Université de Nantes Année 2017/2018

Rapport de stage - Licence 3 Informatique parcours Mathématiques

Conception d'une matheuristique pour la résolution du Pickup and Delivery Problem with Time Windows

Auteur : LATIF Mehdi

Tuteurs : LEHUEDE Fabien, TONNEAU Quentin

Le 4 juin 2018

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué au succès de mon stage et qui m'ont aidé lors de la rédaction de ce rapport.
Tout d'abord, j'adresse mes remerciements à mon professeur, Monsieur Evgeny Gurevsky, enseignant chercheur l'Université de Nantes qui m'a permis de réaliser ce stage.
Je tiens à remercier vivement mes maitres de stage, Messieurs Fabien Lehuédé et Quentin Tonneau pour leur accueil, leur écoute et leur accompagnement pendant toute la durée de cette mission.
Je remercie également toute l'équipe SLP pour leur accueil.

Introduction

Aujourd'hui, le développement de la chaine logistique est devenu une problématique essentielle pour les entreprises industrielles et de services. L'aspect économique, notamment dans l'objectif de réduction des coûts de transport associé à une volonté d'assurer une meilleure satisfaction client et ceci dans des délais minimaux, représente un enjeu global pour tous les secteurs. Cette problématique est aussi très étudiée dans le milieu de la recherche depuis les années 1960.
Ainsi, ce stage réalisé dans le cadre de ma troisième année de licence de mathématiques et d'informatique, a pour objectif l'étude d'un problème de collecte et de livraison assorti de contraintes temporelles. La mission qui m'a été confiée durant ce stage a été le développement d'une méthode exacte, le problème de couverture par ensembles, qui s'inscrit dans une matheurstique permettant de résoudre la problématique présentée précédemment.
Dans un premier temps, nous étudierons plus en détail ce sujet grâce aux méthodes et outils acquis durant mon cursus universitaire puis dans un second temps, nous présenterons les implémentations réalisées pour résoudre ce problème. Enfin, nous terminerons par une analyse expérimentale réalisée avec l'algorithme que nous aurons implémenté.
Ce stage a été encadrée par Monsieur Fabien Lehuédé, responsable de l'équipe SLP au sein du LS2N ainsi qu'enseignement chercheur à l'IMTA et par Monsieur Quentin Tonneau, également membre de l'équipe SLP et enseignant chercheur à l'Université de Nantes. De plus, ce projet a été réalisé avec le concours d'un autre étudiant de licence d'informatique, Emmanuel Rochet.

Table des matières

1 Présentation de l'équipe de recherche Système Logistique et de Production

Ce stage de découverte du monde de la recherche académique a été réalisé au sein de l'équipe Système Logistique et de Production, spécialisée dans les domaines de la recherche opérationnelle ainsi que la maitrise des risques pour la conception et l'optimisation des systèmes de production, la logistique et le transport.
Cette composante du LS2N (Laboratoire des Sciences du Numérique de Nantes) a comme objectif principal de résoudre, et ceci de manière efficace, des problèmes de la littérature dont les résultats obtenus à ce jour ne sont pas optimaux. Pour ce faire, les membres de l'équipe développent des méthodes d'optimisation, propres aux spécifités de chaque problème et ceci, par l'utilisation de méthodes existantes dans ou par l'implémentation de méthodes exactes, d'heurstiques ou encore de métaheuristiques [1].
Les principaux thèmes de cette structure sont :

- La maîtrise des risques et l'aide à la décision pour les systèmes industriels et les services.
- La conception, planification et ordonnancement des systèmes de production et de services
- La conception et optimisation des réseaux logistiques et de transport.
- La recherche fondamentale (conception de métaheursitique, optimisation multiobjectif, ...)

Cette équipe, sous la responsabilité de Monsieur Fabien Lehuédé, regroupe une vingtaine de chercheurs, une dizaine de doctorants sur les sites de l'UFR de sciences et techniques ainsi que sur le site de l'Institut Mines-Télécom Atlantique.

1. Une heuristique est une stratégie de résolution empirique qui exploite la structure d'un problème. Souvent sous-optimale, elle permet cependant de converger vers des résultats optimaux. Une métaheuristique Une métaheuristique est une marche aléatoire dans l'espace de recherche guidée par des heuristiques et ainsi d'explorer les zones qui paraissent prometteuses sans se restreindre à un optimum local. Une métaheuristique est une méthode de résolution qui peut s'appliquer à de nombreuses catégories de problèmes différents

2 Présentation du Pickup and Delivery Problem with Time Windows

Le Pickup and Delivery Problem with Time Windows (ou PDPTW) est une variante du Vehicle Routing Problem (ou VRP) proposé en 1959 par Dantzig et Ramser[4] dont l'objectif est de déterminer les tournées d'une flotte de véhicules afin de livrer une liste de client tout en veillant à minimiser le coût de livraison des biens.
Pour obtenir le PDPTW à partir du problème proposé par Dantzig et Ramser, nous déclarons plusieurs contraintes additionnelles telle que :

- Des contraintes de capacités imposant aux véhicules une capacité d'emport limitée.
- Des contraintes sur fenêtres de temps *i.e.* un intervalle temporel durant lequel un véhicule peut visiter un client.
- Des contraintes sur les temps de service *i.e.* une durée pour les opérations réalisées chez chaque client.
- Des contraintes sur le type d'opérations qui peuvent être réalisées chez chaque clients (*e.g.*, opération de collecte ou de livraison).

Une solution au PDPTW est un ensemble d'itinéraires (ou routes) consistant en séquence de visites aux clients, où chaque itinéraire est assigné à un véhicule et tous les clients sont visités durant leurs fenêtres de temps. L'objectif du PDPTW consiste à déterminer une solution qui minimise la distance totale parcourue ainsi que le nombre de véhicules utilisés.

Dans cette partie, nous allons tout d'abord présenter le PDPTW grâce à un exemple d'application pour une entreprise de livraison puis par sa modélisation sous forme d'un programme linéaire ; dans un second temps, nous expliciterons la matheuristique que nous avons mise en place pour résoudre ce problème.

2.1 Exemple introductif

L'exemple que nous allons présenter a été élaboré à partir de la description du problème proposé par Ropke et Pisinger[6]. Il s'agit de vous présenter de manière plus concrète les diférents aspects du problème que nous tentons de résoudre.

L'entreprise DELIVREMOITOUT est une toute nouvelle société créée par des experts en recherche opérationnelle. Elle est spécialisée dans la livraison de repas gastronomiques préparés par des restaurateurs renommés, livrés directement chez des particuliers par une flotte de cyclistes.
Grâce à son site internet, les clients renseignent leurs commandes en spécifiant :

- Le nom du restaurant dans lequel ils souhaitent commander.
- La quantité de plats qu'ils souhaitent recevoir.
- Une fenêtre horaire durant laquelle ils souhaitent être livrés.

La politique de gestion des livraisons spécifie que les commandes doivent être passées une demi-journée avant la livraison afin que les restaurateurs aient le temps d'acheter des produits frais et de préparer les repas,ce qui peut prendre un peu de temps.
Une fois les commandes effectuées, l'entreprise DELIVREMOITOUT transmet les commandes aux restaurateurs qui vont quant à eux, définir une fenêtre de temps durant laquelle les cyclistes pourront venir

retirer les repas.
On prévoit un temps de service qui indique la durée nécessaire pour effectuer les retraits des repas chez les restaurateurs et les livraisons chez les clients. Dans cet exemple, on supposera que l'ensemble des clients effectuent leurs commandes dans des restaurants distincts.
La flotte de l'entreprise DELIVREMOITOUT est composée d'un nombre limité de cyclistes. Ces derniers ont un sac à dos dont la capacité est limitée et doivent, avant de commencer leurs livraisons, se rendre au siège de l'entreprise [1] récupérer leurs vélos et venir les redéposer après les livraisons. Une commande, composée d'un point de collecte et d'un point de livraison, doit évidemment être servie par le même cycliste ; ce dernier a la possibilité de prendre en charge plusieurs commandes sur un même voyage. Un cycliste peut arriver en avance dans un restaurant ou chez un client mais il devra attendre le début de la fenêtre de temps pour récupérer ou livrer un repas.
Durant la demi-journée précédant les livraisons, l'entreprise DELIVREMOITOUT doit optimiser les itinéraires, c'est-à-dire une suite de points de collecte et de livraison de repas, empruntés par sa flotte de livreurs et ceci en minimisant la distance totale parcourue par les cyclistes, le temps total passé [2] par chaque cycliste ainsi que le nombre de commandes qui ne pourra être traité par ces derniers. Les itinéraires proposés par l'entreprise doivent respecter les fenêtres de temps imposées par les acteurs de la chaine de livraison ainsi que la contrainte de capacité. En effet, un cycliste ne doit pas se retrouver avec plus de repas qu'il ne peut en transporter et il se doit d'arriver à chaque point de collecte et de livraison dans la fenêtre de temps qui est associée à ce point.. Dès lors, chaque cycliste se voit remettre une feuille de route optimisée contenant son horaire de départ de l'entreprise, les horaires auxquels il doit se présenter dans un restaurant ou chez un client ainsi que l'heure à laquelle il est autorisé à quitter ces derniers pour qu'il puisse répondre à la commande suivante inscrite sur sa feuille de route.

