge sont les travaux du prochain semestre. Je vais l'améliorer en train de rédiger.

La structure du rapport :

- Introduction
- 1 État de l'art
- 1.1 Matheuristique
- 1.2 Problème de tournée
- 1.3 Exemples d'utilisation de matheuristique
- 2 Étude approfondie du problème m-PDTSP
- 2.1 Description du problème
- 2.2 Modélisation du problème
- 3 Algorithmes existants pour m-PDTSP
- 3.1 Algorithme de décomposition
- 3.2 MIP-based GRASP
- 4 Nouvelle matheuristique proposée
- 5 Mise en place des tests
- 6 Poursuite de projet
- 6.1 Planning prévisionnel
- 6.2 Outils et logiciels de suivi de projet
- 7 Gestion de projet
- 7.1 Découpage du projet

- 7.2 La planification du projet
- 7.3 Diagramme de Gantt
- 8 Conclusion

Compte rendu n°11 du 07/01/2015

Discussion sur les nouveaux algorithmes. Nous voulons concevoir une matheuristique classique, qui utilise un solveur intelligement, ça veut dire qu'elle utilise beaucoup de fois de solveur. Donc on compte concevoir un algorithme qui appartient à la deuxième groupe de matheuristique étudiée : Heuristique d'amélioration.

Compte rendu n°12 du 14/01/2016

Mise en oeuvre le Cplex avec Eclipse, et se familier avec les utilisations du Cplex par plusieurs exemples simples.

Compte rendu n°13 du 28/01/2016

Implémentation du modèle avec Java et Cplex pour résoudre les petites instances. Il peut résoudre les instances jusqu'à 10 sommets (8 clients et 2 sommets pour le dépôt) et 5 produits à moins de 1 seconde et obtenir une solution optimale exacte.

Compte rendu n°14 du 05/02/2016

Pour augmenter l'efficacité, nous avons fait des étapes suivants :

1. Créer $(n+1)\times(n+1)$ variables x_a au lieu de $(n+2)\times(n+2)$, parce qu'il n'y pas de arc qui part du sommet $n+1$, et il n'y a pas de arc qui arrive au sommet 0.
2. Fixer les variables représentant les arcs (i,j) où $i=j$ à 0, parce que nous pouvons pas partir d'un sommet et arriver au même sommet.
3. Dans la contrainte (2), on enleve tous les sous-ensembles qui a qu'un seul sommet, parce que ce sont les mêmes contraintes que la contrainte (1).

$$x(\delta^+(\{i\})) = 1 \quad \text{pour tout } i \in \mathrm{V}\setminus\{n+1\} \tag{1}$$

$$x(\delta^+(\mathrm{S})) \geqslant 1 \quad \text{pour tout } \mathrm{S}\subset \mathrm{V} : 0\in \mathrm{S}, n+1 \notin \mathrm{S} \tag{2}$$

Compte rendu n°15 du 12/02/2016

Débuggage du grogramme. Des tests sur plusieurs instances différentes.

Compte rendu n°16 du 26/02/2016

Nous avons proposé une nouvelle matheuristique pour résoudre ce problème. Premièrement, on utilise une heuristique "plus proche voisin" pour générer une solution initiale qui respecte aussi les contraintes des précédence, et puis on décompose la solution initiale en plus petit morceaux et les résout avec le modèle susmentionné. C'est-à-dire que nous utilisons le modèle MIP pour l'optimisation locale, donc notre matheuristique appartient à la deuxième famille de matheuristique "Heuristique d'amélioration" que nous avons présenté dans le rapport du dernier semestre.

(Les pseudos codes sont presques les mêmes comme dans le rapport, je répète pas ici)

Recalculer la matrice de demandes :

Dans le graphe original, le premier sommet $0 \in \mathrm{A}$ et le dernier sommet $n+1 \in \mathrm{A}$ n'ont pas d'offres ou demandes des produits. Mais dans le modèle modifié, comme on considère juste un certain nombre de sommets consécutifs A' dans la solution initiales, on perdra des informations sur les demandes des produits. Alors, on va prendre la somme des offres de tous les sommets dans A – A' et le premier sommet dans A' comme l'offre des produits du premier sommet dans A', et la somme des demandes de tous les sommets dans A – A' et le dernier sommet dans A' comme la demande des produits du dernier sommet dans A'.

Recalculer la matrice de distances :

Dans cet algorithme, parce que nous fixons les sommet consécutifs, les distances entre les sommet sélectionnés ne changent pas par rapport au graphe original.

Compte rendu n°17 du 03/03/2016

Ajoute quelques fixations des variables pour déduire le temps de l'exécution de l'algorithme. Voir subsubsection 2.3.2 (Chapitre 3)

Compte rendu n°18 du 10/03/2016

Débuggage du grogramme.

