UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Optimisation de la planification des tournées de véhicules électriques

THÈSE

présentée et soutenue publiquement le 7 décembre 2020

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Tayeb OULAD KOUIDER

Composition du jury

Présidente :	Caroline PRODHON
Rapporteurs :	Jean-Charles BILAUT Marie-Ange MANIER
Examinateurs :	Isabelle CHRISMENT Dominique FEILLET Ammar OULAMARA Wahiba RAMDANE CHERIF-KHETTAF

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Remerciements

Il me sera très difficile de remercier tout le monde car c'est grâce à l'aide de nombreuses personnes que j'ai pu mener cette thèse à son terme.

Je voudrais tout d'abord remercier grandement mon directeur de thèse, Ammar OULAMARA, et ma co-directrice, Wahiba RAMDANE CHERIF-KHETTAF pour toute leur aide.

Je suis ravi d'avoir travaillé en leurs compagnies car outre leurs appui scientifique, ils ont toujours été là pour me soutenir et me conseiller au cours de l'élaboration de cette thèse.

Madame Marie-Ange MANIER et Monsieur Jean-Charles BILLAUT m'ont fait l'honneur d'être rapporteurs de ma thèse, ils ont pris le temps de lire et de m'envoyer leurs commentaires pertinents. Leurs remarques m'ont permis d'améliorer considérablement mon travail. Pour tout cela je les remercie. J'exprime ma gratitude à Madame Caroline PRODHON, à Madame Isabelle CHRISMENT, et à Monsieur Dominique FEILLET, qui ont bien voulu être examinateurs.

Je remercie toutes les personnes avec qui j'ai partagé mes études et notamment ces années de thèse.

Je dédie cette thèse
à la mémoire de Miyou
à la mémoire de mes grands parents
à mes proches.

Sommaire

Chapitre 1

Introduction : Contexte et motivation

Les problématiques autour des véhicules électriques ont connu récemment un intérêt croissant de la communauté scientifique influencé par la transition nécessaire, d'un point de vue économique et écologique, des entreprises vers des véhicules non polluants. En 2009, le gouvernement français par ses deux ministères de l'écologie et de l'industrie présente un plan pour le développement des véhicules électriques [100], une des actions de ce plan consistait en la planification d'achat de 100 000 véhicules électriques, dont 50 000 déjà prévu pour les collectivités territoriales et des entreprises telles que La Poste (10 000) et EDF (50 000). Une loi relative à la transition énergétique pour la croissance verte fut promulguée le 17 août 2015, dont une partie concerne le développement des transports propres [64]. L'article 37 de cette loi oblige les établissements publics et les entreprises nationales, entre autres, à intégrer un quota minimum de véhicules propres lors des prochains renouvellements de leurs flottes automobiles. L'état et ses établissements publics ayant un quota de 50%, et les entreprises nationales et les collectivités territoriales ayant un quota de 20%. Un projet de loi a proposé d'étendre ces quotas à toutes les entreprises ayant un parc automobile de 100 véhicules ou plus. Le quota pour ces entreprises serait progressif et commencera par 10% en 2022 pour finir à 50% en 2030 [7]. Cette loi fut adoptée le 18 juin 2018, et porte parmi ses objectifs l'interdiction complète de vente de véhicules à énergies fossiles en 2040. D'un autre côte, le gouvernement a financé l'installation de plus de 20 000 bornes de recharges sur le territoire français pour faciliter l'utilisation des véhicules électriques.

Grâce aux progrès technologiques, notamment la capacité des batteries, le véhicule électrique devient un outil prometteur pour relever le défi de la décarbonisation des activités de transport. Des services utilisant des véhicules électriques sont déjà déployés pour répondre à la demande de mobilité à travers plusieurs villes, notamment pour les déplacements quotidiens comme Autolib à Paris. Cependant, certains facteurs empêchent l'utilisation massive des véhicules dans toutes les activités de transport. Ces facteurs sont principalement limités à l'autonomie des véhicules électriques, à la longue durée de charge des batteries des véhicules électriques et à la disponibilité d'une infrastructure de charge. Cependant, pour assurer un déploiement réussi des véhicules électriques à court terme, il est important de cibler le développement vers (i) les catégories d'utilisation spécifiques où le véhicule électrique est le plus approprié en termes d'autonomie, de capacité de charge et de coût d'exploitation, et (ii) gérer le fonctionnement d'un environnement complexe d'écosystèmes de véhicules électriques (véhicules - recharges - réseau électrique - gestion de flottes automobiles) avec un accent sur les nouveaux défis en termes d'optimisation, dans l'optique de développer des modèles et outils de décision efficaces de gestion des écosystèmes des véhicules électriques.

En effet, pour planifier leurs activités, plusieurs gestionnaires de flotte utilisent des outils

logiciels de décision tant au niveau tactique (dimensionnement de la flotte, etc.) qu'opérationnel (optimisation des trajets, suivi de la performance, suivi des véhicules, etc.). Cependant, les logiciels de planification d'itinéraires existants ne sont pas adaptés à la planification du routage des véhicules électriques, car ils ne tiennent pas compte des spécificités des véhicules électriques (autonomie, temps de charge, taux de charge des batteries, type et disponibilité des stations de charge, capacité du réseau électrique, coût de l'énergie, etc. Cette mise à niveau nécessite le développement de nouveaux modèles d'optimisation et d'outils d'aide à la décision efficaces pour gérer l'ensemble de l'écosystème des véhicules électriques.

Notre travail vise donc à considérer les spécificités des véhicules électriques dans le cadre d'un problème périodique de tournées de véhicules. Nous devons donc faire face à différentes décisions :

- **l'affectation des clients sur l'horizon :** Quels jours sera visité chaque client ? Il faut donc choisir une combinaison de jours pour chaque client en respectant ses obligations (fréquence, scénarios).
- **L'affectation journalière des clients :** Si un client doit être visité un jour d, par quel véhicule sera-t-il visité ?
- **La construction d'une tournée :** Pour un ensemble de clients, est-il possible, en termes de temps et de capacité de chargement, de les visiter par un seul véhicule ? et dans quel ordre ?
- **L'aspect électrique :** Pour un ensemble de clients, un véhicule électrique a-t-il rechargé assez sa batterie pour tous les visiter ? Si non, peut-il les visiter en rechargeant pendant la tournée aux stations de recharge ? Si oui, lesquelles ?

Chacune de ces décisions est un problème combinatoire en soi. De plus, ces sous-problèmes ne sont pas indépendants et chaque décision prise dans l'un affecte directement les autres.

Dans cette thèse, nous nous focalisons sur l'étude des problèmes de décision et d'optimisation pour la planification des tournées de véhicules électriques. Ces travaux ont été effectués dans le cadre du projet ANR EVERS [3].

Le manuscrit est organisé de la manière suivante :

Dans le deuxième chapitre, nous faisons un état de l'art autour de notre problématique. Dans un premier temps, nous nous intéressons aux problèmes de tournées de véhicules n'impliquant que des véhicules thermiques. Nous commençons par donner une définition "classique" du VRP, puis de certaines de ses variantes dans ce cadre. Nous mettrons l'accent sur la variante se rapprochant le plus de notre problématique, le problème périodique de tournées de véhicules. Une fois les définitions du VRP et de ses diverses variantes présentées, nous ferons état de diverses approches de résolution pour ces problématiques. Dans un deuxième lieu, nous nous intéresserons aux problèmes de tournées de véhicules impliquant des véhicules électriques. Divers travaux sur ce sujet seront considérés, en essayant de décrire au mieux les caractéristiques du problème considéré et les méthodes de résolutions proposées. Le troisième chapitre porte sur la première problématique étudiée, problématique traitant de la planification des tournées de véhicules électriques dans le cadre d'une visite unique à chaque client sur l'horizon et où la flotte de véhicule est limitée. Ce modèle est nommé $PEVRP$. Nous commencerons par définir le problème et le modéliser. Puis, nous proposerons et détaillerons trois heuristiques de résolution. Enfin, des expérimentations seront présentées pour évaluer et comparer les trois heuristiques.

La même problématique est considérée dans le quatrième chapitre, mais une résolution par métaheuristique à grand voisinage est proposée. Il s'agit des méthodes LNS et $ALNS$ qui sont détaillées puis évaluées sur une base d'instances.

Dans le cinquième chapitre, nous considérons une problématique de planification des tournées de véhicules électriques où les clients nécessitent plusieurs visites sur l'horizon et où la flotte est

limitée. Cette version est nommée $m-GEVRP$. Une définition détaillée et une modélisation sont présentées. Nous avons proposé de nouvelles heuristiques et avons étendu les approches LNS et $ALNS$ pour traiter les nouvelles caractéristiques de notre problème.

Le sixième chapitre généralise le $m-GEVRP$ au cas où le nombre de véhicules électriques est illimitée. Pour cette problématique nommée $GPEVRP$, nous avons proposé une résolution par métaheuristique à population. Nous avons présenté une modélisation et une résolution permettant de traiter simultanément la planification des clients (leur affectation aux jours), et la définition des tournées journalières (séquence de clients par jour). Contrairement aux approches à base de LNS qui utilisent une modélisation sous forme de listes de tournées, nous avons adopté une représentation en un tour géant par jour sans délimiteurs de tournées. Quatre opérateurs de croisement et quatre opérateurs de mutation ont été proposés et évalués sur les nouvelles instances générées dans cette thèse.

Chapitre 2

État de l'art

Sommaire

2.1 Introduction

Les problèmes de tournées de véhicules sont largement considérés dans la littérature pour leurs intérêts applicatifs. Par ailleurs, des publications sont régulièrement proposées que ce soit sur la problématique initiale ou sur des variantes du problème. Dans ce chapitre nous présentons un rapide état de l'art sur les problèmes de tournées de véhicules et sur les approches de résolution que nous utilisons dans cette thèse. Nous nous contentons donc de présenter les principes et pour le détail nous renvoyons le lecteur à des publications spécialisées.

Ce chapitre s'organise en quatre sections, la première rappelant la définition du problème de tournées et de ses variantes pour une flotte de véhicules thermiques. La deuxième section fait une énumération non exhaustive des approches de résolution présentes dans la littérature. Dans la troisième section nous nous intéressons à la définition du problème de tournées de véhicules électriques dans la littérature. La dernière section présente un sommaire des travaux sur les véhicules électriques.

2.2 Le problème de tournées de véhicules : utilisations des véhicules thermiques

Le problème de tournées de véhicules (VRP) est inspiré par les besoins de l'industrie, industrie qui utilise principalement, si ce n'est uniquement, les véhicules thermiques dans les flottes d'entreprises. La définition du VRP est donc basée sur ces véhicules et il est par consensus le type de véhicule utilisé si aucune précision n'est apportée. C'est pour cela que nous ne considérons dans cette section que les véhicules thermiques pour définir le VRP et ses diverses variantes

2.2.1 VRP : Définitions et notations

Le problème de tournées de véhicules (VRP) consiste en la recherche des meilleures itinéraires pour une flotte de véhicules partant d'un dépôt afin de visiter un ensemble de clients. L'évaluation de ces itinéraires se fait selon une fonction objectif prédéfinie, telle que le temps de parcours, une distance ou un coût global à minimiser. Ce problème est une extension classique du problème de voyageur de commerce, et tout comme ce dernier, il appartient à la classe des problèmes d'optimisation NP-Complets. La première formulation connue du VRP est celle de Dantzig et Ramser[25] sous le nom de "Truck Dispatching Problem" et depuis il est un problème largement étudié dans la littérature. Le VRP est modélisé par un graphe $G = (V, A)$ / $V = C \cup \{0\}$ où $\{0\}$ est le dépôt, le point de départ de toutes les tournées. C est l'ensemble des clients à visiter et A l'ensemble des arcs reliant les clients. Le poids de chaque arc (i, j) de l'ensemble A correspond au coût (distance, temps ..) engendré par le véhicule en traversant cet arc. Les véhicules doivent commencer et finir leurs tournées au dépôt. L'objectif est donc de visiter chaque client exactement une fois tout en vérifiant les contraintes qui lui sont propres telles qu'une demande à satisfaire (livraison de marchandises) ou un service à effectuer (temps nécessaire). Un ensemble de visites faites par un véhicule est une tournée. Chaque tournée doit respecter les contraintes suivantes :

- Un client ne peut être servi que par un et un seul véhicule
- Chaque véhicule effectue une seule tournée
- Tous les clients doivent être desservis
- Capacité et/ou temps maximums à respecter et/ou nombre de véhicules à satisfaire.

Cette dernière contrainte permet de distinguer le TSP (problème du voyageur de commerce) du VRP et est différente selon les auteurs. Ainsi la définition du sigle VRP lui même est source de confusion [85].

VRP de base

Au delà de sa définition globale, le VRP possède plusieurs déclinaisons inspirées des différentes problématiques réelles. Nous définissons dans ce qui suit le cas classique du VRP puis présentons dans la section suivante brièvement d'autres variantes.

Il est commun dans la littérature de nommer "VRP" le cas du problème de tournées de véhicules avec contraintes de capacité ou Capacitated VRP (CVRP) qui modélise la problématique de livraison de marchandises. Chaque client ayant une demande qui est la quantité de marchandise à lui livrer. Les véhicules peuvent transporter jusqu'à une quantité maximale de marchandises. En plus des autres contraintes, il est nécessaire de vérifier qu'un véhicule puisse respecter les demandes de tous les clients sur sa tournée et donc que la somme des demandes sur sa tournée ne dépasse pas sa capacité.

Nous présentons dans ce qui suit deux des modélisations les plus connues et utilisées pour le CVRP. Les deux modèles sont présentés dans le cas d'un graphe orienté (défini ci-dessous)

mais peuvent facilement être adaptés au cas d'un graphe non orienté. Avant de présenter cette modélisation il est nécessaire d'introduire quelques notations de base qui seront utiles pour la suite :

Notations

- **Graphe orienté** un graphe orienté $G = (V, A)$ est un couple formé de V un ensemble de sommets et A un ensemble d'*arcs*, chaque arc étant associé à un couple de sommets et une direction. Un arc (i, j) est donc différent de l'arc (j, i), au delà de la direction inversée entre les deux arcs, les propriétés (distance, temps, coût) des deux arcs sont différentes.
- **Graphe non orienté** si le graphe $G = (V, A)$ est non orienté, A est un ensemble d'*arêtes* dont la direction est non importante. On représente l'arête entre i et j par $\{i, j\}$.
- **Une coupe** Soit $S \in C$ un sous-ensemble de sommets, la coupe selon S pour un graphe non orienté est l'ensemble des arêtes ayant uniquement une des deux extrémités est dans S ce qui équivaut à $\delta(S) = \{\{i, j\} : i \in S, j \notin S\}$. Pour le graphe orienté on identifie deux ensembles : les arcs sortants $\delta^+(S) = \{(i, j) : i \in S, j \notin S\}$ et les arcs entrants $\delta^-(S) = \{(i, j) : i \notin S, j \in S\}$. La coupe selon S est $\delta^+(S)$ alors que $\delta^-(S)$ est la coupe selon V/S.
- **Nombre minimum de véhicules** Soit $S \in C$ un sous-ensemble de clients et $r(S)$ le nombre minimum de véhicules nécessaires pour effectuer des tournées réalisables en terme de demandes des clients et de capacité du véhicule. Ce nombre peut être obtenu en résolvant un problème de bin packing [62]. Aussi, la demande totale des clients divisée par la capacité d'un véhicule est une borne inférieure. Nous utilisons cette borne pour la deuxième modélisation.