Ce soir-là, l'entreprise a reçu 53 commandes clients qui sont détaillées ci-dessous :

n° Commande	n° Restaurant	n° Client	Quantité à livrer	n° Commande	n° Restaurant	n° Client	Quantité à livrer
1	4	76	10	28	52	102	10
2	6	8	10	29	54	59	20
3	7	3	20	30	55	61	40
4	9	11	20	31	57	60	30
5	10	5	10	32	58	56	40
6	12	2	10	33	63	69	20
7	14	18	30	34	64	75	50
8	17	15	40	35	65	103	10
9	19	13	20	36	66	73	10
10	20	16	10	37	67	70	10
11	21	25	10	38	68	62	10
12	24	104	10	39	72	78	20
13	26	28	40	40	77	74	10
14	29	23	20	41	79	105	20
15	30	27	10	42	80	81	10
16	31	22	10	43	82	71	30
17	33	32	30	44	83	86	20
18	34	38	40	45	85	90	20
19	36	40	10	46	87	92	10
20	37	106	10	47	88	84	20
21	39	35	30	48	91	89	10
22	43	41	20	49	93	94	20
23	44	42	10	50	97	95	10
24	45	47	10	51	98	107	30
25	46	49	10	52	99	96	20
26	50	48	10	53	101	100	20
27	51	53	10				

TABLE 2.1 – Instance de benchmark pour le problème du PDPTW avec 100 clients : Li & Lim's PDPTW benchmark problems.

L'analyse des positions géographiques des restaurants et clients récoltées lors des commandes permet

1. Aussi appelé dépôt dans la suite du rapport
2. On définit le temps passé par un cycliste comme la différence entre l'horaire d'arrivée au siège de l'entreprise et l'horaire de départ depuis ce même siège.

à l'entreprise de générer la carte suivante.

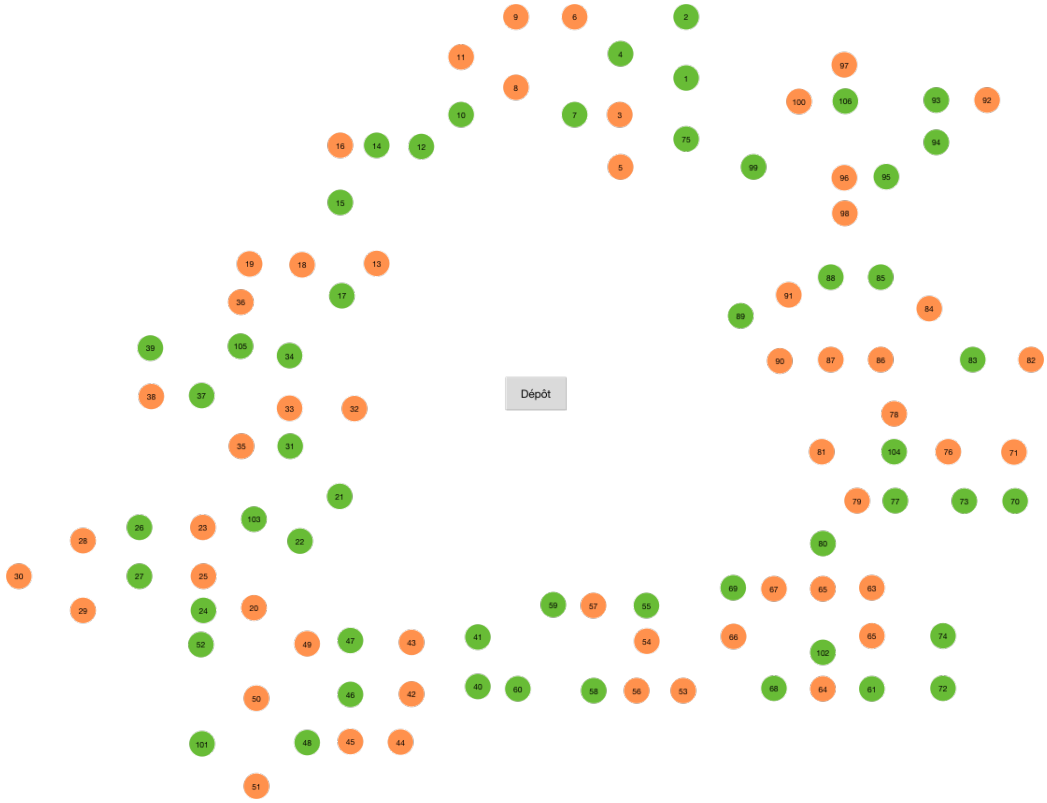

FIGURE 2.1 – Représentation graphique des points de collecte (nœuds oranges) et de livraison (nœuds verts) - Instance à 100 clients issue du benchmark de Li & Lim

Dans ce contexte, l'objectif de l'entreprise DELIVREMOITOUT est donc d'utiliser des méthodes de résolution, exactes ou approchées, pour définir des itinéraires optimaux afin livrer l'ensemble de ces repas.

2.2 Modélisation en programmation mathématique du PDPTW

Dans leurs articles paru en 2006, Ropke et Pinsinger[6] proposent une modélisation complète du PDPTW suivante.

On suppose une instance du PDPTW composée de n requêtes et m véhicules.

Déclaration des paramètres

Le problème se défini comme un graphe $G(\mathcal{P} \cup \mathcal{D}, E, \mathcal{R})$ où $\mathcal{P} = \{1, \dots, n\}$ est un ensemble de nœuds de collecte, $\mathcal{D} = \{n+1, \dots, 2n\}$ l'ensemble des nœuds de livraison, $\mathcal{R}$ l'ensemble des requêtes de l'instance considérée appelé *requestBank* et E représente la liste des arrêtes.
Une requête i est représentée par le couple de nœuds $(i, n+i) \in \mathcal{P} \times \mathcal{D}$. On définit K, l'ensemble des véhicules alloués pour cette instance du problème tel que $|K| = m$. Il est possible qu'un véhicule $k \in K$ ne soit pas en mesure de répondre à toutes les demandes qui lui sont imposées [3] ; On définit alors K_i tel que $K_i \subseteq K$ comme le sous ensemble des véhicules capables de servir la requête i. De plus, on pose

3. Par exemple, dans la situation où les biens nécéssitent des conditions spéciales pour leur transport (exemple : une marchandise nécessitant un camion frigorifique)

$\mathcal{P}_k \subseteq \mathcal{P}$ (respectivement $\mathcal{D}_k \subseteq \mathcal{D}$), un ensemble de points de collecte (respectivement de livraison) tels que $\mathcal{P}_k$ et $\mathcal{D}_k$ sont des sous ensembles de points pouvant être servis par le véhicule $k \in K$; On a ainsi

$$\forall i \in \mathcal{R}, \forall k \in K_i \Leftrightarrow i \in \mathcal{P}_k \wedge i \in \mathcal{D}_k$$

On définit de $N = \mathcal{P} \cup \mathcal{D}$, l'ensemble de tous les nœuds et $N_k = \mathcal{P}_k \cup \mathcal{D}_k$, l'ensemble de tous les nœuds pouvant être servis par le véhicule $k \in K$.
On pose $\tau_k = 2n + k$, $k \in K$ le point de départ (ou dépot) du véhicule k et $\tau'_k = 2n + k + m$, $k \in K$, le point d'arrivée (dépot) d'un véhicule k [4].

Soit $G = (V, A)$ avec $V = N \cup \{\tau_1, \ldots, \tau_m\} \cup \{\tau'_1, \ldots, \tau'_m\}$, l'ensemble des nœuds et $V = A \times A$, l'ensemble des arcs entre les nœuds.
Pour chaque véhicule $k \in K$, on définit le sous graphe $G_k = (V_k, A_k)$ où $V_k = N_k \cup \{\tau_k\} \cup \{\tau'_k\}$ et $A_k = V_k \times V_k$.

Pour chaque arc $(i, j) \in A$, on déclare les variables :

$d_{i,j}$, une distance non négative entre les nœuds i et j

$t_{i,j}$, un temps de trajet non négatif entre les nœuds i et j

Notons que les temps de trajets verifient l'inégalité triangulaire *i.e.*

$$\forall i, j, k \in V,\ t_{i,j} \leq t_{i,k} + t_{k,j}$$

Pour chaque nœud $i \in V$, on pose :

s_i, le temps de service qui représente les durées de chargement et de déchargement des biens à un nœud i

$[a_i, b_i]$, une fenêtre de temps au nœud i avec :

a_i : la date d'ouverture au plus tôt de la fenêtre de temps

b_i : la date de fermeture au plus tard de la fenêtre de temps

Notons que les opérations en un nœud i ne peuvent avoir lieu qu'entre les dates a_i et b_i.
Pour les besoins de la modélisation, on suppose que :

$$\forall i \in V,\ t_{i,n+i} + s_i > 0$$

Un véhicule est autorisé à arriver en nœud i avant l'ouverture de sa fenêtre de temps mais dans ce cas, il devra attendre l'ouverture de cette dernière pour pouvoir effectuer tout type d'opération.
Pour chaque nœud $i \in N$, on pose l_i,la quantité de biens que l'on doit charger ou décharger. Cette quantité vérifie

— $\forall i \in \mathcal{P},\ l_i \geq 0$
— $\forall i \in \mathcal{D},\ l_i = -l_{i-n}$, *Relation entre les points de chargement et de déchargement*

La flotte de véhicules est dite "homogène", chaque véhicule possède une capacitée $c \in \mathbb{N}$

Variables de décision

On pose la variable de décision $x_{i,j,k} \in \{0, 1\}$ avec $i, j \in V$ et $k \in K$ telle que :

$$x_{i,j,k} = \begin{cases} 1 & \text{Si l'arc } (i,j) \text{ est utilisé par le véhicule } k \\ 0 & \text{sinon} \end{cases}$$

On note $S_i \in \mathbb{N}$, $\forall i \in V$ la variable qui représente la date à laquelle le véhicule commence le service au nœud i.
On note $L_i \in \mathbb{N}$, $\forall i \in V$ la variable qui représente la borne supérieure pour la quantité de bien présente

4. Notons que pour un même véhicule $k \in K$, les points de départ et d'arrivée sont confondus.

dans le véhicule après la visite du nœud i.