Compte rendu n°19 du 17/03/2016

Rédaction du rapport.

Webographie

[WWW1] Wikipedia. *Wikimedia Commons*. url : https://commons.wikimedia.org/wiki/File:Map_of_vrp_subproblems.jpg.

Bibliographie

[1] Hanane Allaoua, Sylvie Borne, Lucas Létocart et Roberto Wolfler Calvo. « A matheuristic approach for solving a home health care problem ». In : *Electronic Notes in Discrete Mathematics* 41 (2013), p. 471–478.

[2] Claudia Archetti et M. Grazia Speranza. « A survey on matheuristics for routing problems ». In : *EURO J Comput Optim* 2 (août 2014), p. 223–246.

[3] T. Bektas et G. Laporte. « The pollution-routing problem ». In : *Transportation Research Part B : Methodological* 45 (2011), p. 1232–1250.

[4] Marco A. Boschetti, Vittorio Maniezzo, Matteo Roffilli et Antonio Bolufé Röhler. « Matheuristics : Optimization, Simulation and Control ». University of Bologna, Bologna, Italy. 2009.

[5] Berit Dangaard Brouer, Guy Desaulniers et David Pisinger. « A matheuristic for the liner shipping network design problem ». In : *Transportation Research Part E* 72 (2014), p. 42–59.

[6] I.-M. Chao. « A tabu search method for the truck and trailer routing problem ». In : *Computers and Operations Research* 29 (2002), p. 33–51.

[7] Nicos Christofides. « The Vehicle Routing Problem ». In : *R.A.I.R.O Recherche opérationnelle* 59.2 (1976), p. 345–358.

[8] Federico Della Croce et Fabio Salassa. *A variable neighborhood search based matheuristic for nurse rostering problems*. Published online. Springer Science+Business Media New York 2012, nov. 2012.

[9] J.C. Gerdessen. « Vehicle routing problem with trailers ». In : *European Journal of Operational Research* 93 (1996), p. 135–147.

[10] Hipólito Hernández-Pérez et Juan-José Salazar-González. « The multi-commodity one-to-one pickup-and-delivery traveling salesman problem ». In : *European Journal of Operational Research* 196 (2009), p. 987–995.

[11] Raphael Kramer, Anand Subramanian, Thibaut Vidal et Lucídio dos Anjos F. Cabral. « A matheuristic approach for the Pollution-Routing Problem ». In : *European Journal of Operational Research* 243 (2015), p. 523–539.

[12] Gilbert Laporte. « The Vehicle Routing Problem : An overview of exact and approximate algorithms ». In : *European Journal of Operational Research* 10 (fév. 1992), p. 55–70.

[13] Vittotio Maniezzo, Thomas Stützle et Stefan VoB. *Matheuristics : Hybridizing Metaheuristics and Mathematical Programming*. 1re éd. T. 10. Library of Congress Control Number : 2009935392. Springer New York Dordrecht Heidelberg London : Springer Science+Business Media, 2009. isbn : 978-1-4419-1305-0.

[14] Sophie N. Parragh, Karl F. Doerner et Richard F. Hartl. « A survey on pickup and delivery problems ». In : *JfB* 58 (2008), p. 21–51.

[15] Birger Raa, Wout Dullaert et El-Houssaine Aghezzaf. « A matheuristic for aggregate production–distribution planning with mould sharing ». In : *Int. J. Production Economics* 145 (2013), p. 29–37.

[16] Righini, G. et M. Salani. « Decremental state space relaxation strategies and initialization heuristics for solving the orienteering problem with time windows with dynamic programming ». In : *Computers and Operations Research* 36 (2009), p. 1191–1203.

[17] I. Rodríguez-Martín et Juan José Salazar-González. « The Multi-Commodity One-to-One Pickup-and-Delivery Traveling Salesman Problem : A Matheuristic ». DEIOC, Universidad de La Laguna, Spain. 2011.

[18] Patrick Schittekat, Kenneth Sörensen, Marc Sevaux et Johan Springael. « A matheuristic for the school bus routing problem ». RESEARCH PAPER. Mém.de mast. University of Antwerp, City Campus, Prinsstraat 13, B-2000 Antwerp, Belgium Research Administration – room B.213 : UNIVERSITY OF ANTWERP, oct. 2009.

[19] Juan G. Villegas, Christian Prins, Caroline Prodhon, Andrés L. Medaglia et Nubia Velasco. « A matheuristic for the truck and trailer routing problem ». In : *European Journal of Operational Research* 230 (avr. 2013), p. 231–244.