Modélisation du VRP : modèle à trois indices La première modélisation que nous présentons est celle de flot à trois indices (three-index vehicle flow formulation). Reprenons la notation précédente avec le graphe orienté $G = (C \cup \{0\}, A)$. $\{0\}$ est le dépôt où un ensemble de m véhicules $V = \{1, ..., m\}$ est disponible, chaque véhicule ayant une capacité maximale Q. Chaque client $i \in C$ a une demande $0 \leq q_i \leq Q$. Le poids c_{ij} de chaque arc $(i, j) \in A$ correspond à son coût. Le modèle est comme suit :

$$\text{La variable de décision } x_{ijk} = \begin{cases} 1 \text{ Si le véhicule } k \text{ parcourt l'arc } (i, j) \\ 0 \text{ Sinon} \end{cases}$$

La fonction objectif étant de minimiser le coût total engendré par les routes, le programme linéaire s'écrit comme suit :

$$\min_{x_{ijk}} \quad \sum_{(i,j)\in A} c_{ij} \sum_{k=1}^{m} x_{ijk} \tag{2.1a}$$

$$\text{s.t.} \quad \sum_{i=0}^{n}\sum_{k=1}^{m} x_{ijk} = 1 \qquad \forall j \in C \tag{2.1b}$$

$$\sum_{i=0}^{n} x_{ipk} = \sum_{j=0}^{n} x_{pjk} \qquad \forall p \in C, \forall k \in V \tag{2.1c}$$

$$\sum_{j=1}^{n} x_{0jk} \leq 1 \qquad \forall k \in V \tag{2.1d}$$

$$\sum_{j=1}^{n} x_{0jk} = \sum_{i=1}^{n} x_{i0k} \qquad \forall k \in V \tag{2.1e}$$

$$\sum_{i=1}^{n} q_i \times \sum_{j=0}^{n} x_{ijk} \leq Q \qquad \forall k \in V \tag{2.1f}$$

Les contraintes 2.1b imposent que chaque client est visité exactement une fois. Les contraintes de conservation de flot 2.1c imposent que le même véhicule qui a visité un client, "reparte" de chez ce client. Les contraintes 2.1d et 2.1e assurent que si un véhicule fait une tournée, elle sera unique et commencera et finira au dépôt. Les contraintes 2.1f assurent que le véhicule a assez de capacité pour satisfaire la demande de chacun des clients sur sa tournée.

Ce modèle nécessite donc $m \times n^2$ variables binaires et $n + nm + 3m$ contraintes.

Modélisation du VRP : modèle à deux indices La deuxième modélisation se base sur une variable à deux indices, ce qui réduit le nombre de variables mais ne permet pas d'avoir l'information exacte du véhicule qui fait la tournée. La variable de décision est donc :

$$\text{La variable de décision } x_{ij} = \begin{cases} 1 \text{ Si l'arc } (i,j) \text{ est emprunté lors d'une tournée} \\ 0 \text{ Sinon} \end{cases}$$

La fonction objectif est toujours de minimiser le coût total engendré par les tournées, le programme linéaire s'écrit comme suit :

$$\min_{x_{ij}} \quad \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{2.2a}$$

$$\text{s.t.} \quad \sum_{i=0}^{n} x_{ij} = 1 \qquad \forall j \in C \tag{2.2b}$$

$$\sum_{i=0}^{n} x_{ip} = \sum_{j=0}^{n} x_{pj} \qquad \forall p \in C, \tag{2.2c}$$

$$\sum_{j=1}^{n} x_{0j} \leq m \tag{2.2d}$$

$$\sum_{j=1}^{n} x_{0j} = \sum_{i=1}^{n} x_{i0} \tag{2.2e}$$

$$\sum_{(i,j)\in\delta^+(S)} x_{ij} \geq r(S) \qquad \forall S \subset C \tag{2.2f}$$

Les contraintes 2.2b vérifient que chaque client est visité. Les contraintes 2.2c sont celles de conservation de flot. Les contraintes 2.2d et 2.2e permettent de vérifier qu'au maximum m tournées sont construites et que chacune d'elle commence et finit au dépôt. Les contraintes 2.2f inspirées des contraintes d'élimination de sous-tours du problème de voyageur de commerce, imposent que les seules tournées construites incluent le dépôt et aussi que la somme des demandes d'une tournée reste inférieure à la capacité d'un véhicule.

Ce deuxième modèle nécessite donc n^2 variables binaires et $2n + 2^n$ contraintes. Le gain de variables obtenu par la deuxième modélisation se répercute sur le nombre de contraintes, spécifiquement les contraintes 2.2f dont le nombre est exponentiel relativement au nombre de clients. En plus de donner plus de précisions sur l'affectation des tournées aux véhicules, le premier modèle permet plus facilement de considérer un problème où chaque véhicule a une capacité propre en considérant un vecteur Q indexé sur les véhicules.

2.2.2 Variantes du problème VRP

De multiples applications du VRP ont généré diverses variantes de ce problème et la littérature sur ces variantes est immense. Dans cette section nous présentons premièrement quelques variantes et dans un deuxième temps nous détaillerons plus particulièrement la variante périodique du VRP qui est la variante du VRP qui se rapproche le plus de notre problématique.

Problèmes de tournées de véhicules avec fenêtres de temps (VRPTW) : Dans cette variante le client définit une fenêtre de temps (intervalle) dans laquelle il peut être visité par le véhicule. Si le véhicule arrive en dehors de la fenêtre de temps, il existe deux formulations : La première dite à fenêtres de temps molles (soft time windows constraints), où le client est servi mais une pénalité est ajoutée à la fonction objectif [32][13]. La deuxième formulation dite à fenêtres de temps dures (hard time window constraints), où le véhicule attend le début de l'intervalle s'il arrive plus tôt, et ne pourra pas servir le client s'il arrive plus tard [98][5][60]. Un des premiers travaux relatifs au VRPTW [83] date de 1967, les auteurs traitent un cas pratique d'un VRP avec fenêtres de temps sans le nommer ou le modéliser. Dans [98], Solomon propose des instances du VRPTW à 100 clients qui sont devenues des benchmarks reconnus pour la problématique. L'une des variantes de ce problème est celle où chaque client a plusieurs fenêtres de temps et nécessite une seule visite lors de l'une d'elles [52].

Le problème de tournées de véhicules multi-dépôts (MDVRP) : considère plusieurs dépôts au lieu d'un, dans chacun d'entre eux est affectée une partie de la flotte de véhicules. Chaque client est caractérisé par un sous-ensemble de dépôts, il ne doit être visité qu'une fois à partir de l'un d'eux. Les véhicules doivent commencer et finir leur tournée au dépôt où ils sont affectés et ne peuvent pas visiter les autres dépôts [21][11]. Une variante permet aux véhicules de visiter d'autres dépôts pour réapproviser de la marchandise si nécessaire [23],[99].

Le problème de tournées de véhicules avec livraison et collecte (VRPB) : Les clients peuvent demander une livraison de marchandises ou une collecte de marchandises. On retrouve donc deux types de clients, des clients nécessitant une livraison de marchandises depuis le dépôt et ceux nécessitant une collecte de marchandises vers le dépôt. Chaque véhicule partant du dépôt doit premièrement livrer tous les clients de sa tournée puis collecter les marchandises des clients pour les ramener au dépôt [56].

Le problème de tournées de véhicules avec ramassage et livraison (VRPSD) : Similaire à la variante précédente, on retrouve aussi deux types de clients, sauf que la livraison/collecte se fait entre les clients, le(s) dépôt(s) n'étant qu'un point de départ et de fin pour les véhicules. Les clients sont donc organisés en paires (collecte, livraison) et l'objectif du "VRP with Pickup and Delivery" est de livrer chaque client "livraison" avec les produits récupérés chez le client "collecte" correspondant [27]. Il existe diverses variantes à cette problématique, telles que celle ou on considère que le produit est le même chez tous les clients, il n'est donc pas obligatoire de respecter les paires (collecte, livraison), cette variante et d'autres telles que le transport de personnes (Dial-A-Ride Problem) sont présentées et expliquées dans le survey en deux parties [72][73].

Le problème de tournées de véhicules Multi-trips (VRPMT) : Cette variante permet aux véhicules de repasser par le dépôt pour se ravitailler en marchandises, le véhicule peut donc visiter plus de clients en une journée que ne lui permet sa capacité lors d'une tournée simple. Elle permet donc d'utiliser moins de véhicules mais elle n'est adaptée que dans le cas où il est relativement facile en termes de distance et de temps de repasser par le dépôt [14].

Le problème de tournées de service (TRP) : A l'opposé des livraisons de marchandises, les problématiques de services comme les tournées de soins médicaux ou les services à domicile (dépannage, réparation...), sont modélisés tels que chaque client nécessite un service pouvant être réalisé par la totalité ou un sous-ensemble des techniciens (véhicules) et dont la durée est fixe ou dépendante du technicien affecté. Les techniciens n'ont pas les mêmes compétences ou/et à des niveaux différents. Dans [78] les auteurs présentent le Technician Routing and Scheduling Problem, où chaque technicien a un ensemble de compétences et peut prendre des ressources renouvelables (outils) et non renouvelables (pièce de rechanges) au dépôt, chaque client a une tâche nécessitant une compétence spécifique. L'objectif étant que chaque technicien puisse effectuer les tâches des clients de sa tournée. Dans [6] les auteurs considèrent qu'un véhicule transporte une équipe puisque les tâches des clients peuvent nécessiter plusieurs techniciens, la solution est donc de constituer les équipes et les tournées satisfaisant l'ensemble des clients. Dans [15], les techniciens ont les mêmes compétences mais à différents niveaux d'expériences, les clients ont plusieurs tâches dans le cadre d'une entreprise de services à domicile, la durée des tâches dépend de l'expérience du technicien sur chacune. Ces problématiques sont variées, donc difficiles à englober dans une seule modélisation. De plus elles sont plus proches des problématiques d'affectation ce qui explique le peu de littérature dans le domaine du VRP et du routing en général.

La littérature sur les problématiques de tournées regorge d'autres variantes du VRP que nous ne pouvons pas citer ici vu leurs nombre et leurs spécificité plus ou moins complexes. En plus de ces diverses variantes, il est très commun de trouver des travaux sur des problématiques mixant plusieurs variantes telles que le Rich VRP dans [2] qui combine le Multi-depot VRP

avec le VRP with Time Windows et y rajoute différentes autres contraintes et caractéristiques. Plusieurs surveys ont été élaborés au fil des années pour synthétiser les différents travaux sur les problèmes de tournées, nous citerons ici trois livres surveys ([103], [47] et [104]) traitant du VRP et de ses diverses variantes.

2.2.3 Le problème périodique des tournées de véhicules

Dans cette thèse, nous nous intéressons au "periodic VRP" où "PVRP" qu'on peut traduire par le problème périodique des tournées de véhicules. Dans cette variante, on considère la construction de tournées des véhicules sur un horizon de temps plus large qu'un seul jour. Il est donc nécessaire de décider du/des jour(s) de visite de chaque client en prenant en compte ses besoins et contraintes tels que les jours de disponibilité ou/et le nombre de visites. Le premier cas réel dans la littérature est étudié dans [10] où la problématique de collecte de déchets sur une semaine est posée et une résolution en heuristiques sans modélisation formelle du problème est proposée.

Depuis ce premier cas d'études, plusieurs travaux se sont intéressés à ce problème et on trouve plusieurs façons de le présenter et de considérer l'impératif des visites multiples sur un horizon. Globalement, toutes intègrent directement ou indirectement la notion de fréquence de visite d'un client. La fréquence étant le nombre de fois qu'un client doit être visité sur l'horizon défini. Dans [71] l'auteur définit trois possibles formulations pour les contraintes liées au client dans un PVRP. Ces formulations peuvent exprimer séparément ou en combinaison toute variante relative à ces contraintes :

- Le client fournit un nombre minimum et maximum de jours, le délai entre deux visites doit être dans l'intervalle. Les contraintes ici sont donc d'espacement entre visites et le nombre de visites est déduit sans être fixé initialement.
- Le client communique directement la fréquence souhaitée. Il est possible qu'il complète par des contraintes d'espacement minimum et/ou maximum.
- Le client fournit des scénarios de visite et doit être visité conformément à l'un d'eux. Un scénario étant un ensemble de jours ou le client est visité. Les scénarios peuvent aussi être déduits des espacements ou/et de la fréquence donnés par le client.