On déclare une seconde variable de décision $z_i \in \{0, 1\}$ telle que $\forall i \in \mathcal{P}$:

$$z_i = \begin{cases} 1 & \text{Si la requête } i \text{ se trouve dans la requestBank } \mathcal{R} \\ 0 & \text{sinon} \end{cases}$$

Fonction objectif

La fonction objectif du PDPTW peut être séparée en trois termes distincts auquels nous associons les coefficients suivants :

- α : La somme des distances parcourues par tous les véhicules
- β : La somme des temps passés [5] par chaque véhicule
- γ : Le nombre de requêtes présentes dans la requestBank $\mathcal{R}$

Ainsi, la fonction objectif tend à minimiser la somme pondérée de la distance parcourue, la durée des tournées effectuées par chaque véhicule et le nombre de requêtes non programmées(*i.e.* présentes dans la requestBank $\mathcal{R}$)

$$\min \alpha \sum_{k \in K} \sum_{(i,j) \in A} d_{i,j} x_{i,j,k} + \beta \sum_{kinK} (S_{\tau'_{k'},k} - a_{\tau_k}) + \gamma \sum_{i \in \mathcal{P}} z_i$$

Note : dans le cadre de ce stage uniquement la durée des tournées *i.e.* $\alpha = 0$, $\beta = 1$ et $\lambda = \infty$

Déclaration des contraintes

Nous allons à présent définir les contraintes associées au PDPTW.

Contrainte 1 : Nous devons nous assurer que le point de collecte est visité ou que la requête correspondante se trouve dans la requestBank $\mathcal{R}$

$$\sum_{k \in K_i} \sum_{j \in N_k} x_{i,j,k} + z_i = 1, \forall i \in \mathcal{P}$$

Contrainte 2 : Nous devons nous assurer que le point de livraison (pickup) est visité si le point de livraison a été visité. De plus, il faut que la visite de ces deux points soit réalisée par un seul et même véhicule.

$$\sum_{j \in V_k} x_{i,j,k} - \sum_{j \in V_k} x_{j,n+i,k} = 0, \forall k \in K, \forall i \in \mathcal{P}_k$$

Contrainte 3 : Chaque véhicule commence à son point de départ et y retourne à la fin de sa tournée.

$$\sum_{j \in \mathcal{P} \cup \{\tau'\}} x_{\tau_k,j,k} = 1, \forall k \in K$$

$$\sum_{i \in D \cup \{\tau\}} x_{i,\tau'_k,k} = 1, \forall k \in K$$

Contrainte 4 : En ajoutant cette contrainte à celle déclarée précédemment (3), on s'assure que les trajectoires consécutives entre τ_k et τ'_k existent et sont correctement définies :

$$\sum_{i \in V_k} x_{i,j,k} - \sum_{i \in V_k} x_{j,i,k} = 0, \forall k \in K, \forall j \in N_k$$

5. Le temps passé par un véhicule est définit par la différence entre l'heure de retour au dépôt et l'heure de départ donné à priori.

Contrainte 5 : Cette contrainte permet de s'assurer que les fenêtres de temps et que les variables $S_{i,k}$ sont respectées mais aussi de prévenir des sous tours et ceci en tout nœud composant une route

$$S_i - S_j + s_i + t_{i,j} \leq (1 - x_{i,j,k}) \times M,\ \forall k \in K,\ \forall (i,j) \text{ et } M \text{ une borne supérieure} \in A_k$$

$$a_i \leq S_i \leq b_i,\ \forall k \in K,\ \forall i \in V_k$$

Contrainte 6 : On impose le fait qu'une opération de collecte est réalisée avant celle de livraison et ceci pour chaque requête :

$$S_{i,k} \leq S_{n+i},\ \forall k \in K,\ \forall i \in \mathcal{P}_k$$

Contrainte 7 : Cette contrainte permet de s'assurer que les capacités d'un véhicule sont respectées et ceci tout au long de sa route.

$$L_i + l_j \leq L_j + M' \times (1 - x_{i,j,k}),\ \forall k \in K,\ \forall (i,j) \in A_k \text{ et } M' \text{ une borne supérieure}$$

$$L_i \leq c_k,\ \forall k \in K,\ \forall i \in V_k$$

Contrainte 8 : On rappelle les contraintes d'intégrité :

$$x_{i,j,k} \in \{0,1\},\ \forall k \in \mathcal{P},\ \forall (i,j) \in A_k \text{ et } z_i \in \{0,1\},\ \forall i \in \mathcal{P}$$

$$S_{i,k} \geq 0,\ \forall k \in K,\ \forall i \in V_k \text{ et } L_{i,k} \geq 0,\ \forall k \in K,\ \forall i \in V_k$$

2.3 Motivation et présentation de la matheuristique pour la résolution du PDPTW

2.3.1 Complexité du PDPTW

Le PDPTW est inclus dans la classe des problèmes de type NP-Difficile ; Cela peut être prouvé [6] en effectuant un réduction polynomiale et en considérant le TSP [7] comme un cas particulier qui, quant à lui, est un problème NP-Complet. Pour étayer notre propos, intéressons-nous à la classe du problème TSP. Nous pouvons tout d'abord rappeler quelques définitions des différentes classes de complexités

$\mathcal{P}$: les problèmes de décision résolvables en temps polynomial

$\mathcal{NP}$: les problèmes de décision pour lesquels les solutions peuvent êtres vérifiées en temps polynomial *i.e.* les problèmes de décision non déterministes en temps polynomial.

$\mathcal{NP}$-Difficile : Un problème de décision est NP-difficile si et seulement si il est au moins aussi difficile que n'importe quel problème de décision dans NP.
Il convient de noter qu'un problème de décision NP-difficile ne se trouve pas forcément dans NP, mais qu'il peut se retrouver dans une classe de complexité supérieure (EXP, R,...).

$\mathcal{NP}$-complet : Un problème de décision est NP-complet si et seulement si :

1. il appartient à la classe de problèmes de décision $\mathcal{NP}$
2. il est $\mathcal{NP}$-difficile

Note : tous les problèmes NP peuvent se réduire à un problème NP-complet.

Le TSP sous sa forme décisionnelle *Pour une distance D, existe-t-il un chemin plus court que D passant par toutes les villes et qui termine dans la ville de départ ?* ne peut, pour l'heure, être résolu d'algorithme en temps polynomial.
On dénote plusieurs variantes du problème du TSP proposées dans l'article de Matai, Mittal et Singh [7] :

6. Nous laisserons cette démonstration au plaisir du lecteur
7. Traveling Salesman Problem ou Problème du voyageur de commerce

le sTSP ou symmetric-TSP : : Soit $V = \{v_1, \ldots, v_n\}$ un ensemble de villes (ou nœeuds), $A = \{(r, s) : r, s \in V\}$, un ensemble d'arcs entre ces villes et $d_{r,s} = d_{s,r}$, le coût associé à un arc $(r, s) \in A$ [8]. Le problème sTSP revient à déterminer un circuit [9] d'une durée minimale qui visite chaque ville une seule fois.

le aTSP ou asymmetric-TSP : Il s'agit d'un problème assez similaire au sTSP mais dans ce cas là, au moins une des distances entre $r \in V$ et $d \in V$ peut être différent *i.e.* $d_{r,s} \neq d_{s,r}$

Le mTSP ou multi-TSP : Dans un ensemble donné de nœuds, on suppose qu'il existe m voyageurs de commerces situés initialement en un seul nœud dit de dépôt. Les autres nœuds (villes) qui doivent être visités sont des nœuds intermédiaires. Le mTSP consiste à trouver des itinéraires pour l'ensemble des voyageurs de commerce de telle sorte que chaque nœud $v_i \in V$ ne soit visité qu'une seule et unique fois. Le coût total que l'on souhaite minimiser dans ce problème peut être défini par la distance parcourue par l'ensemble des voyageurs de commerce. Il est possible de rajouter des contraintes au mTSP comme par exemple :

La multiplicité des dépôts : Dans ce cas, tous les voyageurs doivent terminer leurs tournées au dépôt d'où ils sont partis. Le nombre de voyageurs partant du dépôt doit être le même après la fin des parcours des itinéraires.

Le nombre de voyageurs : Le nombre de voyageurs peut être fixé ou adapté suivant le nombre d'itinéraires trouvés.

Des contraintes temporelles : Certains nœuds doivent être visités durant des périodes précises. Ces périodes sont appelées fenêtres de temps (on parle alors de mTSPTW)

...

Le mTSP est également lié au PDP qui consiste à trouver des itiniéraires optimaux pour satisfaire un ensemble de requêtes clients avec des contraintes de capacités pour chaque voyageur. Si les clients doivent être servis durant des périodes spécifiques (fenêtre de temps) alors nous retrouvons notre PDPTW.

On peut en conclure que le PDPTW se réduit au mTSPTW si le point de départ (dépôt) et le point d'arrivée coincident et ceci pour chaque itinéraire.

2.3.2 Motivation à l'utilisation d'une matheuristique

Comme nous venons de le montrer, le PDPTW est un problème difficile à résoudre dans la mesure où les solutions ne sont pas facilement trouvables, mais il est aisé de vérifier qu'une solution est admissible. Comme le présentent Courtois et Landa-Silva[3], les méthodes exactes [10] permettent de résoudre à l'optimalité ; ces dernières sont très performantes sur pour des instances de petites et moyennes tailles.
L'inconvénient majeur de ces méthodes exactes est que pour des instances de taille supérieure est que le temps de résolution ne soit plus acceptable.
Il semblerait donc judicieux de se tourner vers des méthodes dites heuristique [11] ; Ces méthodes sont supposées robustes [12], rapides pour des instances de grandes tailles et plus simple à développer et à maintenir en fonction des exigences du problème. Notons que ces méthodes seront performantes sur des instances de toute taille et permettront d'obtenir de bonnes solution en un temps réduit.