Projet Recherche & Développement

Nouvelles matheuristiques: implémentations et expérimentations : Application aux problèmes de tournée

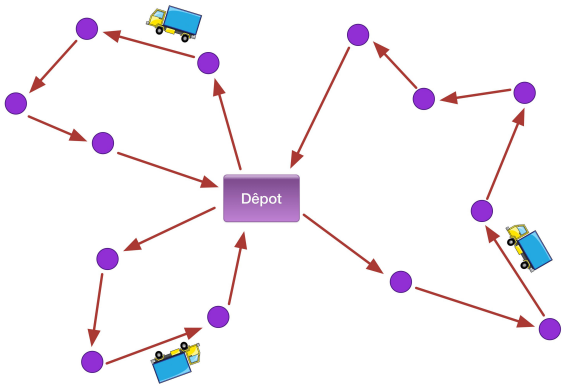
Yan LI

Encadrement : Jean-Louis Bouquard

En collaboration avec
Laboratoire Informatique

Objectifs

- Comprendre comment utiliser des matheuristiques pour résoudre les problèmes de tournée de véhicule
- Choisir un problème de tournée
- Implémenter différentes matheuristiques sur le problème choisi
- Evaluer et apprécier les différentes matheuristiques



problème de tournée

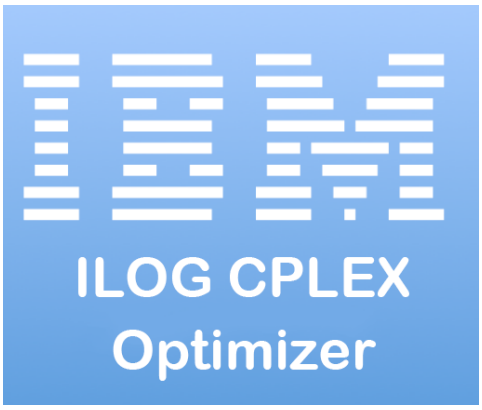
Matheuristique

Nous pouvons classifier les matheuristiques en trois grandes familles:

- Approches basées sur la génération de colonnes (Set-Partitioning)
- Heuristiques d'amélioration
- Approches de décomposition

Résultats attendus

- Compréhension des trois grandes familles de matheuristique par plusieurs exemples d'utilisation.
- Une étude plus approfondie sur le problème one-to-one m-PDTSP *Multi-Commodity One-to-One Pickup-and-Delivery Traveling Salesman Problem*



IBM ILOG CPLEX

Ecole Polytechnique de l'Université François Rabelais de Tours
Département Informatique
64 avenue Jean Portalis, 37200 Tours, France
www.polytech.univ-tours.fr

POLYTECH TOURS
Informatique

Nouvelles matheuristiques: implémentations et expérimentations

Application aux problèmes de tournée

Résumé

Rapport du Projet de Recherche & Développement, Au début de ce projet, nous étudions l'état de l'art des matheuristiques pour les problèmes de tournée de véhicules, ce qui nous permet de mieux comprendre les notions de base dans ce domaine et de nous familiariser avec certains grandes familles de problèmes et d'algorithmes. Ensuite, nous nous concentrons sur le problème one-to-one m-PDTSP (*The Multi-Commodity One-to-One Pickup-and-Delivery Traveling Salesman Problem*), qui est une extension du problème classique du voyageur de commerce. Une étude approfondie, de la modélisation à l'expérimentation, est menée sur ce problème. Dans la partie développement du projet, nous avons proposé deux nouvelles matheuristiques pour le problème one-to-one m-PDTSP, nous avons les implémenté en utilisant Java et Cplex. A la fin de ce rapport, nous avons comparé les résultats expérimentaux des nos deux algorithmes avec les résultats optimaux qui sont déjà connus.

Mots-clés

Matheuristique, problème de tournée, one-to-one m-PDTSP, Cplex

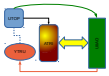
Abstract

This is the report of the project R&D. In this part of the project, first of all, we study the state of the art of matheuristics for routing problems, this allows us to understand the basic concepts in this area and to become familiar with several major categories of routing problems and algorithms. Then we focus on the Multi-Commodity One-to-One-Pickup and Delivery Traveling Salesman Problem (one-to-one m-PDTSP), this is an extension of classical Travelling Salesman Problem. A thorough study from modeling to experiments is conducted on this problem. In the development part of the project, we have proposed two new matheuristics for the problem one-to-one m-PDTSP, we have implemented them using Java and Cplex. At the end of this report, we compared the experimental results of the two algorithms with the optimal results that are already known.

Keywords

Matheuristic, routing problem, one-to-one m-PDTSP, Cplex

Entreprise
Laboratoire Informatique



Étudiants
Yan LI (DI5)

Tuteurs académiques
Jean-Louis Bouquard

Tuteurs entreprise
Jean-Louis Bouquard