La formulation admise comme étant celle du PVRP "classique" reste celle des auteurs dans [17], qui définissent le PVRP en utilisant des scénarios ("schedule" dans le texte) mais en fixant la fréquence f par client. Chaque client i exige donc un nombre f_i de visites qu'on nommera fréquence. et il doit être visité selon un scénario respectant la fréquence parmi l'ensemble des scénarios. Les auteurs présentent la première modélisation du PVRP suivant cette formulation, ce modèle en programmation linéaire en nombres entiers reste le modèle le plus communément utilisé dans la littérature pour décrire ou résoudre le PVRP. Avant de présenter ce modèle, nous notons D l'ensemble des jours de l'horizon et $\check{S}$ l'ensemble des scénarios. Chaque scénario $\check{s}$ est un vecteur de $a_{\check{s}}^d$ où :

$$a_{\check{s}}^d = \begin{cases} 1 \text{ Si une visite est prévue le jour } d \text{ dans le scénario } \check{s} \\ 0 \text{ Sinon} \end{cases}$$

L'ensemble des scénarios possibles pour un client i sera noté $\check{S}_i$ et sera défini selon sa fréquence f_i par la formule $\check{S}_i = \{\check{s} \in \check{S} : \sum_{d \in D} a_{\check{s}}^d = f_i\}$. Nous reprenons la notation utilisée précédemment, à savoir un graphe $G = (C \cup \{0\}, A)$. $\{0\}$ est le dépôt ou un ensemble de véhicules $V = \{1, ..., m\}$ est disponible, chaque véhicule ayant une capacité maximale Q. Chaque client $i \in C$ a une demande $0 \leq q_i \leq Q$ qui doit être satisfaite à chaque visite sur l'horizon. Le poids c_{ij} de chaque

arc $(i,j) \in A$ correspond au coût engendré par le véhicule s'il traverse cet arc. Les variables considérées sont :

$$x_{ijk}^{d} = \begin{cases} 1 \text{ Si l'arc } (i,j) \text{ est visité le jour } d \text{ par le véhicule } k \\ 0 \text{ Sinon} \end{cases}$$

$$y_{i}^{d} = \begin{cases} 1 \text{ Si le client } i \text{ est visité le jour } d \\ 0 \text{ Sinon} \end{cases}$$

$$z_{i}^{\check{s}} = \begin{cases} 1 \text{ Si le scénario } \check{s} \text{ est choisi pour le client } i \\ 0 \text{ Sinon} \end{cases}$$

Le modèle s'écrit comme suit :

$$\min \quad \sum_{d\in D}\sum_{(i,j)\in A}\sum_{k=0}^{m} c_{ij}x_{ijk}^{d} \tag{2.3a}$$

$$\text{s.t.} \quad \sum_{k=0}^{m}\sum_{j=1}^{n} x_{0jk}^{d} = m \tag{2.3b}$$

$$\sum_{j=0}^{n} x_{ijk}^{d} = \sum_{j=0}^{n} x_{jik}^{d} \qquad \forall i \in C, \forall k \in V; \forall d \in D \tag{2.3c}$$

$$\sum_{i,j\in S} x_{ijk}^{d} \leq |S| - 1 \qquad \forall S \subset C, k \in V, d \in D \tag{2.3d}$$

$$\sum_{i=1}^{n} q_i \times \sum_{j=0}^{n} x_{ijk}^{d} \leq Q \qquad \forall d \in D, \forall k \in V \tag{2.3e}$$

$$\sum_{\check{s}\in\check{S}_i} z_{i}^{\check{s}} = 1 \qquad \forall i \in C \tag{2.3f}$$

$$y_{i}^{d} = \sum_{\check{s}\in\check{S}_i} z_{i}^{\check{s}} a_{\check{s}}^{d} \qquad \forall d \in D, \forall i \in C \tag{2.3g}$$

$$\sum_{k=0}^{m} x_{ijk}^{d} \leq \frac{y_{i}^{d} + y_{j}^{d}}{2} \qquad \forall d \in D, \forall i,j \in C \tag{2.3h}$$

$$\sum_{k=0}^{m}\sum_{i=0}^{n} x_{ijk}^{d} = y_{j}^{d} \qquad \forall j \in C, \forall d \in D \tag{2.3i}$$

$$\sum_{j=1}^{n} x_{0jk}^{d} \leq 1 \qquad \forall k \in V, \forall d \in D \tag{2.3j}$$

Les contraintes (2.3b), (2.3c), (2.3d), et (2.3e) sont les contraintes classiques du VRP et permettent respectivement de vérifier que tous les véhicules partent du dépôt, la conservation de flots, l'élimination des sous-tours, et le respect des contraintes de capacités des véhicules. Les contraintes (2.3f) assurent qu'un scénario adéquat est choisi pour chaque client alors que les contraintes (2.3g) assurent que la variable y_i^d est bien définie. Les contraintes (2.3h) vérifient qu'un arc ne peut être parcouru un jour d que si les deux clients sont visités ce même jour d et les contraintes (2.3d) imposent qu'un client est visité chaque jour de son scénario choisi. Finalement, les contraintes (2.3j) vérifient qu'un véhicule n'est utilisé qu'une fois par jour.

Cette modélisation est utilisée dans [17] pour obtenir une approximation de la solution en relaxant le problème dû à sa grande complexité.

Dans [90] on trouve une formulation d'un cas particulier le "Assignement vehicle routing problem" où le client fournit la fréquence souhaitée et un ensemble de jours où il peut être visité. Cette variante peut être amenée au cas PVRP expliqué plus haut en déduisant toutes les combinaisons possibles pour un client à partir de sa fréquence et de l'ensemble des jours puis en les traduisant en scénarios. Une formulation en programme linéaire mixte est proposée [90].

2.3 Approches de résolution des problèmes de tournées

Le problème de tournées de véhicules étant une généralisation du problème de voyageur de commerce, sa complexité est donc a minima égale à ce dernier. De ce fait et comme tout problème NP-complet, sa résolution par des approches exactes reste trop onéreuse si ce n'est impossible dans des délais raisonnables. Néanmoins, plusieurs travaux se sont intéressés à ses résolutions et ont réussi à traiter des instances relativement conséquentes. Malgré ces efforts, les méthodes de résolution approchées restent dominantes dans la littérature grâce à leur efficacité à trouver des solutions jugées bonnes en des temps records. Nous présentons dans ce qui suit une brève et globale revue des méthodes de résolutions exactes les plus utilisées pour le VRP puis nous nous intéresserons à une sélection non exhaustive des méthodes approchées présentes dans la littérature appliquées au VRP.

2.3.1 Méthodes exactes

Procédure par Séparation et Évaluation (Branch and Bound) Cette méthode génère un arbre de recherche pour exploiter l'ensemble des solutions S. D'un côté la séparation permet de partitionner l'ensemble S pour mieux l'explorer, de l'autre côté l'évaluation permet de décider si il faut ou non explorer une partie selon l'éventualité de l'existence d'une solution optimale. Généralement la méthode part du modèle mathématique, qui sera relaxé d'un ensemble de contraintes difficiles (telles que les contraintes d'intégrité) puis résolu. Si la solution obtenue vérifie les contraintes non considérées (peu probable) la solution est optimale. Sinon, une séparation en plusieurs sous problèmes qu'on appellera noeuds enfants est faite. La séparation garantit que la résolution des noeuds enfants ne donnent pas la même solution (non réalisable) obtenue au noeud parent. La procédure d'évaluation permet de choisir le noeud à résoudre et/ou à éliminer certains noeuds qui ne peuvent contenir la solution optimale.

La première application de cette méthode est en 1986 dans [58]. La méthode part du modèle 2.2 et relaxe les contraintes (2.2f) vu leur nombre exponentiel, pour la suite on appellera ce modèle relaxé par $PL1$. La solution obtenue par la résolution de $PL1$ peut ne pas être réalisable pour le CVRP par la présence de tournées ne contenant pas le dépôt et/ou par le fait que la somme des demandes d'une tournée peut excéder la capacité d'un véhicule. Si la solution obtenue est réalisable pour le CVRP elle est forcément optimale. Sinon il existe, au moins, une tournée T de la solution obtenue qui ne respecte pas une contrainte du CVRP, cette tournée ne peut donc pas être incluse dans la solution finale. La séparation permet d'obtenir un ensemble de noeuds enfants à partir de $PL1$. Chacun d'entre eux interdira une arête de T en rajoutant une contrainte à $PL1$ de type $x_{ij} = 0$. Ces PL enfants sont résolus à leur tour et séparés si besoin. Les meilleurs résultats à notre connaissance sont ceux dans [35], où les auteurs utilisent une multitude de bornes inférieures pour mieux évaluer l'arbre de décision et ont réussi à résoudre des instances générées aléatoirement et allant jusqu'à 300 clients. La même méthode est utilisée dans [101] et permet de résoudre des instances reconnues de la littérature allant jusqu'à 47 clients.

Une autre méthode de Branch and bound fut appliquée au VRP with Backhauls dans [102], les auteurs ont relaxé le VRPB vers un problème d'arbre couvrant minimum avec dépôt à m degrés. De plus, les auteurs ont utilisé une borne inférieure lagrangienne pour accélérer le processus de résolution. Les résultats sont probants puisque des instances de 100 clients étaient résolues.

Branch and cut Cette méthode utilise le principe des coupes dans le polyèdre des solutions et la combine avec la méthode de séparation et évaluation. Partant d'un problème d'optimisation et d'un programme linéaire relaxé PL_r de ce dernier, si le PL_r donne une solution non réalisable du problème, une coupe est une contrainte qui une fois rajoutée au PL_r élimine cette solution et permet donc de mieux cerner le polyèdre de solutions. Il existe généralement un nombre exponentiel de coupes potentielles pour un problème, il n'est donc pas intéressant de les rajouter initialement au programme linéaire au risque de le rendre insolvable. La méthode du branch and Cut, applique le principe du Branch and Bound et tente de trouver et rajouter des coupes à chaque résolution d'un noeud avant de le séparer si besoin.

Les meilleurs résultats obtenus par le branch and Cut de la littérature sont obtenus par deux versions différentes de la méthode, celle dans [84] et celle dans [61] qui réussissent à résoudre des instances ayant jusqu'à 100 clients. Ces résolutions sont cependant coûteuses, puisque une des résolutions dans [84] nécessite 2 millions de secondes.

Génération de colonnes Cette méthode est utilisée pour résoudre les programmes linaires avec un nombre trop important (exponentiel) de variables. Soit $\check{m}$ le nombre de contraintes et $\check{n}$ le nombre de variables d'un PL, on sait que $\check{m} < n$ et qu'au minimum $\check{n} - \check{m}$ variables sont nulles dans la solution optimale. La méthode utilise ce principe et considère seulement un sous-ensemble de variables dans un sous-programme linéaire PL', les autres variables étant mises à zéro. En utilisant la dualité, on sait si la solution obtenue par la résolution de PL' est optimale pour le PL. Si ce n'est pas le cas une variable est rajoutée au PL' en utilisant les coûts réduits et en résolvant le programme dual. L'opération est répétée jusqu'à l'obtention d'une solution optimale.

Cette méthode est combinée à un branch and bound dans [1] et permet de résoudre des instances de 25 clients, ce qui était considérable à cette époque. Depuis, peu de travaux probants ont utilisé uniquement la génération de colonnes ou l'ont combinée seulement avec le Branch and Bound.

Branch and cut and price Cette méthode rajoute le principe de génération de colonnes au Branch and Cut. Initialement un sous-problème avec un sous-ensemble de variables est considéré puis à chaque séparation et donc à chaque noeud une variable (génération de colonnes) et une contrainte (coupe) sont rajoutées au programme linéaire. Dans la génération de colonnes, le fait de ne rajouter que des variables à chaque itération permet d'avoir la même structure du dual et donc de mieux appréhender sa résolution. Il est donc préférable de bien choisir la coupe à ajouter pour garder la même structure du dual obtenue. Ces coupes sont appelées les coupes robustes. Dans le cas de coupes non robustes il est nécessaire de prévoir la complexité croissante des problèmes duaux.

Le Branch and Cut and Price avec des coupes robustes utilisé dans [40] permet de résoudre toutes les instances classiques de la littérature ayant 135 clients ou moins.

2.3.2 Méthodes exactes pour le PEVRP

Dû à la complexité plus haute du PVRP, la majorité des travaux de la littérature se concentre sur la proposition de méthodes heuristiques et métaheuristiques ou, plus rarement, sur l'utilisation d'outils de résolution exacte (programmation linéaire, génération de colonnes...) pour obtenir des solutions "bonnes" en un temps acceptable. Par exemple, dans [36] une formulation continue obtenue à partir de la relaxation d'un programme linéaire et de l'utilisation de multiples fonctions d'approximation est utilisée pour obtenir des solutions non entières qui peuvent orienter les décisions dans un contexte stratégique. Dans [70], une formulation obtenue d'une décomposition Dantzig–Wolfe est relaxée, en terme d'intégrité des variables, puis résolue avec une méthode de génération de colonnes ; la solution, étant non entière, est arrondie avec une méthode de séparation et évaluation.

A notre connaissance, les seuls travaux obtenant des résultats pour la résolution exacte d'un PEVRP sont présentés dans [8], leur méthode généralise la formulation du CVRP proposée dans [9] au problème périodique et génère cinq types de coupes et des bornes obtenues de différentes relaxations du modèle initial. Cette méthode avec une limite de 4h de calcul permet de résoudre 14 des 28 instances de la littérature considérées.

2.3.3 Heuristiques et métaheuristiques

Nous présentons dans cette partie quelques heuristiques des plus connues appliquées au VRP, puis nous dressons un aperçu des métaheuristiques utilisées.

Heuristiques

Méthode des gains (Clark and Wright) L'heuristique de "Clarke and Wright" [19] est une des premières et des plus connues des heuristiques développées pour le CVRP et elle fut aussi adaptée pour de nombreuses variantes. L'heuristique se base sur le gain obtenu par la fusion de deux tournées distinctes, gain qui se calcule par la formule ($\Delta_{ij} = c_{i0} + c_{j0} - c_{ij}$) pour la fusion d'une tournée qui se termine par la visite du client i et une tournée qui commence par la visite du client j. L'heuristique commence par des tournées se composant d'un unique client et donc par autant de tournées que de clients. La deuxième étape consiste à calculer le Δ pour chaque paire de tournées et de fusionner les tournées dont le gain est maximum et dont la tournée résultante est réalisable. Si le nombre de véhicules est une variable de décision, la deuxième étape est répétée jusqu'à ce qu'aucune fusion ne soit possible. Sinon elle est répétée jusqu'à l'obtention d'autant de tournées que de véhicules.

Méthode d'insertion Les méthodes d'insertion construisent les tournées itérativement en insérant des clients à chaque itération selon un critère prédéfini jusqu'à obtenir une solution complète.

La plus connue est celle du **plus proche voisin**, qui à chaque itération insère le client le plus proche du dernier client visité jusqu'à atteindre le seuil de capacité du véhicule puis crée une nouvelle tournée à partir du dépôt.

Une autre méthode est celle de **la meilleure insertion**, cette méthode commence par autant de tournées que de véhicules, ces tournées sont vides puisqu'elles se composent d'une boucle autour du dépôt. A chaque itération, l'insertion des clients non visités est testée et celle engendrant le minimum de coût sans violer les contraintes est insérée.

Méthodes à deux phases Le problème du VRP peut être décomposé en deux sous-problèmes : partitionner les clients en groupes (Clustering) et décider de l'ordre de parcours (Routing). Deux familles d'heuristiques ont vu le jour, les premières et les plus connues sont les méthodes "cluster first, route second" où les clients sont regroupés en clusters puis chaque cluster est transformé en tournée ; la deuxième famille d'heuristiques considère l'inverse puisque une route visitant tous les clients est construite puis partitionnée en plusieurs tournées.

Pour la première famille, on citera l'algorithme "Sweep" [44] qui utilise les coordonnées polaires des clients pour la phase de clustering puis construit une tournée pour chaque cluster. Les coordonnées polaires sont définies à partir du dépôt comme pôle, un client choisi aléatoirement permet de définir l'axe polaire et d'initialiser le premier cluster. Un balayage dans l'ordre croissant des coordonnées angulaires affecte les clients au cluster courant. Si l'ajout d'un client viole la contrainte de capacité dans le cluster courant, un nouveau cluster est créé et défini comme cluster courant. Une fois les clusters créés, un problème du voyageur de commerce est résolu pour chacun.

L'algorithme "Split" [80] de la famille des "route first, cluster second" construit initialement une tournée visitant tous les clients, puis un graphe auxiliaire est créé suivant l'ordre du tour où chaque arc correspond à une tournée réalisable, la résolution du plus court chemin donne une solution du VRP qui est la meilleure solution suivant l'ordre établi par la tournée géante.