A titre d'exemple, l'utilisation d'une métaheuristique a permis à Li et Lim[5], en 2001, de trouver

8. Ce coût peut être la distance euclidienne entre les villes $r \in V$ et $s \in V$ si ces dernières sont données par leurs coordonnées (x_i, y_i). On parle alors de Euclidean TSP.

9. Dans un graphe orienté, on appelle circuit une suite d'arcs consécutifs (chemin) dont les deux sommets extrémités sont identiques.

10. *e.g.* la résolution du programme linéaire présenté précédemment

11. Une heuristique est une méthode de calcul qui fournit rapidement une solution réalisable, pas nécessairement optimale ou exacte, pour un problème d'optimisation difficile.

12. La robustesse d'un algorithme est son aptitude à se protéger des conditions initiales d'utilisation potentiellement non conformes aux données attendues lors de sa conception.

une solution optimale pour une instance du problème à 100 nœud. Leur métaheuristique met en place des opérateurs de déplacement, d'échange et de réagencement des nœuds dans une route. Vous trouverez ci dessous un exemple de résultats obtenus :

```
Instance name : lc101
Solution
Route 1 : 81 78 104 76 71 70 73 77 79 80
Route 2 : 57 55 54 53 56 58 60 59
Route 3 : 98 96 95 94 92 93 97 106 100 99
Route 4 : 13 17 18 19 15 16 14 12
Route 5 : 32 33 31 35 37 38 39 36 105 34
Route 6 : 90 87 86 83 82 84 85 88 89 91
Route 7 : 43 42 41 40 44 46 45 48 51 101 50 52 49 47
Route 8 : 67 65 63 62 74 72 61 64 102 68 66 69
Route 9 : 5 3 7 8 10 11 9 6 4 2 1 75
Route 10 : 20 24 25 27 29 30 28 26 23 103 22 21
```

TABLE 2.2 – Solution obtenue par Li et Lim pour l'instance du PDPTW lc10. Source :Li & Lim's PDPTW benchmark problems.

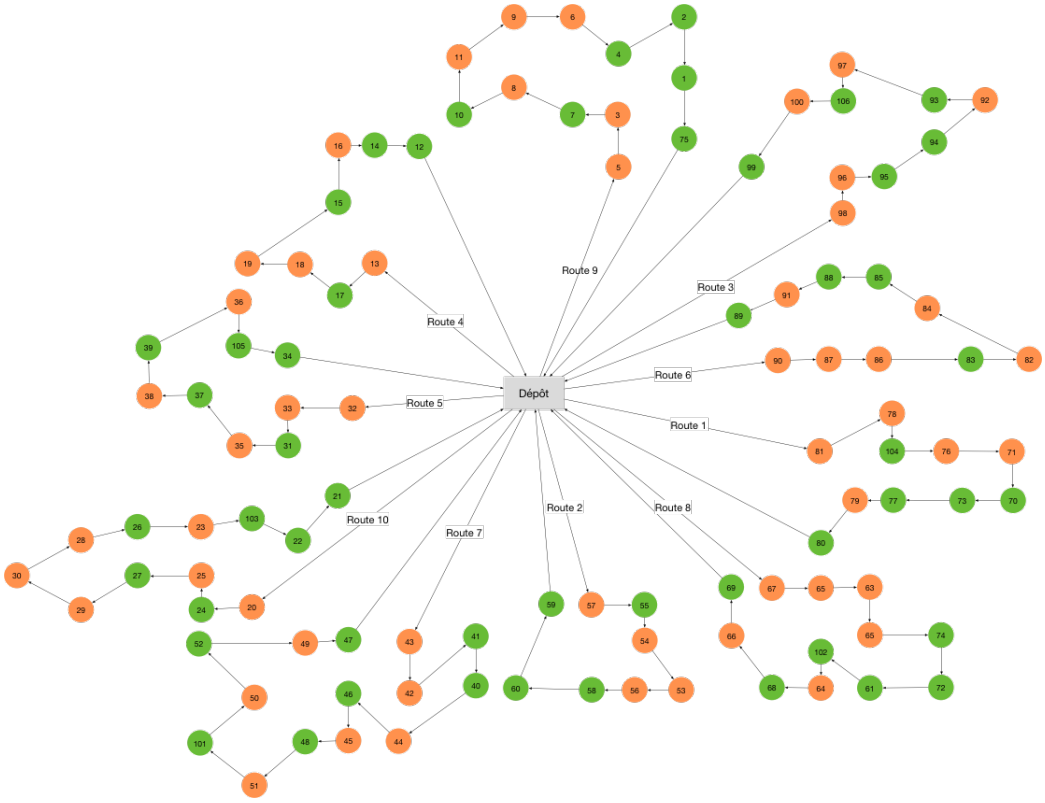

FIGURE 2.2 – Représentation graphique de la solution obtenue par Li et Lim pour l'instance du PDPTW lc101

Comme nous l'avons rappelé dans l'introduction, notre objectif est donc d'implémenter une matheuristique [13] reposant sur un algorithme de recherche à voisinage large (Large Neighborhood Search ou LNS) ainsi que sur les solutions retournées par une composante de couverture d'ensemble (Set Covering problem ou SCP).

13. Une matheuristique est un algorithme d'optimisation basée sur l'interopérabilité (capacité d'élément à opérer ensemble) entre des métaheuristiques, des algorithmes de résolution exacte et des techniques de programmation mathématiques ; On appelle métaheuristique, un algorithme d'optimisation visant à résoudre des problèmes d'optimisation difficile pour lesquels on ne connaît pas de méthode classique plus efficace. Ces méthodes sont souvent des algorithmes stochastiques, *i.e.* basés sur des processus aléatoires, et itératifs permettant de converger vers l'optimum local d'un problème.

2.3.3 Présentation de la matheuristique

La matheuristique que nous avons développée durant ce stage est présentée dans l'article de Tellez[8] écrit notamment en avec la collaboration de Monsieur Lehuédé.

La matheuristique présentée dans cet article et que nous allons implémenter par la suite est une combinaison de deux algorithmes :

— Application d'une heuristique de recherche à voisinage large (ou LNS pour Large Neighbourhood Search)

— Résolution exacte d'un problème de couverture d'ensemble (ou SCP pour Set Covering Problem)

L'approche LNS détaillée par Christiaens et Vanden Berghe[1] consiste en la création d'une solution initiale et admissible s qui va ensuite être itérativement dégradée dans une proportion choisie aléatoirement et ceci par suppression de requêtes servies par les routes de s. Dans un second temps, nous allons reconstruire une solution s' à partir du voisinage de s noté $\mathcal{N}(s)$. Si la solution obtenue par la méthode répond à un critère d'acceptation définie, alors s' devient notre nouvelle solution courante ($s \leftarrow s'$). De plus, si le coût s' est supérieur à la meilleure trouvée jusqu'à présent notée s^*, alors s^* prend la valeur de s'.

Indépendamment de la mise à jour de s et s^*, l'ensemble des routes non dominées de s' obtenue par application de la méthode *ruin and recreate* sera ajouté à un ensemble de route appelé *Pool* sur lequel sera lancé la composante de SCP ; en effet, l'objectif principal de la méthode LNS est de fournir une multitude de routes non dominées qui seront retenues ou non par la phase de résolution exacte du problème.

Après un nombre fini d'itérations de la composante LNS, nous lançons la résolution exacte du SCP sur les routes collectées dans la *Pool* et ceci sur pour une durée fixée t_{limite}. De plus, nous utilisons s^* comme solution de base permettant au solver de réduire ses temps de calculs [14]. Dans le cas où la résolution du SCP n'a pas prouvé l'optimalité, nous vidons la *Pool* et recommençons la phase de LNS avec la meilleure solution s^* obtenue jusqu'alors. Si le SCP a réussi à prouver l'optimalité, alors nous tentons de raffiner la solution retournée en supprimant les éventuelles requêtes qui seraient présentes dans plusieurs routes de la solution.

La mécanique LNS-SCP est répétée tant que le critère d'arrêt n'est pas atteint ; En pratique, ce critère peut être le nombre maximum d'itérations ou le temps de calcul maximal.

Précisions sur la composante LNS

Critère d'acceptation d'une solution

On définit le seuil d'acceptation $\alpha \in \mathbb{R}^+$ de telle sorte qu'une solution s' obtenue par après destruction et réparation de la soluton s n'est acceptée que si elle vérifie la relation suivante :

$$|s'| \leq |s^*| \text{ et } \text{coût}(s') < (1+\alpha) \times \text{coût}(s^*)$$

Avec s^*, la meilleur solution obtenue lors des itérations du LNS et le coût d'une solution correspond à la somme cumulée des distances parcourues par chacunes de routes de la solution.

Ce seuil α nous permet d'autoriser l'acceptation d'une solution s' qui serait *"moins bonne"* que la meilleure solution courante s^* et ceci à $\alpha\%$ près.

Choix de la proportion de s à détruire :

La quantité de requête à détruire est une variable aléatoire suivant une loi Uniforme continue sur l'intervalle $[\phi^-, \phi^+]$ avec $0 < \phi^- < \phi^+ < 1$.

Opérateurs de destruction et de réparation

On pose q la quantité de requête détruire. L'opération de destruction $\sigma^- \in \Sigma^-$ consiste à supprimer q requêtes choisie aléatoirement dans une solution s. Pour ce faire, on choisit dans un premier temps

14. Notion de Warmstart détaillée dans la suite du rapport

une route puis un nœud dans cette route indépendamment de son statut (pickup ou delivery) ; enfin, on supprime la requête *i.e* le couple de nœuds contenant celui désigné aléatoirement.
Chaque requête supprimée est placée dans un ensemble $\mathcal{R}$ appelé requestBank. Ainsi, pour reconstruire une solution, il suffit de parcourir l'ensemble des requêtes $r \in \mathcal{R}$ et d'évaluer les possiblités d'insertions de r dans toutes les routes $\omega \in s$ [15].
Monsieur Rochet a travaillé sur plusieurs stratégies d'insertion $\sigma^+ \in \Sigma^+$ correspondant à des parcours de $\mathcal{R}$ différents :

- Réparation ordrée : Nous évaluons l'insertion par ordre d'apparation dans $\mathcal{R}$ - Comportement FIFO [16]
- Réparation aléatoire : Nous mélangeons $\mathcal{R}$ et nous utilisons la méthode de réparation ordrée.
- Réparation par chargement : Nous trions $\mathcal{R}$ par ordre croissant de quantité pour chaque requête puis nous utilisons la méthode de réparation ordrée.
- Réparation par distance : Nous calculons la distance entre les points de pickup et de delivery, nous trions les requêtes par ordre croissant de distance et nous utilisons la méthode de réparation ordrée.

Pour chaque réinsertion d'une requête r dans une route s, nous choisissons σ^+ aléatoirement suivant une loi Uniforme discrète sur $n = |\Sigma^+|$.

Précisions sur la composante SCP

Notion de dominance entre les routes

Dans le cadre de la résolution du SCP, on définit la notion de dominance entre deux routes de la manière suivante :
Supposons ω et ω', deux routes issues de $\Omega = \{1, \dots, |\Omega|\}$ un ensemble de routes. On pose $\Pi(x)$, le coût de la route $x \in \Omega$.
On dit que **la route ω domine la route ω'** si et seulement si :

1. ω et ω' servent les mêmes requêtes (visitent les mêmes nœuds)
2. Le coût de ω est inférieur strictement à celui ω' *i.e.*

$$\Pi(\omega) \leq \Pi(\omega')$$

Problème de couverture d'ensemble :

Le Cormen[2] définit le problème de couverture d'ensemble (ou SCP) comme un problème de décision $\mathcal{NP}$-Difficile dans lequel on souhaite couvrir un ensemble fini d'élements X avec une famille $\mathcal{F}$ de sous ensembles de X de telle sorte que chaque élément de X appartienne à au moins un sous ensemble de $\mathcal{F}$:

$$X = \bigcup_{S \in \mathcal{F}} S$$

On dit alors que qu'une sous famille $S \in \mathcal{F}$ couvre les éléments de X. On associe un poids chaque sous ensemble d'éléments de X.
Le problème a pour objectif de trouver un sous ensemble $\mathcal{C} \in \mathcal{F}$ de poids minimum qui couvrant tous les éléments de X :

$$X = \bigcup_{S \in \mathcal{C}} S$$

Dans la matheuristique présentée dans ce rapport, le SCP est construit de la manière suivante :
On pose $\Omega = \{1, \dots, |\Omega|\}$ un ensemble de tournées non dominées [17] collectées pendant les précédentes itérations du LNS, c_ω $\forall \omega \in \Omega$ le coût associé à une route ω, $\mathcal{R}$, un ensemble de requêtes, K, le nombre

15. La méthode sera détaillée plus tard dans le rapport.
16. First In First Out
17. Une route ω sera dominante si elle satisfait une des conditions suivantes :
    - La route ω couvre plus de requête pour un poids équivalent ω'
    - La route ω couvre le même ensemble de requête que ω' et ceci avec un coût inférieur à celui de ω'

maximal de véhicules alloué et Ω_r, les tournées qui servent la route $r \in \mathcal{R}$. On définit la variable de décision x_ω telle que :

$$x_\omega = \begin{cases} 1 & \text{Si la route } \omega \text{ est selectionée} \\ 0 & \text{sinon} \end{cases}$$

Dès lors, le problème s'écrit de la manière suivante :

$$\min \sum_{\omega \in \Omega} c_\omega x_\omega \tag{2.1}$$

$$\sum_{\omega \in \Omega_r} x_\omega \geq 1 \qquad \forall r \in \mathcal{R} \tag{2.2}$$

$$\sum_{\omega \in \Omega} x_\omega \leq K \tag{2.3}$$

$$x_\omega \in \{0, 1\},\ \omega \in \Omega \tag{2.4}$$

Algorithme 1 : Structure du LNS-RSCP

Entrées : $\sum^-$ un ensemble d'opérateurs de destruction, $\sum^+$ un ensemble d'opérateur de réparation, η le nombre initial d'itération entre deux résolution du SCP et t_{limite} le temps limite pour la résolution du SCP

Sorties : s^* la meilleure solution trouvée

Construire une solution initiale s en utilisant $\sigma^+_{init} \in \sum^+$

`//On notera` $\mathcal{R}$`, l'ensemble des routes générées`

$\Omega \leftarrow \emptyset$ `//Une pool de route`

$\mathcal{B} \leftarrow \emptyset$ `//La requestBank`

$iter \leftarrow 0$

Tant que *Le critère de terminaison n'est pas atteint* **faire**

- `//Composante LNS`
- $s' \leftarrow s$
- **Destruction d'une quantité :**
  - $\Phi \leftarrow$ Générer aléatoirement une quantité de requêtes à supprimer telle que $\Phi \sim U([\Phi^-, \Phi^+])$
- **Sélection des opérateurs :**
  - Sélectionner aléatoirement un opérateur de destruction $\sigma^- \in \sum^-$ et un opérateur de réparation $\sigma^+ \in \sum^+$
- **Destruction :**
  - Appliquer l'opérateur σ^- pour supprimer un nombre $\max\{1, \lfloor \Phi \cdot |\mathcal{R}| \rfloor\}$ de requêtes présentes dans s' et placer les requêtes dans $\mathcal{B}$
- **Réparation :**
  - Appliquer l'opérateur σ^+ pour réinsérer les requêtes présentes dans $\mathcal{B}$ dans la solution s'
- **Si** *Le critère d'acceptation est rencontré* **alors**
  - $s \leftarrow s'$
- **fin**
- **Si** $coût(s') > coût(s)$ **alors**
  - $s^* \leftarrow s'$
- **fin**
- `//Composante SCP`
- Mettre à jour Ω avec les routes non dominées de s'
- **Si** $iter \mod \eta = 0$ **alors**
  - $s \leftarrow$ **solveSCP** $(\Omega, s^*, t_{limite})$
  - **Si** *Le SCP n'est pas résolu à l'optimalité* **alors**
    - Réinitialiser Ω
    - Après deux échecs dans la résolution à l'optimalité du SCP $\Rightarrow$ Augmenter la valeur de η
  - **fin**
  - Supprimer les requêtes en doublon dans les routes de s
  - $s^* \leftarrow s$
- **fin**
- $iter \leftarrow iter + 1$

fin

retourner s^*

3 Présentation du travail réalisé

3.1 Structures des instances et représentation UML du programme

Dans cette partie, nous vous proposons d'étudier la structure générale du programme que nous avons mis en place durant ce stage. Nous détaillerons par la suite quelques algorithmes qui ont été implémentés et ceci aussi bien dans la définition d'une solution au PDPTW que dans celle de la composante de Set Covering.

3.1.1 Structure d'une instance du problème PDPTW

Durant tout le projet, nous avons utilisé des instances de benchmark pour du PDPTW proposées par le site www.sintef.no.
Une instance est construite de la maniètre suivante :

Nb_V	Kpa_V							
N° Node	x	y	Qte	Earliest	Latest	Service time	N° Pickup	N° Delivery

TABLE 3.1 – Structure d'une instance de benchmark

Nb_V est le nombre de véhicules alloué pour cette instance, *Kpa_V*, la capacité homogène de chaque véhicules, *N° Node*, le numéro du noeud x et y les coordonnées du point dans le plan, *Qte* la quantité (***Qte*** < 0 si le noeud est de type pickup et ***Qte*** > 0 si le noeud est de type delivery), *Earliest* (resp *Latest*) le temps d'ouverture (resp de fermeture) de la fenêtre de temps et *N°Pickup* (resp *N°Delivery*), une référence au noeud qui doit être livré (resp où l'on doit collecter la quantité à livrer). Si un noeud est le point de collecte d'une requête (resp de livraison d'une requête), alors le champ delivery (resp pickup) contiendra la référence au second noeud du couple.
Ainsi, dans l'exemple ci-dessous, nous pouvons établir que la requête correspond à la collecte de 20 unités au noeud n°2 qui devront être livrées au noeud n°6 ; La collecte ne pourra avoir lieu que durant l'intervalle $[621, 702]$ et la livraison ne pourra quant à elle s'effectuer que dans l'intervalle $[825, 870]$. On notera également que chaque opérations nécessite un temps de service de 90.

```
25 200 1
0 40 50 0 0 1236 0 0 0
2 45 70 -20 825 870 90 6 0
6 40 69 20 621 702 90 0 2
```

TABLE 3.2 – Exemple d'une instance de problème.

3.1.2 Structure générale du programme

Vous trouverez sur le diagramme UML suivant, la déclaration des différentes classes du programme que nous avons développé ainsi que leurs dépendances.