Méthodes de recherche locale Contrairement aux heuristiques précédentes, ces méthodes partent d'une solution existante réalisable. Elles utilisent un opérateur de voisinage pour générer une/des nouvelles solutions qui peuvent être meilleures que la courante. Un voisinage est une fonction qui par une modification élémentaire sur une solution permet d'obtenir une nouvelle solution. La méthode génère donc à chaque itération un ensemble de solutions voisines à la recherche d'une amélioration. Une amélioration étant une solution ayant un moindre coût (fonction objectif à minimiser). La procédure est répétée jusqu'à ce qu'il n'existe plus d'amélioration.

Un exemple de voisinage est le 2-opt, qui a été défini pour le problème du voyageur de commerce. Le 2-opt consiste à supprimer deux arcs (a, b) et (c, d) d'une tournée et reformer la tournée en ajoutant les arcs (a, c) et (b, d). Autrement dit, deux clients sont permutés et l'ordre des clients entre eux est inversé. Cette méthode peut être appliquée au VRP dans une tournée ou entre deux tournées mais avec plus de précautions concernant les contraintes de capacité.

Un voisinage simple et spécifique au VRP est celui de l'échange de client entre deux tournées, le principe est donc de choisir deux clients dans deux tournées et les échanger. Cet échange peut être encouragé par la proximité d'un client à une autre tournée et inversement.

Métaheuristiques

Les métaheuristiques sont les méthodes les plus utilisées dans la littérature pour la résolution du VRP et de ses variantes. D'abord, la qualité de la solution obtenue est largement meilleure que celle d'une heuristique, d'ailleurs la majorité des métaheuristiques commencent par une ou plusieurs solutions d'heuristiques et tendent à en obtenir une meilleure. De plus, le temps d'exécution d'une métaheuristique étant généralement un paramètre d'entrée, il est modulable selon le besoin. Nous définirons dans ce qui suit sept métaheuristiques qui nous semblent les plus utilisées pour les problèmes de tournées.

Recuit simulé (SA) : Cette méthode s'inspire de la métallurgie et plus précisement de la technique de réchauffement et de refroidissement contrôlé d'un matériau pour augmenter la taille de ses cristaux et réduire leurs défauts. Partant d'une solution, la méthode la modifie selon

un voisinage pour obtenir une nouvelle ; Si la nouvelle solution améliore la fonction objectif, on l'accepte comme solution courante, sinon on l'accepte avec une probabilité exp $\frac{-\Delta}{T}$. Le Δ étant la différence de coût entre les deux solutions et T la température, paramètre empirique inspiré du phénomène physique. L'acceptation d'une mauvaise solution permet d'explorer une plus grande partie de l'espace des solutions et tend à éviter d'être bloqué sur un optimum local. C'est pour cela que la température est initialisée à une valeur très élevée pour permettre une exploration large de l'espace des solutions puis un décroissement lui est appliqué (réduction continue, par palier..) pour intensifier la recherche à la fin.

On retrouve dans [88] une des premières applications du Recuit Simulé sur un problème de tournées, les auteurs y considèrent un voisinage composé de différents mécanismes tels qu'inverser les tournées ou échanger des clients entre différentes tournées et testent l'algorithme sur des instances allant jusqu'à 500 clients. Depuis, différents travaux se sont intéressés au recuit simulé pour résoudre des problèmes de tournées, on citera ici [24] où plusieurs recuits simulés ont été utilisés simultanément en parallèle et en cohésion pour résoudre un problème de tournées de véhicules avec fenêtres de temps.

La recherche Tabou TS L'algorithme démarre d'une solution initiale et explore son voisinage, la meilleure solution du voisinage est choisie et cela même si elle n'est pas meilleure que la solution courante. Ce choix différent des heuristiques de recherche locale est fait pour éviter d'être bloqué sur un optimum local. Cependant, il est possible et probable de revenir assez rapidement à un optimum local après l'avoir quitté ce qui est appelé : phénomène de bouclage. Pour éviter ce phénomène, une mémoire est utilisée où sont stockées les dernières solutions visitées pour interdire tout déplacement vers une solution déjà explorée. Ces solutions sont déclarées des solutions taboues, d'où le nom de la méthode. Elles sont stockées dans une liste de taille (longueur) donnée, appelée liste tabou. Donc à chaque itération, une meilleure solution du voisinage est choisie si elle ne fait pas partie de la liste tabou.

Dans [42], une recherche tabou est utilisée pour résoudre un VRP classique, les auteurs autorisent l'algorithme à considérer des solutions non réalisables ce qui facilite et diversifie le parcours dans l'espace des solutions. Les résultats obtenus sur des instances classiques sont compétitifs et satisfaisants.

La recherche à voisinage variable VNS Le VNS a été proposé [67] dans les années 90. Cette méthode utilise k voisinages $N = \{N_1, ...N_k\}$ et une méthode de recherche locale en 4 étapes comme suit :

1. Construire une solution initiale S, initialiser $i = 0$
2. Choisir aléatoirement une solution S' voisine de S avec le voisinage N_i
3. Appliquer la recherche locale sur S', soit S'' la solution obtenue
4. Si S'' est meilleure que S, alors $S = S''$ et $i = 1$. Sinon $i = i + 1$
5. Si $i = k$, Stop. Sinon aller à l'étape 2.

L'idée est donc d'échapper à l'optimum par une utilisation aléatoire d'un voisinage différent à chaque fois.

Cette méthode fut largement adaptée et améliorée pour les problèmes de tournées tels que dans [66], [63], [4]

La recherche à voisinage large "large neighborhood search" (LNS) L'objectif ici est de définir un voisinage plus large et pour cela deux opérateurs sont définis : Le premier détruit

partiellement la solution, le deuxième opérateur prenant une solution partielle en paramètre, la reconstruit vers une solution complète. Dans le cas classique du LNS, la solution n'est acceptée que si elle est meilleure, mais il existe plusieurs travaux où un critère d'acceptation différent est utilisé.

Shaw introduit le LNS [97] et le teste sur des instances du CVRP et du VRPTW. La méthode commence avec un seul client à supprimer puis le ré-insère, mais au vu de l'évolution de la solution l'algorithme peut augmenter ce nombre jusqu'à 30 clients. L'auteur estime que les clients qui seront supprimés ensemble doivent être le plus "liés" possible pour augmenter les chances d'une bonne réinsertion et définit une fonction calculant deux à deux cette dépendance entre clients. Il utilise du "Branch and Bound" et/ou une heuristique selon la complexité de la réinsertion. La métaheuristique était prometteuse par les résultats obtenus sur des instances classiques de la littérature mais aussi innovatrice par la façon d'élargir le principe de voisinage tout en le rendant plus simple à adapter aux différentes problématiques et leurs variantes. D'autres chercheurs s'y intéressent dont D.Pisinger et S.Ropke qui introduisent le "Adaptive Large Neighborhood Search ALNS", une généralisation du LNS où plusieurs opérateurs de destruction et de construction sont considérés, et le testent sur des instances du problème de collecte et de livraison avec fenêtre de temps [89]. Plus tard, ils présentent un "framework" dans [79] permettant d'utiliser le ALNS sur diverses variantes du VRP.

L'algorithme génétique GA S'inspirant de la "sélection naturelle" en biologie, à chaque étape de l'algorithme une population de solutions est considérée. Cette population se compose de solutions sélectionnées de la précédente population après divers croisements et mutations.

Les premiers travaux concluants du GA sur du VRP sont ceux de Prins dans [81] où la méthode Split [80] expliquée précédemment est utilisée. Prins utilise des tours géants comme chromosomes, ce qui à la fois facilite les croisements entre différents chromosomes et garantit l'obtention relativement simple de la meilleure solution possible à partir de n'importe quel chromosome.

Colonies de fourmis AS Inspiré du comportement des fourmis, l'algorithme AS utilise plusieurs agents artificiels mimant le comportement d'une fourmi. Initialement les agents cherchent une solution individuellement en diffusant des phéromones sur chaque arc sur leur passage, et moins la solution est coûteuse plus il y'aura de phéromones dessus. Le processus est répété itérativement et les phéromones diffusés à l'itération précédente augmentent les chances des fourmis d'emprunter les arcs les plus intéressants, à la fin du processus seules les meilleures solutions sont marquées par des phéromones. Dans [86], les auteurs obtiennent de bons résultats pour le VRP en procédant en quatre étapes, la première applique l'algorithme AS globalement et obtient une solution, la deuxième étape utilise cette solution pour clustériser le graphe, la troisième étape consiste à appliquer l'algorithme localement sur chaque cluster et finalement les solutions partielles sont rassemblées pour obtenir une solution globale. Tout au long du processus les phéromones sur chaque arc sont sauvegardées et peuvent être utilisées dès la première étape si la procédure est répétée.

2.3.4 Métaheuristiques appliquées au PVRP

Une fois la problématique du PVRP définie dans la littérature plusieurs métaheuristiques sont proposées. Dans [22] une recherche tabou est adaptée, les auteurs explorent itérativement le voisinage de la solution courante en testant des changements de scénarios des clients, ce voisinage inclut des solutions non réalisables en termes de capacité mais une pénalité est ajoutée à leur score

(valeur de la fonction objectif). La solution du voisinage ayant le meilleur score est la nouvelle solution courante. Pour éviter d'être bloqué dans un minimum local, les auteurs ajoutent une pénalité aussi aux solutions ayant des configurations fréquentes.

Un algorithme génétique est utilisé dans [29] où chaque chromosome est une affectation des clients à un de leurs scénarios, L'évaluation d'un chromosome se fait par la résolution d'un VRP pour chaque jour en utilisant l'heuristique de "Clarke and Wright" [19]. L'algorithme applique à chaque itération des croisements et des mutations à une "bonne" sélection de chromosomes obtenus lors de la/ ou des précédente(s) itération(s).

Dans [18], les auteurs utilisent le relaxation du programme Linéaire proposé dans [17] pour obtenir une solution initiale pouvant violer les contraintes de capacité, puis appliquent une modification de scénario à un client, si la modification provoque une amélioration de la solution courante ou engendre une dégradation inférieure à un seuil défini, la modification est acceptée. Le seuil est réduit progressivement et la méthode s'arrête quand aucune modification n'est acceptée.

Ces trois méthodes sont testées sur des instances classiques et une comparaison détaillée est présentée dans le chapitre [37], publié en 2008 et qui reste une référence dans la littérature pour le PVRP et contient une étude exhaustive des travaux faits dessus.

En 2009 les auteurs de [18] proposent une méthode de recherche à voisinage variable VNS. Les auteurs montrent l'efficience de cette méthode sur le PVRP en le comparant au résultats d'autres méthodes de la littérature.

2.4 Les problématiques liées aux véhicules électriques

Les caractéristiques des véhicules ont une importance primordiale dans la modélisation et la résolution des problèmes de tournées. C'est pour cela que l'intégration des véhicules électriques nécessite une nouvelle considération des problématiques de tournées de véhicules. La transition, tout autant pour des raisons écologiques que économiques, des entreprises vers des véhicules électriques nécessite donc de revoir et d'adapter les outils et méthodes de résolution installés.

Description du EVRP

Les premiers travaux sur le problème de tournées incluant des véhicules électriques est [48], où la flotte est composée de véhicules électriques et de véhicules thermiques et l'objectif est de minimiser les coûts fixes et variables générés durant la tournée. Les véhicules électriques peuvent recharger à tout emplacement de la tournée et un temps de recharge fixe est rajouté à la tournée. Dans [31] les auteurs étudient un problème de tournées de véhicules avec capacité en considérant une flotte composée de véhicules alternatifs. Les véhicules alternatifs étant des véhicules utilisant des "carburants" non fossiles et dont la recharge nécessite une attention particulière (peu de stations, temps de recharge conséquent...). Ils intègrent la possibilité pour les véhicules de visiter une station de recharge en cas de nécessité pour finir la tournée, mais un véhicule ne peut visiter qu'une seule station et doit y être rechargé complètement. Cette problématique sous le nom de "Green vehicle routing problem GVRP" est formulée en programme linéaire en nombres entiers. L'objectif du GVRP, comme défini dans l'article, est de construire autant de tournées que de véhicules alternatifs tout en prenant en compte l'autonomie de chaque véhicule et en permettant des visites à des stations de recharge si besoin. Contrairement à la modélisation classique du VRP, le GVRP ne considère pas une demande en marchandises du client mais un temps de service propre à chaque client, de fait la contrainte de capacité n'est pas requise mais une contrainte de temps maximum T_{max} par tournée est introduite.

Nous présenterons dans ce qui suit, le modèle du GVRP proposé dans [31] en considérant les véhicules alternatifs comme exclusivement des véhicules électriques. La première étape de modélisation est d'intégrer l'ensemble B des stations de recharge dans le graphe du modèle mais nous savons que dans le modèle classique du VRP les sommets, à l'exception du dépôt, ne peuvent être visités qu'une seule fois principalement à cause des contraintes d'élimination des sous-tours. Il est donc nécessaire pour permettre plusieurs visites à une station de recharge de lui créer autant de copies que de visites permises. Il ne suffit donc pas d'ajouter les stations de recharge considérées au graphe, mais un ensemble B' de plusieurs copies de chacune d'entre elles. Ce nombre de copies est un paramètre important à considérer puisqu'il doit être assez grand pour ne pas éliminer des solutions intéressantes mais minimal pour ne pas avoir un graphe considérable. Nous renvoyons le lecteur vers [31] pour estimer et fixer le nombre de copies nécessaires

Soit donc le graphe non orienté $G = (V, A)$, V étant composé du dépôt $\{0\}$, de C l'ensemble des clients et de l'ensemble B' de copies des stations de recharge. Chaque arête $\{i, j\}$ est caractérisée par une distance d_{ij} en kilomètres, un temps de parcours t_{ij} et un coût c_{ij}. Nous avons donc un ensemble V de m véhicules électriques (VE) partant du dépôt avec une recharge maximum Q_e et avec une consommation linéaire de r par kilomètre.

Notations

$$p_i = \begin{cases} \text{Temps de service si } i \text{ est un client} \\ \text{Le temps d'une recharge complète si } i \text{ est une station de recharge} \\ 0 \text{ sinon (dépôt)} \end{cases}$$

Variables Les variables considérées sont : y_i le niveau de recharge du véhicule en arrivant au sommet i ; π_i l'heure d'arrivée du véhicule chez le client i et x_{ij} la variable binaire égale à 1 si Si l'arête $\{i, j\}$ est parcourue.

Les variables y et π sont initialisées au départ du dépôt à $y_0 = Q$ et $\pi_0 = 0$. A chaque visite d'une borne la variable y est réinitialisée alors que la variable π est augmentée du temps d'une recharge complète.