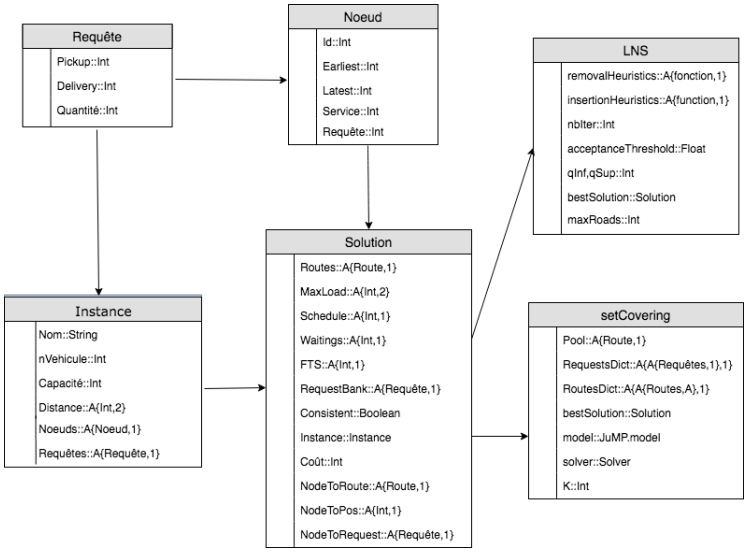

FIGURE 3.1 – Structure générale du projet où A signifie Array

3.2 Implémentations réalisées

3.2.1 Utilisation de Julia

Un des points essentiels de ce stage a été que l'ensemble des implémentations a été réalisé en *Julia*. *Julia* est un langage libre de programmation haut niveau à typage fort et dynamique (compilation en JIT [1]) en développement au MIT depuis 2009 par Jeff Bezanson, Stefan Karpinski, Viral B. Shah, Alan Edelman et disponible en version publique depuis 2012.
La philosophie de *Julia* est que *Le calcul scientifique a traditionnellement exigé les plus hautes performances, mais les experts du domaine se sont largement tournés vers des langages dynamiques plus lents pour le travail quotidien.* Ce langage se veut être performant pour tout type calcul scientifique, du prototypage à la mise en production et s'inspire des syntaxes de Matlab, R, Python et bien d'autres encore et ceci pour proposer un *langage [...] flexible, adapté à l'informatique scientifique et numérique, avec des performances comparables aux langages statiques traditionnels.*
Soutenu par une forte communauté, *Julia* est en perpétuelle évolution. Cependant, il ne rivalise pas encore avec des langages spécifiques comme par exemple dans le traitement des données statisiques où R reste une référence ; Julia reste plus compétitif dans les domaines de l'optimisation mathématique ou encore du Machine Learning.

Sources :
- Zeste de Savoir : A la découverte de Julia
- Documentation Julia

3.2.2 Implémentation de la structure de solution

Calculs des dates

Mon principal travail sur la définition de la structure d'une solution concerne les fonctions relatives aux temps et aux évaluations d'insertion de requêtes dans des routes.
Pour ce faire, j'ai utilisé les formules de calcul que Quentin Tonneau à mentionné dans sa thèse[9].
Pour la suite des descriptions des algorithmes, nous rappelons les variables définies lors de la modélisation du problème :

a_i **:** la date d'ouverture de la fenêtre de temps du noeud i

b_i **:** la date de fermeture de la fenêtre de temps du noeud i

1. Just In Time - A la volée

$t_{i,j}$: le temps de parcours entre les noeuds i et j

s_i : le temps de service du noeud i

Les opérateurs de réparation nécessitent de très régulièrement tester la faisabilité de l ?insertion de requête dans une route afin de les comparer et d'effectuer la meilleure possible *i.e.* la moins coûteuse.
La performance de ce test est donc cruciale pour garantir l'efficacité de l'approche heuristique.
Nous avons donc implémenté une série d'indicateurs pré-calculés permettant de tester en temps constant ($O(1)$) la faisabilité d'une insertion de requête et d'obtenir le coût de la solution résultante.
Ces indicateurs sont : les Forward Time Slack et les capacités résiduelles.
Nous définissons les temps d'arrivée et de service au plus tôt/tard de la manière suivante :

Le calcul de la date d'arrivée plus tôt A_i et la date de début de service au plus tôt E_i d'un véhicule en un noeud i sont donnés par les formules suivantes :

$$A_i = E_{i-1} + s_{i-1} + t_{i-1,i}$$

$$E_i = \max\{A_i, a_i\}$$

En injectant la relation E_i dans le calcul de A_i, on obtient la relation suivante pour le calcul de la date d'arrivée au plus tôt :

$$A_i = \max\{A_{i-1}, a_{i-1}\} + s_{i-1} + t_{i-1,i}$$

Le calcul de la date d'arrivée plus tard L_i et la date de début de service au plus tard D_i d'un véhicule en un noeud i sont donnés par les formules suivantes :

$$D_i = L_{i+1} - s_i - t_{i,i+1}$$

$$L_i = \min\{D_i, b_i\}$$

En injectant la relation D_i dans le calcul de L_i, on obtient la relation suivante pour le calcul de la date d'arrivée au plus tard :

$$L_i = \min\{L_{i+1} - s_i - t_{i,i+1}, b_i\}$$

On définit le temps d'attente W_i d'un véhicule en un noeud i comme la différence entre le temps de début de service au plus tôt E_i et la date d'arrivée au plus tôt A_i :

$$W_i = E_i - A_i$$

On définit le Forward Time Slack (ou FTS) comme un indicateur qui retourne la durée durant laquelle la visite d'un noeud i peut être repoussée sans occasionner de dépassement de fenêtre de temps et ceci, pour le reste de la tournée :

$$FTS_i = L_i - E_i$$

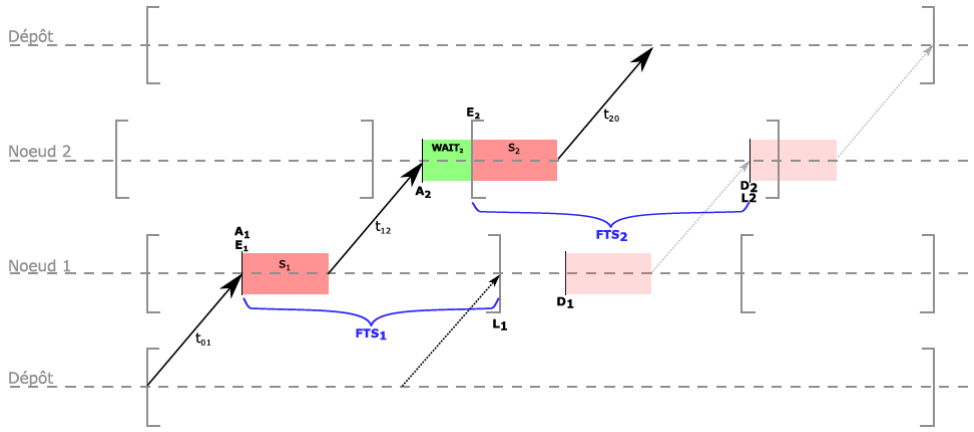

FIGURE 3.2 – Représentation graphique de variables et du FTS[9]

Insertion des requêtes dans les routes

Le processus d'évaluation de l'insertion d'une requête dans une route, composée d'un noeud de collecte et d'un noeud de livraison, comprend plusieurs étapes. Intéressons nous tout d'abord au calcul de la variation de distance induite par l'insertion d'une requête dans une route.
On pose r une requête telle que $r = (p_r, d_r)$. On souhaite ajouter p_r (resp d_r) entre les noeuds d'indices de position dans la route $(i, i+1)$ (resp $(j, j+1)$). On notera que l'évaluation d'insertion d'une requête en un point x revient à se poser la question suivante : *Est il possible d'insérer une requête après un noeud x déjà présent dans la route.*

Cas 1 : Insertion de r avec $i = j$ *i.e.* insertion de p_r en position $i+1$ et d_r en position $i+2$

Cas 2 : Insertion de r avec $i < j$ *i.e.* insertion de p_r en position $i+1$ et d_r en position $j+1$

Soient ω, une route et ω', une route obtenue à partir de ω après insertion d'une requête ; le calcul de la distance de ω' à partir de ω est donné par

$$dist(\omega') = dist(\omega) + \begin{cases} d_{i,p_r} + d_{p_r,i+1} - d_{i,i+1} + d_{j,d_r} + d_{d_r,j+1} - d_{j,j+1} & \text{si } i < j \\ d_{i,p_r} + d_{p_r,d_r} + d_{d_r,i+1} - d_{i,i+1} & \text{si } i = j \end{cases}$$

Dès lors que l'on sait simuler le coût d'une insertion dans une route à des positions données, nous pouvons alors tester la faisabilité d'une insertion.
Pour ce faire, nous allons pour une requête r et des indices de position i et j donnés, vérifier les assertions suivantes :

1. La quantité associée à la requête r peut être transportée par un véhicule entre les noeuds i et j ?
   Pour un ordonnancement des requêtes au plus tôt, nous allons simuler l'insertion de la requête.
2. L'ajout du point de collecte associé à la requête r après le noeud i n'entrainte pas de dépassement de la fenêtre de temps du noeud $i+1$?
3. La variation de temps δ induite par l'insertion du point de collecte de la requête r n'entraine pas de dépassement du FTS associé au point $i+1$?
4. L'ajout du point de livraison associé à la requête r, en considérant la variation δ, n'entraine pas de dépassement de la fenêtre de temps du noeud j ?
5. La variation de temps δ' induite par l'insertion du point de livraison de la requête r n'entraine pas de dépassement du FTS associé au point $j+1$?

Si ces cinq assertions sont vraies, alors on considère qu'il est possible d'insérer la requête r aux positions i et j avec $i \leq j$.
Note :

- Par convention, on retournera le triplet $(-1, -1, 0)$ si l'insertion de r est faisable en i et j.
- On pose p_r, le point de collecte de la requête r, d_r son point de livraison et q_r, sa quantité.
- On déclare les fonction $\textbf{charge}_\omega$(x,y) retournant la quantité disponible dans un véhicule sur la route S entre les noeuds x et y

Dès lors que nous savons évaluer la faisabilité de l'insertion d'une requête r dans une route ω à des positions données $(i, j) \in \{1, \ldots, |\omega|\}^2$ avec $i \leq j$, nous pouvons étudier la faisabilité des insertions pour une requête r sur tous les indices d'une route ω *i.e.* $\left\{\forall (i,j) \in \{1, \ldots |\omega|\}^2 | i \leq j\right\}$ à l'aide d'une simple boucle **Pour**. Nous introduisons cependant un nouveau paramètre dans ce parcours, le *blinkrate*[2] développé par Christiaens et Vanden Berghe[1] qui correspond à la probabilité de l'évèvement : *"Ne pas réaliser l'opération d'insertion d'une requête r dans une route ω"*. Dans le cas où pour une même requête r, nous trouverions plusieurs couples d'indices (i, j) possibles, alors nous choisirons celle qui présente un coût d'insertion simulé c minimum.

2. fréquence de clignement

Algorithme 2 : Algorithme de vérification de la faisabilité de l'insertion d'une requête r à des i et j positions données

Entrées : ω, une route, r, une requête, i et j, des indices de position de noeuds dans S
Sorties : p, l'indice d'insertion du pickup, d, l'indice d'insertion du delivery, c le coût après insertion

```
p ← -1,d← -1, c← 0
Si q_r ≤ charge_ω(i + 1, j) alors
    h ← E_k,∀k ∈ {1, ..., |ω|}      /* h est un vecteur contenant l'ordonancement au plus tôt du
     service pour toutes les tâches de ω */

    x_1 ← max(h[i] + s_i + t_{i,p_r}, a_{p_r})
    Si x_1 ≤ b_{p_r} alors
        x_2 ← max(a_{i+1}, x_1 + s_{p_r} + t_{p_r,i+1})
        δ ← x_2 − h[i + 1]
        Si δ ≤ FTS_{i+1} alors
            x_3 ← max(h[j] + s_j + d_{j,d_r}, a_{d_r}) + δ
            Si x_3 ≤ b_{d_r} + δ alors
                x_4 ← max(a_{j+1}, x_3 + s_{d_r} + t_{d_r,j+1})
                δ' ← x_4 − h[j + 1]
                Si δ' ≤ FTS_{j+1} alors
                    /* On obtient ω' en ajoutant r dans ω en i et j                    */
                    p ← i,d ← j, c ← |Δ (dist(ω'), dist(ω))|      /* Δ est la variation de coût entre
                     deux routes */
                fin
            fin
        fin
    fin
fin
retourner p,d,c
```

3.2.3 Implémentation du Set Covering

Nous allons à présent étudier la composante de Set Covering qui fût ma principale mission durant ce stage.
Nous rappelons que le set Covering (noté SC par la suite) est notamment composé d' :

une pool : notée P, un ensemble de routes non dominées et collectées lors des itérations de la composante de *LNS*

un dictionnaire de routes : noté *roD*, une table associative qui pour chaque requête r de l'instance du problème considéré (notée I), associe les routes ω_j dans lesquelles r est contenue, avec $j \in \{1, \dots, |P|\}$.

$$\forall i \in \{1, \dots, |I|\},\ \mathrm{r}_i = \left\{\{\omega_j\} \mid \mathrm{r}_i \subseteq \omega_j, \forall j \in \{1, \dots, |P|\}\right\}$$

un dictionnaire de requêtes : noté *reD*, une table associative qui pour chaque route ω_i présentes dans P, avec $i \in \{1, \dots, |P|\}$, associe les requêtes r_j de l'instance du problème considéré (notée I) servies par ω_i

$$\forall i \in \{1, \dots, |P|\},\ \omega_i = \left\{\{r_j\} \mid r_j \subseteq \omega_i, \forall j \in \{1, \dots, |I|\}\right\}$$

Processus d'ajout de solutions dans la pool du Set Covering

L'ajout d'une solution S dans la pool de SC nécessite de tester la dominance de chaque route de la solution S notées ω_i avec $i \in \{1, \dots, |S|\}$ sur les routes déjà contenues dans le pool de SC, notées ω'_j avec $j \in \{1, \dots, |P|\}$; Il s'agit également de mettre à jour les dictionnaires *reD* et *roD* en fonction des suppressions de routes dominées et des ajouts de certaines qui seraient dominantes. Le détail du processus est expliqué dans le pseudo-code suivant. On définit les fonctions **Ajouter**(X,y) (resp **Supprimer**(X,y))

qui ajoute (resp supprime) un élément y dans l'ensemble X et qui met à jour les dictionnaires *reD* et *roD*

Algorithme 3 : Algorithme d'ajout d'une solution S dans la pool P d'un Set Covering.

Entrées : SC, un Set Covering et P, la pool de SC
S, une solution
Sorties : SC mis à jour avec les routes dominantes de S

```
Si P est vide alors
    Pour chaque ω ∈ S faire
        Ajouter( P,ω)
    fin
Sinon
    Pour chaque ω ∈ S faire
        Pour chaque ω' ∈ P faire
            Si ω domine ω' alors
                Supprimer( P,ω')
                Ajouter( P,ω)
            Sinon
                Ajouter( P,ω)
            fin
        fin
    fin
fin
```

Modélisation du Set Covering

Pour réaliser les opérations de modélisation et de résolution du modèle de SCP définit précédemment, nous utilisons l'outil *JuMP* (Acronyme de Julia for Mathematical Optimization), un langage de modélisation mathématique intégré dans *Julia*. Ce dernier permet de modéliser une grande variété de classes de problèmes (programmation linéaire, non linéaire, en nombre entier,...), possède une syntaxe très proche des expressions mathématiques naturelles ainsi qu'une compatibilité avec une multitude de serveurs qu'ils soient libres ou commerciaux (Gurobi, Cbc, CPLEX, ...).
Le solver utilisé durant ce stage a été *Cbc* (Acronyme de Coin-or Branch and Cut), un solver open-source dédié à la programmation mixte en nombres entiers (ou MIP) [3] et développé par la fondation *COIN-OR* (Acronyme de Computational Infrastructure for Operations Research).

Pour tenter de réduire le temps d'éxécution du SCP, nous introduisons une *Warmstart*, c'est à dire, l'affectation d'une valeur à certaines variables du problème obtenues à partir de la meilleure solution admissible obtenue jusqu'alors. Cette méthode permet à l'algorithme de converger plus rapidement vers l'optimalité.

La modélisation mathématique du SCP grâce au langage *JuMP* et ceci partir du modèle définit précédemment par Tellez[8] est la suivante :

```
function generateModel(SC)
    m = Model(solver = SC.solver)
    P = SC.pool
    K = SC.K # Le nombre de véhicules autorisé pour cette instance
    nbRequest = length(SC.routesDict)
    nbRoute = length(P)
    warmStart = SC.bestSolution
    D = SC.routesDict
    ### Déclaration de variables :
    @variable(m, x[P], Bin)
    #### Fonction objectif
    @objective(m, Min, sum((P[i].cost) * x[P[i]] for i in 1:nbRoute))
```

3. Les programmes linéaires mixtes en nombres entiers sont des programmes linéaires dans lesquels certaines variables sont contraintes à prendre des valeurs entières relatives.

```
    @constraint(m, CtrReq[i in 1:nbRequest], sum(x[r] for r in D[i]) >= 1)
    @constraint(m, CtrQteK, sum(x[P[i]] for i in 1:nbRoute)<=K)
    ### Affectation des valeurs de variables de la warmStart
    for i in 1:length(warmStart.routes)
        if (findRouteIndexInPool(SC, warmStart.routes[i])!=0)
            setvalue(x[warmStart.routes[i]],1)
        end
    end
    # writeLP(m,"modèle.txt")
    return m
end
```

Processus de suppresion des requêtes présentes dans plusieurs routes

Après avoir effectué la résolution du SCP, il est possible que certaines requêtes soient présentes dans plusieurs routes selectionnées, nous devons donc simuler la suppression d'une même requête r dans ces mêmes routes, conserver celle qui présentera une variation de coût minimum et supprimer les autres routes.

Posons ω une route c_ω, le coût associé à cette route, ω', une route obtenue après suppression de la requête r et $c_{\omega'}$, son coût associé. Le coût simulé après suppression d'une requête r est donné par la relation suivante :

$$c_{\omega'} = c_\omega + \begin{cases} d_{i-1,i+1} - d_{i-1,i} - d_{i,i+1} + d_{j-1,j+1} - d_{j-1,j} - d_{j,j+1} & \text{si } (j > i+1) \wedge (j \neq |\omega| - 1) \\ d_{i-1,i-2} - d_{i-1,i} - d_{i,i+1} - d_{i+1,i+2} & \text{si } (j = i+1) \wedge (j \neq |\omega| - 1) \\ d_{i-1,i+1} - d_{i-1,i} - d_{i,i+1} + d_{j-1,1} - d_{j-1,j} - d_{j,1} & \text{si } (j > i+1) \wedge (j = |\omega| - 1) \\ d_{i-1,1} - d_{i-1,i} - d_{i,i+1} - d_{i+1,1} & \text{si } (j = i+1) \wedge (j = |\omega| - 1) \end{cases}$$

où $d_{x,y}$ est la distance entre les noeuds x et y, $(i, j) \in \{1, \ldots, |\omega|\}^2$ des indices de position dans ω.
Dans la relation ci-dessus, les cas où $j = i + 1$ correspondent à la suppression de noeuds consécutifs et les cas $j = |\omega| - 1$ à la suppression d'une requête dont le noeud de collecte se trouve juste avant le retour au dépôt.