Modèle mathématique

$$\min_{x_{ij}} \quad \sum_{\{i,j\}\in A} c_{ij}x_{ij} \tag{2.4a}$$

$$\text{s.t.} \quad \sum_{i\in C}^{i\neq j} x_{ij} = 1 \qquad \forall j \in C \tag{2.4b}$$

$$\sum_{i\in C}^{i\neq j} x_{ij} \leq 1 \qquad \forall j \in B \tag{2.4c}$$

$$\sum_{i\in V} x_{ip} = \sum_{j\in V} x_{pj} \qquad \forall p \in V \tag{2.4d}$$

$$\sum_{j\in V}^{j\neq 0} x_{0j} \leq m \tag{2.4e}$$

$$\sum_{i\in V}^{i\neq 0} x_{i0} \leq m \tag{2.4f}$$

$$\pi_j - \pi_i \geq (t_{ij} + p_j)x_{ij} - T_{max}(1 - x_{ij}) \qquad \forall i,j \in V \tag{2.4g}$$

$$0 \leq \pi_0 \leq T_{max} \tag{2.4h}$$

$$t_{0j} \leq \pi_j \leq T_{max} - (t_{j0} + p_j) \qquad \forall j \in V, j \neq 0 \tag{2.4i}$$

$$y_j \leq y_i - (r.d_{ij}x_{ij}) + Q(1 - x_{ij}) \qquad \forall j \in C, i \in V \tag{2.4j}$$

$$y_j = Q \qquad \forall j \in B \tag{2.4k}$$

$$y_j \geq \min\{r.d_{j0}, r.(d_{jl} + d_{l0})\} \qquad \forall j \in C, \forall l \in B \tag{2.4l}$$

L'objectif (2.4a) est donc de minimiser les coûts de parcours, les contraintes de (2.4b) à (2.4f) sont les contraintes classiques du VRP. Les contraintes (2.4g) servent à calculer l'heure d'arrivée à chaque sommet et en combinaison avec les contraintes (2.4h) et (2.4i) elles vérifient que chaque véhicule revient au plus tard à T_{max} au dépôt. Les contraintes (2.4k) initialisent la charge des véhicules au départ du dépôt à Q alors que les contraintes (2.4j) calculent le niveau de charge du véhicule à chaque sommet en réduisant du niveau de charge du véhicule calculé au sommet précédent, la consommation de l'arête parcourue. Les contraintes (2.4l) assurent qu'à chaque client visité, il reste assez d'énergie pour revenir au dépôt directement ou en passant par une station de recharge.

Le modèle du GVRP fut un des premiers et des plus reconnus concernant le EVRP et il reste une base concernant les problèmes de tournées des véhicules électriques malgré le fait que beaucoup de contraintes réelles n'y sont pas considérées. Nous présentons dans la section (2.5) quelques variantes du EVRP présentes dans la littérature et qui partent du VRP ou de ses variantes ou/et qui intègrent d'autres contraintes liées à l'écosystème du véhicule électrique.

EVRP périodique

Malgré les différents travaux concernant les tournées des véhicules électriques, le problème périodique de tournées de véhicules n'a pas été étendu, à notre connaissance, aux véhicules électriques dans la littérature. Les seuls travaux qui s'en rapprochent sont ceux de [65] où les auteurs s'intéressent au "Periodic green vehicle routing problem" une problématique écologique (green) par le fait d'intégrer la minimisation des émissions des voitures thermiques et non pas par l'intégration des véhicules électriques.

Nous avons précédemment présenté quelques travaux sur des problématiques de tournées incluant les véhicules électriques. Sachant que le PVRP implique en plus des tournées une question de planification sur un horizon de temps, nous nous intéressons brièvement dans cette partie aux travaux sur des problématiques de planification impliquant des véhicule électriques.

La seule étude considérant une problématique avec véhicules électriques sur un horizon de temps est celle de [106] où les auteurs traitent la problématique de localisation affectation de nouvelles stations de recharge. La problématique n'est donc pas une problématique de tournées mais de localisation. En plus de choisir la localisation, l'objectif est de donner un planning de construction le tout se basant sur des demandes dynamiques simulant l'utilisation des véhicules électriques.

2.5 État de l'art des problématiques liées aux véhicules électriques

Les problématiques de tournées liées aux véhicules électriques sont encore récentes mais leurs difficultés et importances font qu'il existe déjà un nombre considérable de travaux dessus. Ces travaux sont inspirés généralement de problématiques réelles et traitent différentes contraintes spécifiques, il est donc difficile de les classifier par variante. Nous présentons dans ce qui suit un état de l'art aussi exhaustif que possible, des différents travaux et variantes du EVRP de la littérature.

Les premiers travaux concernant les tournées des véhicules électriques sont ceux de [48], les auteurs s'intéressent à une problématique de collectes et de livraisons avec une flotte composée de véhicules thermiques et de véhicules électriques. Ils proposent un modèle où la recharge est possible tout au long de la route donc sans considération physique d'une station de recharge. Un modèle est proposé, où une variable est ajoutée pour vérifier l'autonomie et ajouter un temps de recharge à chaque fois que l'autonomie est à zéro.

Après ces premiers travaux plusieurs variantes et leurs modélisations furent présentées, on citera ici le Green vehicle routing problem [31] décrit précédemment et pour lequel, après modélisation, deux heuristiques constructives et une heuristique améliorative classiques du VRP seront adaptées. Les auteurs dans [68] reprennent la même problématique et proposent une heuristique "route first, cluster second" basée sur une génération d'un ensemble de tournées réalisables puis la résolution d'un problème de partition pour construire la meilleure solution. Pour obtenir des tournées réalisables, les auteurs proposent un mécanisme permettant d'insérer optimalement des stations de recharge dans une tournée violant la contrainte d'énergie.

Schneider et al. [96] étudient un VRP avec fenêtre de temps en ne considérant que des véhicules électriques mais avec une flotte non limitée. Les visites à des stations de recharge ne sont pas limitées mais un véhicule doit recharger complètement sa batterie et le temps de recharge est calculé selon l'énergie restante dans le véhicule. Une formulation et une hybridation d'une méthode de "recherche à voisinage variable" et d'une recherche tabou sont proposées. On trouve dans [20] la même problématique posée mais les auteurs ne considèrent pas de stations de recharges distinctes mais permettent aux véhicules de recharger lors de la visite de certains clients ; les auteurs proposent une formulation mathématique et en déduisent différentes bornes pour la solution optimale. Les travaux dans [50] reprennent la même problématique proposée dans [96] mais considèrent plusieurs types de véhicules électriques qui diffèrent par leurs capacités et leurs coûts.

Breunig et al. proposent "the electric two-echelon VRP" [12], problématique où les marchandises sont transportées en deux étapes : la première utilisant des véhicules thermiques permet de

ravitailler des dépôts intermédiaires ; dans la deuxième étape, des véhicules électriques sont utilisés pour livrer les marchandises des dépôts intermédiaires aux clients. Dans la première étape, les véhicules thermiques sont adaptés pour les longues distances contrairement au VE, alors que dans la deuxième étape, la non pollution des VEs est un avantage pour le milieu urbain. Le problème a été formulé sur un multigraphe et résolu avec des méthodes exactes pour des petites instances ; une métaheuristiques LNS est utilisée pour des instances plus larges.

Les auteurs de [59] partent de tournées déjà construites et d'une flotte de bus hétérogène (thermiques, électriques, hybrides), ils considèrent la problématique de remplacement des anciens bus par des nouveaux sous un certain budget tout en optimisant l'affectation des tournées aux bus et en envisageant des passages par des stations de recharge si nécessaire. La problématique de passage d'une flotte de bus thermique à une flotte de bus électrique (EBFTP) est posée dans [76], les auteurs modélisent la transition vers une flotte de bus plus écologique, pour une entreprise de transport, en programmation linéaire en nombres entiers. Une transition sur plusieurs périodes est considérée, ainsi que plusieurs facteurs tels que les coûts d'achat, les coûts d'exploitation, ou les investissements dans les infrastructures de recharge. [105] considère le problème d'affectation de tournées construites à des bus électriques exclusivement en prenant en compte la recharge partielle, le but étant de minimiser la distance parcourue et surtout le nombre de bus dans une optique stratégique.

The green and pollution vehicle routing problem est une variante du VRP qui se concentre sur l'aspect écologique et dont l'objectif est de promouvoir l'utilisation des énergies renouvelables et de minimiser les émissions de gaz à effet de serre. Le véhicule électrique n'ayant pas d'émissions directes, l'auteur dans [34] calcule un gain écologique allant jusqu'à 80 % par rapport à un véhicule thermique. Les auteurs proposent aussi un modèle calculant les émissions provoquées par un VE selon les routes empruntées dans différents pays.

Le principal problème que posent les VE dans le transport de marchandises est l'autonomie limitée des batteries. Les auteurs dans [49] ont analysé plus de 40 VE disponibles à l'échelle mondiale qu'ils classent en catégories : petits, moyens-grands, très performants, et les voitures de sport. Tous les modèles de VE utilisent une technologie à base de lithium, en particulier les batteries au lithium-ion [26] avec une capacité de batterie et une autonomie variant entre 12 et 90 kWh et 85 à 528 Km. Un VE personnel moyen a une capacité de batterie de 30 kWh, ce qui est suffisant pour voyager 250 km. Dans le transport, les véhicules électriques utilisés ont une autonomie plus courte (160-240 km). Les batteries montées dans les VE sont la cause principale des coûts d'acquisition élevés d'un côté et aussi des limitations techniques à mesure que la batterie se dégrade, entraînant une diminution de la capacité maximale. Les auteurs dans [75] ont conclu que la batterie devrait être remplacée après cinq à dix ans. Les auteurs ont également décrit les facteurs qui influencent une telle dégradation de la batterie : surcharge, batterie faible, hautes et basses températures...

Recharger complètement un véhicule peut atteindre jusqu'à huit heures selon la technologie utilisée. Il peut donc être rédhibitoire de recharger complètement un véhicule pendant une tournée quand une recharge partielle suffit pour finir la tournée ou même visiter d'autres clients. De plus si le coût de la recharge est dépendant de l'énergie rechargée, le rechargement partiel en devient encore plus intéressant d'un point de vue économique. C'est pour ces raisons que différents travaux ont intégré la possibilité de recharger partiellement dans leurs méthodes de résolution. [33] reprend le green VRP en intégrant la recharge partielle et plusieurs technologies de recharges et montrent à travers des comparaisons que l'intégration des recharges partielles permet d'envisager plus de solutions et d'avoir des gains considérables en termes de coûts. Les travaux de Sassi et al. [95, 92, 94] intègrent en plus de la recharge partielle, la possibilité de choisir la technologie de recharge à la station tout en respectant la compatibilité entre véhicules

et bornes. Les travaux dans [54] considèrent la recharge partielle et trois technologies de recharge pour une problématique de tournées de véhicules avec fenêtres de temps. En plus de la recharge partielle, les auteurs dans [55] considèrent l'éventualité ou le véhicule électrique doit faire la queue à son arrivé à une station de recharge. Les auteurs divisent une journée en cinq intervalles (matin, midi, fin d'après-midi, soir et nuit) et utilisent les équations du système de file d'attente M/G/1 pour estimer les temps d'attente dans chaque intervalle.

Globalement la consommation d'énergie du véhicule électrique est modélisée linéairement par relaxation, mais différents travaux se sont intéressés à cette consommation et ont présenté d'autres modèles plus précis tels que [43] qui modélise la consommation en morceaux selon différents paramètres tels que la qualité de la route, le type de moteur, la vitesse du véhicule et le niveau de charge de la batterie. Ces paramètres permettent de modéliser l'énergie consommée par le moteur et celle récupérée par le mouvement du véhicule, en plus d'une estimation de la consommation des services auxiliaires (climatisation, chauffage...). Ce modèle est comparé à une consommation linéaire dans [63] pour un problème Dial-a-ride, les auteurs montrent que le modèle réaliste a un gain de 0.14% par rapport au modèle linéaire sur les solutions obtenues. Plus généralement, les performances des véhicules électriques dépendent de plusieurs facteurs et une analyse de ses comportements permet une meilleure gestion des problématiques touchant au routing. Dès 2014, des tests sont faits en Colombie [87] pour fixer les différents paramètres influant sur la consommation d'énergie dans un VE. Les résultats montrent que jusqu'à 30% de la batterie est utilisée par des services auxiliaires tels que la climatisation ou chauffage. Ils montrent aussi l'impact de la qualité des routes sur la consommation et préconisent l'utilisation des véhicules électriques dans un milieu urbain. Les travaux dans [45] utilisent un modèle réaliste de consommation d'énergie qui intègre la vitesse, la pente et la répartition de la charge dans un problème de tournées de véhicules avec fenêtres de temps et flotte mixte ; les auteurs développent une métaheuristique ALNS pour la résolution et constatent une amélioration de la qualité de la solution par rapport à une considération linéaire de la consommation d'énergie.

Comme la consommation d'énergie, la recharge des véhicules électriques est souvent relaxée à une fonction linéaire. Dans [69] les auteurs introduisent le E-VRP-NL, une extension du EVRP où la recharge est approchée par une fonction linéaire par morceaux. Les auteurs comparent leur approximation de la fonction de recharge à celles de la littérature et montrent que le fait de négliger la nature non-linéaire de la recharge peut conduire à des solutions infaisables ou trop coûteuses. Les auteurs dans [39] reprennent la même problématique (E-VRP-NL) et proposent deux nouvelles formulations en programmes linéaires à variables mixtes. La première formulation utilise des variables indexées sur les arcs pour calculer le temps de parcours et la consommation d'énergie. La deuxième formulation nécessite l'énumération de la séquence de stations de recharge qui peuvent être visitées par un VE entre chaque paire de sommets hors station (client ou dépôts). Cette énumération permet de construire un mutligraphe ayant que le dépôt et les clients comme sommets. Cette dernière formulation ne nécessite donc pas de générer des copies pour chaque station, ce qui représente un avantage par rapport aux précédentes formulations.

Une première revue de la littérature est publiée en 2016 par Juan et al. [53], les auteurs s'y intéressent à toutes les problématiques touchant à l'écosystème du véhicule électrique. Une autre revue est publiée en 2019 [30], où les auteurs se sont plus concentrés sur les problématiques de tournées de véhicules impliquant des véhicules électriques et leurs résolutions.

2.6 Conclusion

Dans ce chapitre, nous avons présenté les bases des problèmes de tournées de véhicules thermiques ainsi qu'une synthèse des différents travaux sur les problématiques de tournées de véhicules électriques. Ceci nous a permis de constater que très peu de travaux existent dans la littérature sur les tournées périodiques avec des véhicules électriques. Cette synthèse nous aidera aussi à adapter et utiliser les méthodes de résolution les plus efficaces.

Chapitre 3

Problème de planification des tournées de véhicules électriques avec mono-visite sur l'horizon : définitions et heuristiques

Sommaire

3.1 Introduction

Le PVRP a été largement étudié dans la littérature car il permet de modéliser plusieurs applications réelles comme la livraison de colis, les tournées des services à domicile. Dans ce type

d'activités, l'introduction des VE's commence à connaître une bonne croissance d'où l'intérêt d'étudier le problème de planification de véhicules électriques.

Dans ce chapitre nous proposons le premier modèle de la littérature pour le PEVRP. Nous avons choisi de considérer le cas particulier où les clients exigent une visite unique sur l'horizon (fréquence égale à 1) pour mieux analyser les mécanismes permettant de gérer les véhicules électriques sur une période de planification de plusieurs jours sans s'intéresser, dans un premier temps, à la difficulté combinatoire engendrée par des visites multiples.

Notre problème est évidemment NP-difficile puisqu'il inclut le EVRP comme un cas particulier lorsque la période est réduite à une seule journée. Il est donc très difficile de résoudre de manière exacte des instances de tailles conséquentes en un temps réaliste. La meilleure façon de s'attaquer à ce problème est alors d'utiliser des approches heuristiques. Dans cette section, nous proposons une extension de l'heuristique de meilleure insertion à savoir BIH (Best Insertion Heuristic). Une autre approche, basée sur l'analyse des clusters, à savoir CLH (Clustering heuristic) est également proposée. Deux variantes du CLH sont développées, la première étant séquentielle dans l'insertion des clients dans les clusters et la deuxième traitant les insertions de manière récursive.

3.2 Définition du problème et formulation mathématique

Nous nous intéressons dans ce chapitre au problème de planification des tournées de véhicules électriques avec fréquence unique. Ce problème est à la frontière du niveau tactique et opérationnel, contenant le problème EVRP et un problème de planification des visites sur un horizon de temps. Il consiste sommairement à affecter une journée de visite à chaque client et à construire parallèlement les tournées pour chaque véhicule de chaque jour de l'horizon.

Notre problème est donc celui de l'acheminement périodique des véhicules électriques, dans lequel un ensemble de clients nécessitent chacun une visite unique sur la période de planification, cette visite devant respecter l'ensemble des options de visites exprimées par les clients. La visite d'un client doit donc être programmée dans une des journées où ils sont disponibles. L'objectif classique est de minimiser le coût total, y compris les coûts de routage et les coûts de recharge au cours des tournées. Ce problème s'apparente au problème périodique des tournées de véhicules dont il est un cas particulier par le fait d'avoir une fréquence unique pour chaque client et une variante par le fait d'utiliser exclusivement des véhicules électriques.

Le PEVRP se définit sur un graphe complet $G = (V, A)$. V désigne l'ensemble des sommets composé de l'ensemble C des n clients, un dépôt noté par 0, et l'ensemble B de ns stations de recharge. A est l'ensemble des arc, $A = \{(i, j)|i, j \in V\}$. Chaque arc (i, j) de A est décrit par la distance $d_{i,j}$, le temps de déplacement $t_{i,j}$, et le coût du déplacement $c_{i,j}$. Lorsqu'un arc (i, j) est parcouru par un véhicule électrique (EV), il consomme une quantité d'énergie $e_{i,j} = r \times d_{i,j}$, où r indique un taux de consommation d'énergie constant. Chaque client i a une demande q_i, et un temps de service s_i. Nous considérons un horizon de temps H de np périodes typiquement "jours", dans lequel chaque client i a une fréquence $f(i) = 1$, et un ensemble de jours de visite autorisés $D(i) \in H$. Cela signifie que le client i doit être servi exactement une fois sur l'horizon, et que ce jour de visite appartient à $D(i)$.

Au dépôt, une station de recharge est localisée et permet aux véhicules de recharger pendant la nuit. Pendant les différentes journées de l'horizon, il est possible de visiter les différentes stations, incluant celle localisée au dépôt, et recharger moyennant un coût de recharge défini. La plupart des études de la littérature sur le EVRP considèrent que le coût de la recharge dépend soit de la puissance délivrée par les bornes, soit du temps total consacré à la recharge des véhicules.

Dans notre travail, nous considérons un coût de recharge fixe de Cc, qui ne dépend ni de la quantité d'énergie livrée, ni du temps nécessaire pour recharger le véhicule. Cette hypothèse est plus réaliste dans la mesure où en France, les opérateurs n'ont pas le droit de vendre de l'énergie mais vendent un service de recharge qui est donc forfaitaire. Les prix ne dépendent donc pas de la quantité d'énergie, mais de la qualité du service (recharge rapide ou lente) [93].

Soit Ve l'ensemble des m véhicules électriques, chacun ayant une capacité Q d'emport des marchandises et une batterie de capacité E. Les véhicules partent avec une recharge complète du dépôt.

Une solution réalisable à notre problème se compose d'un ensemble de tournées et d'une planification des recharges pour chaque véhicule sur l'horizon. Une tournée réalisable est une séquence de sommets qui satisfait l'ensemble des contraintes suivantes :

- Chaque tournée doit commencer et se terminer au dépôt ;
- La quantité totale des marchandises livrées le long de la tournée, donnée par la somme des demandes q_i pour chaque client visité, ne doit pas dépasser la capacité du véhicule Q ;
- La durée totale de chaque tournée, calculée par la somme des durées de parcours entre les sommets visités, la somme des temps de recharge sur les potentiels bornes visitées et le temps de service de chaque client visité, ne peut dépasser T_{max} ;
- Pas plus de m véhicules électriques sont utilisés ;
- Chaque client doit être visité une fois au cours de la période de planification, et le jour de la visite doit être dans $D(i)$.

Le PEVRP consiste à affecter chaque client i à un jour de service $d \in D(i)$ et à ordonner les visites pour minimiser le coût total des tournées et de la recharge sur H. La fonction objectif à minimiser est $f(S) = \sum_{tr \in S} TourCost(tr)$, où les tr sont les tournées composant la solution S et $TourCost$ la fonction calculant le coût d'une tournée. $TourCost(tr) = \alpha \times TourDist(tr) + Cc \times nbs(tr)$ où $TourDist(tr)$ est la distance totale de la tournée tr, et $nbs(tr)$ est le nombre total de visites aux stations de recharge dans la tournée tr. α est un poids donné représentant le coût d'une unité de distance.

Pour le modèle mathématique nous allons partir du même graphe G décrit plus haut à l'exception du dépôt qui sera dupliqué en deux sommets, un dépôt de départ $\bar{0}$ et un d'arrivée $\underline{0}$. Cette modification est nécessaire pour calculer l'heure de fin d'une tournée et vérifier qu'elle ne dépasse pas T_{max}, $O = \{\bar{0}, \underline{0}\}$ devient donc l'ensemble des deux. Aussi nous utiliserons l'ensemble B' qui ne représentera plus l'ensemble des stations mais un ensemble contenant plusieurs copies de chaque station. Cela permet de garder un schéma où tous les sommets, à l'exception du dépôt, ne peuvent être visités qu'une fois tout en permettant de visiter une borne plusieurs fois dans la solution. Cette unicité permet d'un coté d'exprimer les contraintes d'élimination des sous-tours et de l'autre de ne pas confondre deux tournées qui visitent la même borne. Nous renvoyons le lecteur vers [37] où la fixation du nombre de copies pour chaque borne est discutée. Ci-dessous nous allons recenser les notations et variables nécessaires pour le modèle puis nous présenterons le modèle. Cela nous donnera donc $V' = C \cup B' \cup O$, et on posera $V'_{/O} = C \cup B'$ l'ensemble des sommets à l'exception des dépôts.

Variables La variable principale permet de décider si un arc est emprunté par un véhicule ou non et cela pour chaque véhicule et chaque jour de l'horizon :

$$x^d_{ijk} = \begin{cases} 1 \text{ Si l'arc}(i,j) \text{ est emprunté par le véhicule } k \text{ le jour } d \\ 0 \text{ Sinon} \end{cases}$$

Paramètres	**Définitions**
$\bar{0}$ ($\underline{0}$)	Copie du dépôt d'où partent (arrivent) les véhicules
O	$= \{\bar{0}, \underline{0}\}$
C	Ensemble des n clients
B	Ensemble des ns bornes
B'	Ensemble des ns' copies des bornes
V'	Ensemble de tous les sommets $= B' \cup C \cup O$
$V'_{/O}$	Ensemble des sommets à l'exception du dépôt $= B' \cup C$
V_e	Ensemble des m véhicules
H	L'ensemble np des jours de l'horizon
$D(i)$	Ensemble des jours où le client i est disponible
t_{ij}	Temps de parcours de l'arc (i, j)
e_{ij}	Énergie nécessaire pour parcourir l'arc (i, j)
c_{ij}	Coût de parcours de l'arc (i, j)
Cc	Coût forfaitaire d'une visite à une borne
T_{max}	Durée maximale d'une tournée
s_i	Temps de service du client i
q_i	Demande du client i
$taux$	Temps de recharge par unité d'énergie
Q	Capacité de charge des marchandises d'un véhicule
E	Capacité maximale de la batterie d'un VE

TABLE 3.1 – Notations

D'autre variables intermédiaires sont nécessaires pour vérifier les contraintes de temps et d'énergie :

- y_i Quantité d'énergie encore disponible dans la batterie du véhicule en arrivant au sommet i
- z_i Quantité d'énergie rechargée à la borne i, cette variable est exclusive aux stations
- π_i Heure d'arrivée du véhicule visitant le sommet i. Sachant qu'un client n'est visité qu'une fois, il n'est pas nécessaire de connaître le véhicule faisant cette visite.

Modèle mathématique

$$\min \quad \sum_{d\in H}\sum_{k\in V_e}\sum_{\{i,j\}\in A} c_{ij}x^d_{ijk} + Cc\sum_{d\in H}\sum_{k\in V_e}\sum_{i\in B'}\sum_{j\in V'} x^d_{ijk} \tag{3.1a}$$

$$\text{s.t.} \quad \sum_{d\in D(j)}\sum_{k\in V_e}\sum_{i\in V', i\neq j} x^d_{ijk} = 1 \qquad \forall j \in C \tag{3.1b}$$

$$\sum_{d\in H}\sum_{k\in V_e}\sum_{i\in V', i\neq j} x^d_{ijk} \leq 1 \qquad \forall j \in B' \tag{3.1c}$$

$$\sum_{i\in V'} x^d_{ipk} = \sum_{j\in V'} x^d_{pjk} \qquad \forall d \in H, \forall p \in V'_{/O}, \forall k \in V_e \tag{3.1d}$$

$$\sum_{j\in V', j\neq \bar{0}} x^d_{\underline{0}jk} = 1 \qquad \forall d \in H, \forall k \in V_e \tag{3.1e}$$

$$\sum_{i\in V', j\neq \underline{0}} x^d_{i\bar{0}k} = 1 \qquad \forall d \in H, \forall k \in V_e \tag{3.1f}$$

$$\pi_j - \pi_i + T_{max}(1 - x^d_{ijk}) \geq (t_{ij} + s_i)x^d_{ijk} \qquad \forall i \in C, j \in V', d \in H, \forall k \in V_e \tag{3.1g}$$

$$\pi_j - \pi_i + (M)(1 - x^d_{ijk}) \geq t_{ij}x^d_{ijk} + z_i * taux \qquad \forall i \in B', j \in V', \forall k \in V_e \tag{3.1h}$$

$$t_{0j} \leq \pi_j \leq T_{max} - (t_{j0} + s_j) \qquad \forall j \in C \tag{3.1i}$$

$$t_{0j} \leq \pi_j \leq T_{max} - (t_{j0} + taux * z_j) \qquad \forall j \in B' \tag{3.1j}$$

$$\pi_{\bar{0}} = 0, \quad \pi_{\underline{0}} \leq T_{max} \tag{3.1k}$$

$$y_j - y_i + e_{ij}x^d_{ij} \leq E(1 - x^d_{ij}) \qquad \forall i \in C, j \in V', d \in H \tag{3.1l}$$

$$y_j - y_i + e_{ij}x^d_{ij} \leq E(1 - x^d_{ij}) + z_i \qquad \forall i \in B', j \in V', d \in H \tag{3.1m}$$

$$y_{\bar{0}} = E, \quad y_{\underline{0}} \geq 0 \tag{3.1n}$$

$$0 \leq z_j \leq E - y_j \qquad \forall j \in B' \tag{3.1o}$$

$$\sum_{i\in C} q_i \times \sum_{j\in C} x^d_{ijk} \leq Q \qquad \forall k \in V' \tag{3.1p}$$

La fonction objectif (3.1a) est la somme des coûts des arcs empruntés dans la solution et des coûts des stations visitées. Les contraintes (3.1b),(3.1c) traduisent le fait qu'un client doit être visitée une fois sur l'horizon, et que la copie d'une borne ne peut être visité plus qu'une fois. Les contraintes de conservation de flot (3.1d) imposent que le même véhicule qui a visité un sommet (client ou borne) un jour donné, "reparte" de chez ce client le même jour. Les contraintes (3.1e),(3.1f) assurent que chaque véhicule fait une tournée par jour. Les contraintes (3.1g),(3.1h) servent à calculer l'heure d'arrivée chez chaque client (borne). Pour la contrainte (3.1h), nous avons utilisé M, qui est une borne supérieure de $(\pi_i + z_i * taux)$, et qui peut être exprimé par $T_{max} + taux * E$. Les contraintes (3.1i) et (3.1j) vérifient que les tournées ne dépassent pas la durée maximale. Pour calculer l'énergie d'un véhicule une fois arrivé chez un client ou une borne, nous avons établis les contraintes (3.1l) et (3.1m). La contrainte (3.1o) permet de vérifier que la recharge ne dépasse pas la capacité de la batterie d'un véhicule. Les contraintes (3.1k) et (3.1n) initialisent et bornent les variables de temps et de recharge pour les dépôts (initial et final). Les contraintes (3.1p) vérifient que les marchandises nécessaires pour satisfaire l'ensemble des demandes des clients d'une tournée ne dépassent pas la capacité du véhicule.

Cette formulation nécessite ν variables et μ contraintes :

$$\nu = n^2 \times m \times np + ns'^2 \times m \times np + m \times np + 2n + 3ns' + 2 \tag{3.2}$$

$$\begin{aligned}\mu =& n^2 \times np \times m + n^2 \times np + ns'^2 \times m + n \times ns' \times np \times m + n \times np \times m \\ &+ ns' \times np \times m + n \times ns' \times m + n \times ns' \times np + n \times np + 3np \times m \\ &+ 3n + 4ns' + m + 1\end{aligned} \tag{3.3}$$

3.3 Représentation et évaluation d'une solution

3.3.1 Représentation d'une solution

Pour représenter la solution, il est nécessaire de décrire l'ordre des clients dans chaque tournée, et l'affectation de chaque tournée à un jour de l'horizon tout en respectant le nombre de véhicules disponibles chaque jour. D'un autre côté, la recharge partielle étant possible il est aussi nécessaire de préciser la quantité d'énergie rechargée à chaque visite d'une borne.

Nous avons représenté la solution en matrice où chaque vecteur ligne est une tournée, chaque tournée (vecteur) commence et finit par un 0 représentant le dépôt. Les clients sont représentés par leurs indices appartenant à $\{1, .., |C|\}$, les bornes sont aussi représentées par leurs indices appartenant à $\{|C|+1, .., |C|+|B|\}$. Après l'indice d'une borne, la quantité d'énergie rechargée est affichée en % de la batterie du véhicule. Cette représentation nécessite d'avoir les données de l'instance, telles que la matrice distance ou la capacité des véhicules.