Dès lors que nous savons simuler le retrait d'une requête r dans une route ω, il nous est possible, pour une solution retournée par le SCP, notée S, d'étudier et de supprimer au besoin les requêtes présentes dans plusieurs routes.
Pour ce faire, nous pouvons générer le dictionnaire de routes défini plus tôt dans la description du Set Covering. Grâce au parcours de ce dictionnaire, nous pouvons savoir par quelle(s) route(s) sont desservies chaque requête. Dans le cas où une requête serait servie par plusieurs routes, il faudra alors étudier les coûts simulés de suppression et conserver le minimum de ces derniers. Le pseudo code suivant va présenter dans le comportement de la méthode :

Algorithme 4 : Algorithme de suppression des requêtes doublons

Entrées : S, une solution, roD_S, le dictionaire de route de S
Sorties : S, la solution dans laquelle chaque requête r est servie par une unique route

```
1  Pour chaque r ∈ roD_S faire
2      Si |r| > 1 alors
3          V ← [ ]
4          Pour chaque ω ∈ r faire
               /* c'_ω : le coût simulé de ω après le retrait de r                    */
5              Ajouter( V, c'_ω)
6          fin
           /* On s'assure que l'on conserve la route contenant le c'_ω minimum       */
7          Supprimer( arg min(V))
8          Tant que V ≠ 0 faire
                   /* On considère que V est une pile, la méthode LIFO [a] nous permet de nous
                    assurer que l'on ne change pas les indices des routes qui précèdes dans S. De
                    même, on supprimera d'abord le point de livraison puis celui de collecte. On
                    notera le cas particulier où l'on génère une route vide, cette dernière sera
                    automatiquement supprimée */
9              Supprimer( S,ω,Dernier(V))
10             Supprimer( Dernier(V))
11         fin
12     fin
       /* mise à jour de roD_S                                                       */
13 fin
14 retourner S
```

a. Last In First Out

4 Analyse expérimentale

4.1 Présentation des résultats expérimentaux

Pour tester notre implémentation, nous avons réalisé 10 tests de 5 minutes pour chaque instances proposées et ceci avec les paramètres suivants :

— Nombre d'itérations de LNS avant de résoudre un SCP $\eta = 1000$

— Temps limite pour la résolution du SCP : $t_{\text{limite}} = 1$ minute

— Seuil d'acceptation $\alpha = 0.05$

— Intervalle de la loi uniforme continue (Destruction aléatoire de route) $\phi^- = 3$, $\phi^+ = 6$.

Les tableaux suivants représentent les résultats obtenus.

Note : On calcule la valeur d'écart par application de la formule suivante :

$$e = \frac{V_{\text{moyenne}}}{V_{\text{optimale}}} - 1 \text{ (en \%)}$$

	Optimum	**Valeurs obtenues**			**Nbiter**	**Ecart**
		Minimum	**Moyenne**	**Maximum**		
lr 101	1650,8	1868,0	1923,8	2014,2	1359	16,54%
lr 102	1487,57	1762,4	1847,6	1951,3	1503	24,20%
lr 103	1292,68	1479,6	1668,4	1841,6	1935	29,06%
lr 104	1013,39	1283,3	1536,7	1817,9	1829	51,64%
lr 105	1377,11	1642,8	1786,9	2149,1	1505	29,75%
lr 106	1252,62	1495,6	1729,4	1883,2	1864	38,06%
lr 107	1111,31	1365,0	1580,1	1913,2	1928	42,18%
lr 108	968,97	1329,8	1622,8	2297,1	1919	67,48%
lr 109	1208,96	1424,7	1542,6	1772,7	2085	27,60%
lr 110	1159,35	1241,8	1338,6	1542,6	2035	15,46%
lr 111	1108,9	1334,6	1497,6	1732,4	1932	35,05%
lr 112	1003,77	1272,5	1373,1	1775,4	2216	36,80%

TABLE 4.1 – Résultats expérimentaux pour les instances de type lr

	Optimum	Valeurs obtenues			Nbiter	Ecart
		Minimum	Moyenne	Maximum		
lc 101	828,94	1214,4	1367,2	1695,2	1672	64,94%
lc 102	828,94	1351,1	1474,1	1593,1	1708	77,83%
lc 103	1035,35	1075,0	1206,8	1533,4	2075	16,56%
lc 104	860,01	1039,8	1111,9	1217,2	2104	29,29%
lc 105	828,94	1175,6	1347,9	1534,5	1840	62,61%
lc 106	828,94	1090,8	1269,8	1618,2	2022	53,19%
lc 107	828,94	1174,9	1519,1	1736,2	1770	83,25%
lc 108	826,44	1001,7	1179,4	1664,4	1978	42,70%
lc 109	1000,6	935,2	1133,1	1601,4	2163	13,24%

TABLE 4.2 – Résultats expérimentaux pour les instances de type lc

	Optimum	Valeurs obtenues			Nbiter	Ecart
		Minimum	Moyenne	Maximum		
lrc 102	1558,07	2038,8	2168,9	2302,0	1588,4	39,21%
lrc 103	1258,74	1629,7	1740,2	2076,9	1762,3	38,25%
lrc 104	1128,4	1381,5	1745,3	2058,4	1764,0	54,67%
lrc 105	1637,62	2034,1	2173,1	2539,7	1513,5	32,70%
lrc 106	1424,73	1750,2	1949,1	2145,8	1640,9	36,80%
lrc 107	1230,14	1411,8	1661,3	1928,1	1922,8	35,05%
lrc 108	1147,43	1318,3	1407,8	1758,4	2125,3	22,70%

TABLE 4.3 – Résultats expérimentaux pour les instances de type lrc

4.2 Conclusions sur les résultats expérimentaux

L'analyse de l'écart obtenu par rapport à l'optimum proposé pour chaque instance par le Sintef nous montre que nos résultats sont loin d'être satisfaisants pour prétendre à être compétitifs pour la résolution du PDPTW.
L'étude des temps de résolutions (non présentés dans ce rapport) nous permet de déduire que l'algorithme est capable de trouver très rapidement (souvent au bout de 50 secondes) des solutions améliorantes et qu'après une résolution SCP, il n'arrive plus à trouver de meilleures solutions. Cependant, l'analyse du nombre d'itérations réalisées pour chaque instance nous indique que notre algorithme a pu, dans la majorité des cas, réaliser plusieurs résolutions du SCP, censées permettre à l'algorithme de ne pas être *"piégé"* dans les optimums locaux. Des tests sur de plus grandes durées ont été réalisées mais par faute de temps, ne peuvent être analysés dans ce rapport
Face à de tels résultats, il nous est possible de proposer plusieurs axes d'améliorations. Tout d'abord, nous devrions modifier notre protocole de test pour obtenir un historique plus précis du comportement de l'algorithme pendant les phases de résolution et ainsi pouvoir analyser son comportement face aux situations où il se retrouve *"piégé"* dans des optimums locaux. Les résultats de cette étude nous permettraient de redéfinir notre stratégie de résolution. Plusieurs idées ont été proposées pour prévenir ce type de résultats :

- L'application d'une pondération sur certains opérateurs de reconstruction qui se révèleraient *"meilleurs"* face à situations rencontrées.
- La déclaration et la mise en place de nouveaux opérateurs de destruction et de reconstruction.
- La mise en place d'une gestion dynamique de la pool de solution permettant par exemple de borner le nombre routes contenu.

— La création de nouvelles stratégies de dominance pour filtrer les routes lors de leurs insertions dans la pool.
— Une modification de la valeur η *i.e.* le nombre d'itérations de LNS à effectuer avant de lancer une phase de résolution du SCP et ceci, en fonction des résultats obtenus par ce dernier (approche proposée par Tellez[8]).

D'autres possibilités sont encore à étudier. De manière générale, si le LNS ne propose pas de routes suffisamment variées, alors Nous pouvons être sûrs que le SCP ne pourra pas retourner de solutions optimales et/ou faisables.

5 Conclusion générale

Ce stage, que j'ai souhaité réaliser dans le domaine de la recherche académique, a été pour moi une très bonne expérience. Le fait de travailler sur un sujet qui m'intéressait et faisait écho à mes études antérieures a été une source de grande motivation.
Les échanges réguliers, aussi bien formels qu'informels, entretenus avec les membres du laboratoire m'ont permis de me rendre compte des vrais enjeux de la recherche fondamentale et appliquée et de me conforter dans mes objectifs concernant ma poursuite d'études.
Réaliser des états de l'art, réfléchir à la conception et à l'implémentation de méthodes de résolution, se remettre en cause et recommencer, autant d'étapes inhérentes au milieu de la recherche qui se sont révélées très formatrices.
J'ai cependant rencontré des difficultés dans le travail de groupe qui m'ont amené à remettre en question mon approche dans ce type de situation.
Je sors grandi de cette expérience et je tiens encore une fois à remercier Messieurs Lehuédé et Tonneau de m'avoir permis de prendre part à ce projet.

Bibliographie

[1] Jan Christiaens and Greet Vanden Berghe. A fresh ruin recreate implementation for the capacitated vehicle routing problem, 2016.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Third Edition. The MIT Press, 3rd edition, 2009.

[3] Timothy Curtois, Dario Landa-Silva, Yi Qu, and Wasakorn Laesanklang. Large neighbourhood search with adaptive guided ejection search for the pickup and delivery problem with time windows. EURO Journal on Transportation and Logistics, Jan 2018.

[4] G. B. Dantzig and J. H. Ramser. The truck dispatching problem. Manage. Sci., 6(1) :80–91, October 1959.

[5] H. Li and A. Lim. A metaheuristic for the pickup and delivery problem with time windows. Proceedings 13th IEEE International Conference on Tools with Artificial Intelligence, pages 160–167., 2001.

[6] Stefan Ropke and David Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. Transportation Science, 40(4) :455–472, November 2006.

[7] Surya Singh, Murari Lal Mittal, and Rajesh Matai. Traveling salesman problem : an overview of applications, formulations, and solution approaches. In Donald Davendra, editor, Traveling Salesman Problem, chapter 1. InTech, Rijeka, 2010.

[8] Oscar Tellez, Samuel Vercraene, Fabien Lehuédé, Olivier Péton, and Thibaud Monteiro. The fleet size and mix dial-a-ride problem with reconfigurable vehicle capacity. Research report, Laboratoire DISP, September 2017.

[9] Quentin Tonneau. Optimisation de la chaine logistique des déchets non dangereux. PhD thesis, École doctorale Mathématiques et sciences et technologies de l'information et de la communication (Rennes), 2017. Thèse de doctorat Informatique et applications Ecole nationale supérieure Mines-Télécom Atlantique Bretagne Pays de la Loire 2017.