Un exemple d'une solution pour 2 véhicules et trois jours avec 15 clients et deux bornes, les indices des clients sont donc dans $\{1, .., 15\}$ et ceux des bornes dans $\{16, 17\}$.

Jour	Véhicule	Ordre des sommets visités			
1	1	1	2	16	3
1	2				
2	1	4	5	6	7
2	2	8	9	10	17
3	1	13	11	12	
3	2	14	15		

TABLE 3.2 – Exemple d'une solution

Cette solution 3.2 sera représenté par la matrice 3.3.

La première tournée du premier jour est celle qui visite les sommets $\{1, 2\}$ puis visite la borne 16 où le véhicule recharge 53,4 % de batterie en plus de ce que contenait sa batterie, puis finit sa tournée en visitant le client 3. Le deuxième véhicule n'est pas utilisé le premier jour, la tournée étant vide. Les deux suivantes sont celles du deuxième jour et finalement les deux dernières du troisième.

Il est évident que cette représentation ne suffit pas à vérifier une solution ou à en calculer le coût, pour cela il est nécessaire d'avoir les données liées à l'instance telles que les matrices de distances et de coût, la capacité des véhicules, etc.

0	1	2	16	53,4 %	3	0
0	0					
0	4	5	6	7	0	
0	8	9	10	17	20,5 %	0
0	13	11	12	0		
0	14	15	0			

TABLE 3.3 – Matrice représentant une solution

3.3.2 Évaluation d'une solution

Comme explicité dans le modèle mathématique, le coût d'une solution se compose du coût des arcs parcourus dans les différentes tournées et des coûts de recharge engendrés par les visites aux bornes. Cette fonction de coût f, qu'on nommera de coût simple, est significative pour comparer différentes solutions à condition que tous les clients soient visités dans chacune d'elles. Dans les cas de solutions trouvées n'incluant pas la visite de tous les clients, cette fonction perd de sa pertinence puisqu'elle s'appuie sur le postulat que tous les clients sont atteints. De plus, cette fonction n'intègre pas de coût fixe relatif aux véhicules utilisés et ne permet donc pas de comparer des solutions ayant un nombre de véhicules différent.

Pour pallier à cette éventualité, nous avons établi une deuxième fonction d'évaluation, la fonction de coût complet, elle part du coût simple et y ajoute un coût fixe pour chaque utilisation d'un véhicule dans la solution (tournée non vide) et des pénalités calculées sur la base des clients non visités. Premièrement, un coût fixe C_f est ajouté pour chaque véhicule utilisé dans la solution. La fonction $nb_{ve}()$ retourne le nombre de tournées non vides dans la solution et donc le nombre de véhicules utilisés. Deuxièmement, les pénalités sont calculées en estimant les véhicules supplémentaires nécessaires pour atteindre les clients non visités. On simule donc l'ajout de nouveaux véhicules, qu'on appellera véhicules virtuels, et on construit des tournées avec les clients non visités tout en respectant les contraintes liés au véhicules (capacité, temps) et des clients (jours de disponibilité). Pour chaque tournée (non vide) ajoutée, le coût de la tournée (parcours, recharge) et un coût fixe C_{f2} liés au véhicule virtuel supplémentaire sont ajoutés dans le coût complet. Le coût d'un véhicule virtuel C_{f2} est logiquement supérieur au coût d'un véhicule réel C_f pour pénaliser les solutions incomplètes.

Pour avoir le nombre de véhicules virtuels nécessaires et construire les tournées, nous avons utilisé une procédure de "First Improvement" sur les clients non visités. Cette procédure est explicitée dans l'Algorithme 1.

L'Algorithme 1 part donc d'une solution S, son coût simple $f(S)$ et de la liste des clients non visités L. La première étape de l'algorithme est de rajouter le coût fixe des véhicules utilisés dans la solution S (ligne 3). Si des clients ne sont pas visités ($L \neq \emptyset$), l'algorithme crée une tournée T_d pour chaque jour d de l'horizon puis tente l'insertion des clients non visités dans ces tournées créées en adoptant une politique de moindre coût. Chaque client inséré dans une tournée est supprimé de L. Pour chaque tournée T_d où des clients sont insérés, l'algorithme rajoute le coût de cette tournée ($TourCost(T_d)$) et une pénalité C_{f2} au coût complet. S'il reste des clients non visités($L \neq \emptyset$), la procédure est répétée jusqu'à insérer tous les clients de L.

Algorithm 1 Algorithme d'évaluation d'une solution S

Input : L, la liste des clients non visités, $f(S)$ le coût simple de la solution
Output : $f_c(S)$, le coût complet
$f_c(S) := f(S) + C_f \times nb_{ve}$
while $L \neq \emptyset$ **do**
    Une tournée vide T_d est créée pour chaque jour d de l'horizon
    for all client $i \in L$ **do**
        Tester l'insertion de i dans toutes les tournées T_d telles que $d \in D(i)$,
        if $Insertion.faisable(i)$ **then**
            Choisir la tournée engendrant la moindre augmentation de coût et y insérer le client i
            Supprimer i de L
        end if
    end for
    for all $d \in H$ **do**
        if $Non.vide(T_d)$ **then**
            Calculer $TourCost(T_d)$ le coût (parcours, recharge) de T_d
            $f_c(S) := f_c(S) + TourCost(T_d) + C_{f2}$
        end if
    end for
end while
Retourner $f_c(S)$

3.4 Considération de la contrainte d'énergie ($repar(Tr')$)

Les contraintes classiques des problèmes de tournées de véhicules sont majoritairement strictes et à ressources limitées, sans parades possibles. Par exemple, si une tournée dépasse la durée maximale, il n'est pas possible de trouver une "parade" permettant l'augmentation de la ressource temps. De même, la capacité d'emport d'un véhicule ne peut être augmentée épisodiquement. A l'inverse, si la consommation énergétique d'une tournée dépasse l'énergie disponible dans un véhicule, il est possible de "réparer" la tournée en "injectant" de l'énergie par le biais d'une visite à une station de recharge pendant la tournée.

Il existe deux approches pour avoir des solutions vérifiant les contraintes d'énergie ; la première est de vérifier la faisabilité énergétique d'une solution une fois tous les clients visités et de tenter une réparation si besoin. La deuxième approche est de vérifier et réparer les contraintes d'énergie à chaque modification sur les différentes tournées lors de la construction de la solution. L'avantage de la première approche est de ne vérifier/réparer la solution qu'une seule fois, mais cette approche peut être coûteuse en termes de combinatoire (à quelle position insérer une station ?, combien de stations insérer ? ...) et l'échec d'une réparation fait perdre entièrement la solution construite. Cette approche peut être plus appropriée pour des instances très peu contraintes en termes de consommation d'énergie (flotte mixte, nombre de tournées minimes...).

Nous avons donc choisi la deuxième approche qui nous permet de vérifier progressivement la contrainte de consommation d'énergie. Nous allons donc définir des méthodes de vérifications/ réparations adaptées à cette approche. Nous nous positionnons dans la situation où une solution réalisable est modifiée pour ajouter la visite d'un nouveau client. Plus précisément, nous nous positionnons au niveau d'une tournée Tr vérifiant les différentes contraintes et où un nouveau client i a été inséré à une position donnée, cette tournée modifiée Tr' doit donc vérifier les

différentes contraintes. La contrainte de disponibilité d'un client se vérifie avant l'insertion d'un client, puisqu'on n'insère un client que dans ses jours de disponibilité. Il reste alors trois types de contraintes à vérifier sur la tournée pour chaque insertion :

- **Capacité du véhicule :** La somme des demandes des clients visités dans la tournée Tr et de la demande du client à insérer i ne doit pas dépasser la capacité du véhicule. Cette contrainte peut se vérifier avant même l'insertion du client i puisqu'elle ne dépend pas de la position d'insertion.
- **Durée maximale d'une tournée :** La durée de la tournée étant différente selon la position de l'insertion du client i, cette contrainte doit être vérifiée à chaque position d'insertion.
- **Consommation d'énergie :** En plus de devoir vérifier cette contrainte à chaque position d'insertion, la violation de cette contrainte peut être "réparée" en insérant, à titre d'exemple, une station de recharge. Il est donc nécessaire, en plus de vérifier si cette contrainte est respectée, de vérifier si la tournée peut être réparée et à quel coût.

Soit Tr' la tournée résultante de l'insertion du client i dans la tournée Tr à une position donnée. Supposons que cette tournée Tr' vérifie les deux premières contraintes mais viole la contrainte de consommation d'énergie. La fonction $repar(Tr')$ permet de réparer cette tournée. Si plusieurs réparations sont possibles, la fonction $repar(Tr')$ retourne la tournée réparée la moins coûteuse. Si aucune réparation n'est possible, la fonction retourne la tournée non réparée.

Si l'énergie résiduelle du véhicule ne suffit pas à visiter tous les clients et revenir au dépôt, la fonction $repar(Tr')$ procède en deux étapes, la première étape cherche à augmenter la quantité d'énergie rechargée dans des stations de recharge déjà visitées dans la tournée. Si cette étape suffit, la fonction retourne la tournée réparée. La solution d'augmenter les recharges existantes est intéressante puisqu'elle n'augmente pas le coût de la solution (prix de recharge forfaitaire). Si cette étape n'est pas suffisante, la fonction passe à la deuxième étape, où la tournée est réparée en insérant une station de recharge. Pour l'insertion de la station, nous choisissons la station de recharge la moins coûteuse qui permet de satisfaire la contrainte énergétique. Ces deux étapes sont faites tout en vérifiant que la contrainte de durée maximale n'est pas violée.

Avant de détailler les deux étapes, voici les notations utilisées.

Notions et notations Une tournée Tr est une suite de sommets (dépôt, client, borne). Le véhicule part avec un niveau de batterie $MaxEnergy$ et consomme l'énergie en parcourant la tournée. Nous autorisons par usage ce niveau à être négatif pour mieux visualiser les tournées infaisables. Dans la figure (3.1), nous donnons un exemple d'une tournée réalisable et dans la figure (3.2) la même tournée viole la contrainte d'énergie après l'ajout de la visite du client (5). Nous utilisons ces deux exemples pour mettre en évidence les notions et fonctions listées dans le tableau (3.4).

Dans les exemples 3.1 et 3.2, nous avons mis en avant les informations liées à la consommation d'énergie :

- La consommation d'énergie sur les arcs exprimée en % de la capacité de la batterie.
- Le niveau de la batterie à l'arrivée du véhicule à chaque sommet ($EnergyIn$) et au départ ($EnergyOut$)
- Pour les sommets stations (B_i), la différence $EnergyOut - EnergyIn$ correspond à la recharge exprimée en % de la capacité de la batterie.

Dans les deux exemples 3.1 et 3.2, nous n'avons pas dupliqué le dépôt en deux copies ($\bar{0}$ et $\underline{0}$) pour mieux visualiser la tournée. Cette duplication reste nécessaire pour différencier le début de la tournée de sa fin et traiter la tournée comme une séquence.

$\bar{0}$	La copie du dépôt représentant le départ d'une tournée.
$\underline{0}$	La copie du dépôt représentant la fin d'une tournée.
$MaxEnergy$	L'énergie maximale qui peut être chargée dans un véhicule.
$succ(k)$	Le sommet situé après k dans la tournée.
$prec(k)$	Le sommet situé avant k dans la tournée.
$PrecStation(k)$	Retourne la dernière station de recharge visitée avant un sommet k. Si aucune station n'est visitée avant ce sommet la fonction retourne $\bar{0}$.
$EnergyIn(k)$	Retourne la quantité d'énergie disponible dans le véhicule en arrivant au sommet k.
$EnergyOut(k)$	Retourne la quantité d'énergie disponible dans le véhicule en quittant le sommet k.
$Recharge(b)$	Retourne la quantité d'énergie rechargée à la station b.
$GetMinEnergy(Tr)$	Retourne le niveau de batterie le plus bas atteint :$\min_{k \in Tr} EnergyIn(k)$.
$FirstNegative(Tr)$	En parcourant Tr du début à la fin, retourne le premier sommet k, de sorte que $EnergyIn(k) \leq 0$ ou $EnergyOut(k) \leq 0$, sinon retourne ∞.
$UpdateCharging(Tr)$	Actualise les valeurs $EnergyIn(u)$ et $EnergyOut(u)$ pour chaque sommet dans Tr. Cette procédure est utilisée après une modification sur la tournée.
$TourCost(Tr)$	Coût de la tournée, entre coûts de parcours et coûts de recharge
$Demande(Tr)$	Somme des demandes des clients visités dans le tour.
$Temps(Tr)$	Durée de la tournée Tr.

TABLE 3.4 – Notations

La valeur de $MaxEnergy$ permet de respecter la capacité limitée de la batterie durant une recharge extérieure.

$PrecStation(k)$ remonte la séquence de la tournée à partir du sommet k et retourne la première station trouvée avant d'atteindre le dépôt de départ ; si aucune station n'est trouvée, la fonction retourne $\bar{0}$. Dans l'exemple (3.1), $PrecStation(4) = B_1$ et $PrecStation(2) = \bar{0}$. Dans 3.2, $PrecStation(\underline{0}) = B_1$ permet de savoir à quelle station augmenter la recharge, si cette station existe.

S'il ne visite pas de borne pendant la tournée, l'énergie d'un véhicule diminue et atteint son minimum en arrivant au dépôt. Il suffit dans ce cas, de vérifier que le niveau de la batterie est positif au dépôt pour qu'une tournée soit faisable en termes d'énergie. S'il visite au moins une borne, le véhicule peut arriver au dépôt avec un niveau de batterie positif sans garantir que la tournée est réalisable. Il est donc nécessaire de vérifier le niveau de la batterie à chaque arrivée à un sommet ($k \in Tr$), et vérifier que le minimum de ces valeurs $GetMinEnergy(Tr)$ est positif pour garantir qu'une tournée est réalisable. Dans l'exemple (3.1), $GetMinEnergy(Tr) = 0$, la tournée est donc réalisable. A l'inverse, dans 3.2 $GetMinEnergy(Tr) = -15\%$ et la tournée viole donc la contrainte d'énergie. De plus si la tournée est irréalisable, il est nécessaire d'injecter

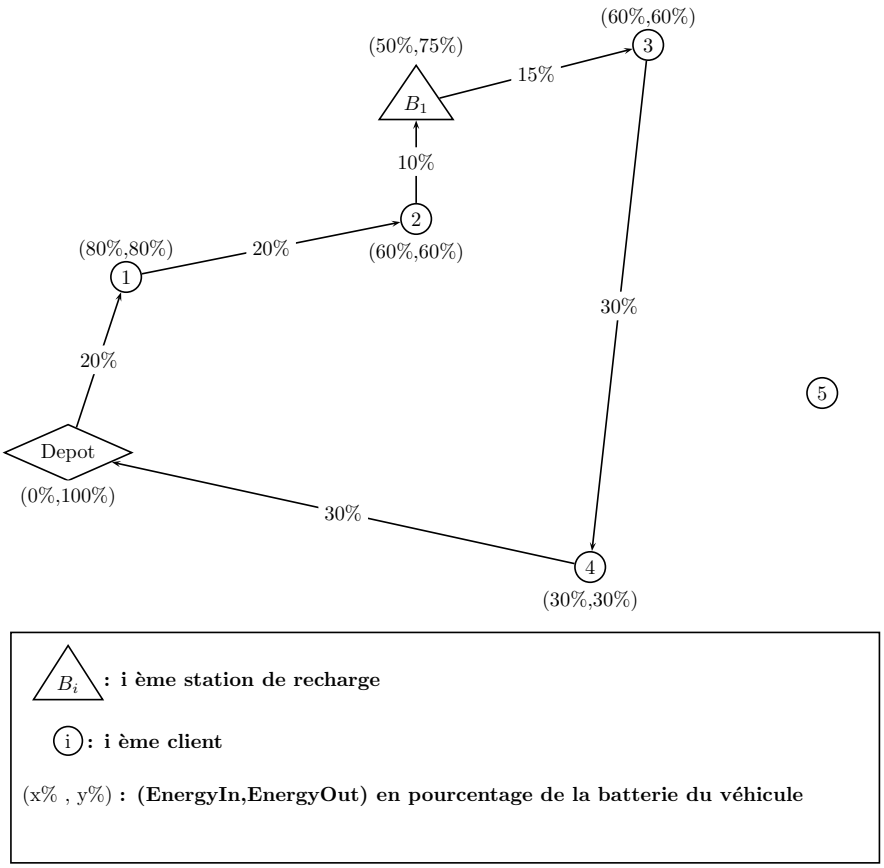


FIGURE 3.1 – Exemple d'une tournée réalisable

au minimum ($-GetMinEnergy(Tr)$) dans la tournée pour la réparer. Il suffirait donc dans 3.2 d'augmenter la recharge à la borne de 15% pour rendre la tournée réalisable. Si cette augmentation de recharge est faite, la fonction $UpdateCharging(Tr)$ actualise les valeurs de $EnergyIn$ et $EnergyOut$. Nous verrons dans la prochaine section, les conditions et la procédure pour réparer une tournée en augmentant les recharges des stations visitées.

3.4.1 Étape 1 : Augmentation de la recharge existante ($IncreaseCharging$)

Le VE ayant une batterie limitée, nous pouvons réduire le champ d'action de cette procédure sur une partie de Tr' sans perdre en efficacité. Soit $u^- = FirstNegativeNode(Tr')$ le premier sommet que le VE n'atteint pas ($EnergyIn(u^-) \leq 0$) , et soit $b^- = PrecStation(u^-)$ la station précédant ce sommet. Augmenter l'énergie des bornes visitées après u^- est évidemment inutile pour la réparation de la tournée. D'autre part, une augmentation de la recharge des bornes avant b^- revient exactement à la même chose qu'augmenter la quantité d'énergie rechargée dans b^-. Dans l'exemple 3.2, si une borne B_j était visitée avant B_1, l'augmentation de la recharge à cette borne (B_j) de x% reviendrait exactement à une augmentation de recharge de x% à B_1. De plus, si on augmente la recharge à cette borne B_j de 30% en pensant avoir injecté 30% dans la tournée, dans les faits seulement 25% ont été réellement injecté dû à la capacité limité de la batterie. Il

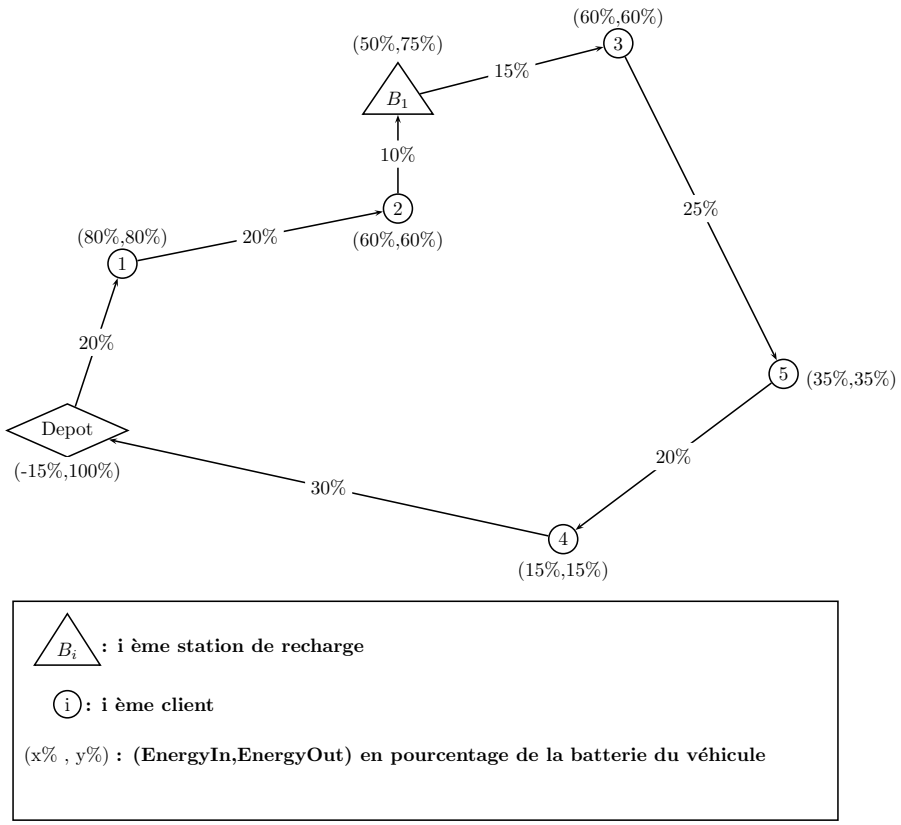


FIGURE 3.2 – Exemple d'une tournée violant la contrainte d'énergie

est donc suffisant de ne considérer une augmentation d'énergie que dans la dernière borne visitée avant le sommet négatif.

Dans l'exemple 3.2, nous avons $FirstNegativeNode(Tr') = \underline{0}$ et $PrecStation(\underline{0}) = B_1$. Il suffit donc de recharger $-GetMinEnergy(Tr) = 15\%$ pour réparer la tournée. Le véhicule partant avec un niveau 75% de la station B_1, il est donc possible de recharger jusqu'à 90% et réparer la tournée.

Pour réparer la tournée, il est nécessaire et suffisant d'augmenter la recharge dans b^- de $(-GetMinEnergy(Tr))$. Cette augmentation qui doit s'ajouter à la recharge déjà faite à la station, se restreint à la capacité de la batterie du véhicule et n'est donc pas toujours possible. Si la limite de la batterie du véhicule ne permet pas d'augmenter de $-GetMinEnergy(Tr)$, il reste intéressant d'augmenter la recharge jusqu'à la limite de la batterie pour réduire l'énergie à injecter dans les étapes suivantes. De plus, cette augmentation jusqu'à la limite de la batterie peut permettre d'atteindre une autre station de la tournée, dont la recharge sera augmentée à son tour. D'un autre côté, le temps nécessaire à toute recharge supplémentaire ne doit pas entraîner un dépassement du T_{max}.

Pour cette étape nous avons défini une fonction $IncreaseCharging(Tr')$ qui augmente efficacement les recharges dans la tournée. Les détails de la fonction sont explicités dans l'Algorithme 2.

Le but de la fonction $IncreaseCharging(Tr')$ est d'augmenter efficacement les recharges

Algorithm 2 *IncreaseCharging*(Tr')

1: **Input :**Tournée Tr'
2: **Output :** Tr' où la recharge est augmentée
3: Calculer t' le temps nécessaire pour recharger $-GetMinEnergy(Tr)$
4: **if** t' implique une violation de la contrainte de durée maximale **then** :
5: retourner Tr' ▷ Tournée non réparée
6: **end if**
7: $UpdateCharging(Tr')$
8: $u^- := firstNodeNegativeNode(Tr')$, $b^- = PrecStation(u^-)$
9: **if** $(MaxEnergy = EnergyOut(b^-))$ **then** ▷ La borne recharge déjà au maximum
10: Retourner Tr' ▷ Tournée non réparée
11: **end if**
12: **if** $(MaxEnergy - EnergyOut(b^-))) \geq GetMinEnergy(Tr')$ **then**
13: Augmenter la recharge de b^- de $(GetMinEnergy(Tr'))$, ▷ Tournée réparée
14: $UpdateCharging(Tr')$
15: Retourner Tr' ▷ Tournée réparée
16: **else**
17: Augmenter la recharge de b^- de $(MaxEnergy - EnergyOut(b^-))$, ▷ Augmentation maximale
18: **Allez à** 7
19: **end if**

de $-GetMinEnergy(Tr)$. Pour cela, il faut premièrement vérifier que cette augmentation est possible en termes de temps et donc que le temps nécessaire à cette recharge supplémentaire n'induit pas un dépassement de la durée de la recharge au delà de T_{max} (Algorithme 2, ligne 3) et donc une violation de la contrainte de durée maximale. Si cette augmentation est réalisable en termes de temps, nous allons repérer la station de recharge b^- où une augmentation est efficace (Algorithme 2, ligne 8), si la recharge est déjà maximale à cette station, il est donc impossible d'augmenter la recharge et la tournée ne peut être réparée par cette fonction. Si la batterie du véhicule permet de recharger $-GetMinEnergy(Tr)$ en plus de sa recharge actuelle (Algorithme 2, ligne 12), on applique cette augmentation et la tournée est réparée. La tournée de l'exemple 3.2 est réparée et donne la tournée dans 3.3.

Si la capacité de la batterie ne permet pas d'augmenter la recharge de $-GetMinEnergy(Tr)$ (Algorithme 2, ligne 16), la fonction augmente la recharge au maximum pour réduire l'énergie à injecter. C'est le cas dans l'exemple 3.4 où il serait nécessaire d'augmenter la recharge dans B_1 de $-GetMinEnergy(Tr) = 40\%$. En effet, la recharge dans B_1 allant jusqu'à 90%, il est impossible de l'augmenter de 40% et donc réparer la tournée. L'algorithme augmente la recharge au maximum, ce qui permet de diminuer $-GetMinEnergy(Tr)$ à 30% et de réduire la difficulté de la réparation dans les étapes suivantes (3.5).

Une fois cette augmentation faite et la tournée actualisée, il est possible que les sommets u^- et b^- aient changé. Ce potentiel changement nécessite de relancer la procédure avec la tournée modifiée. En effet, si b^- est une autre station de la tournée, elle peut permettre d'encore réduire $-GetMinEnergy(Tr)$ où même de réparer la tournée. C'est pour cela que l'algorithme relance la procédure à chaque augmentation non réparatrice (Algorithme 2, ligne 18), et ne s'arrête que si le véhicule recharge au maximum à la station b^-.

Si la procédure $IncreaseCharging(Tr')$ retourne une tournée non réparée comme dans l'exemple 3.5, il est nécessaire de passer à l'étape 2 où une insertion de station sera testée

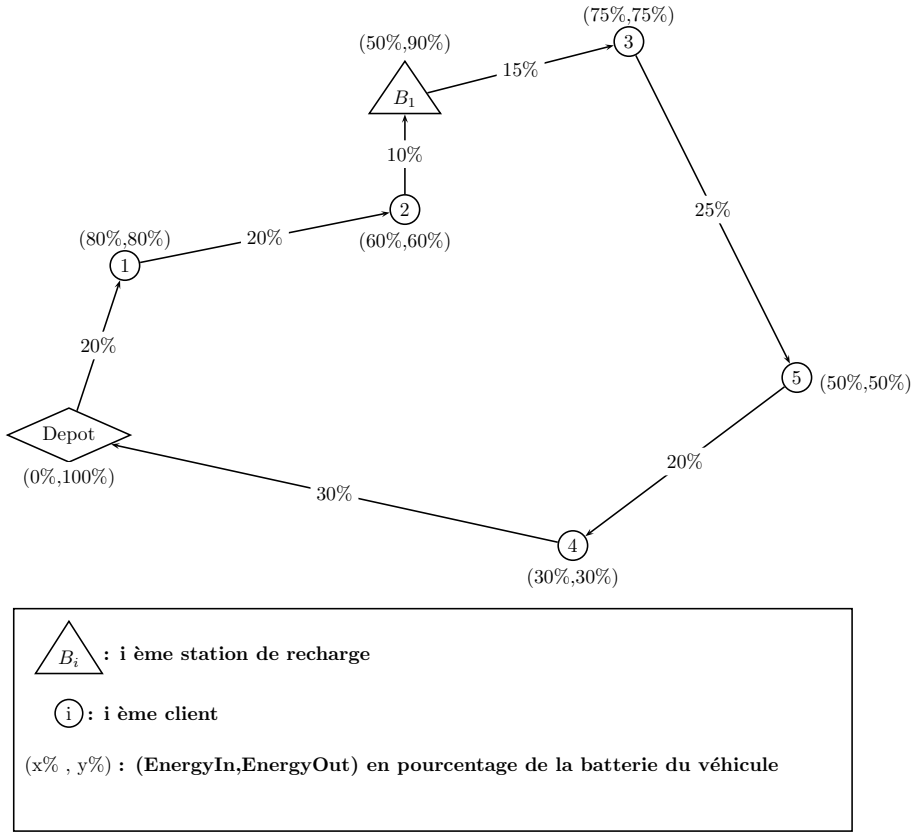


FIGURE 3.3 – Exemple d'une tournée réparée en augmentant la recharge

pour la réparer.

3.4.2 Étape 2 : Insertion d'une station ($InsertChargingStation$)

De même que dans l'étape précédente, le traitement se fait entre le premier sommet non atteint u^- et la station précédente b^-. La fonction $InsertChargingStation(Tr')$ (Algorithme 3) insère la borne réparant la tournée à moindre coût. Si aucune réparation n'est possible la fonction retourne la tournée non réparée.

L'Algorithme 3 teste l'insertion d'une borne à chaque position entre b^-(inclut) et u^-(non inclut), le choix de la borne à insérer à chaque position est fait selon le gain d'énergie potentielle de chacune des bornes tout en vérifiant que leurs insertions ne violent pas la contrainte de durée maximale d'une tournée.

Dans la figure 3.6, nous avons repris la tournée de la figure 3.5 et mis en avant toutes les stations atteignables $\{B_2, B_3, B_4\}$ à partir des sommets successifs $\{B_1, ..., B_5\}$. Pour éviter toute confusion dans l'exemple, nous n'avons pas considéré la revisite de B_1 mais cette possibilité est valable dans l'algorithme.

À une position donnée k, l'algorithme sélectionne les stations atteignables et les stocke dans un ensemble B_k (Algorithme 3, ligne 5). Entre ces stations atteignables, celles qui permettent

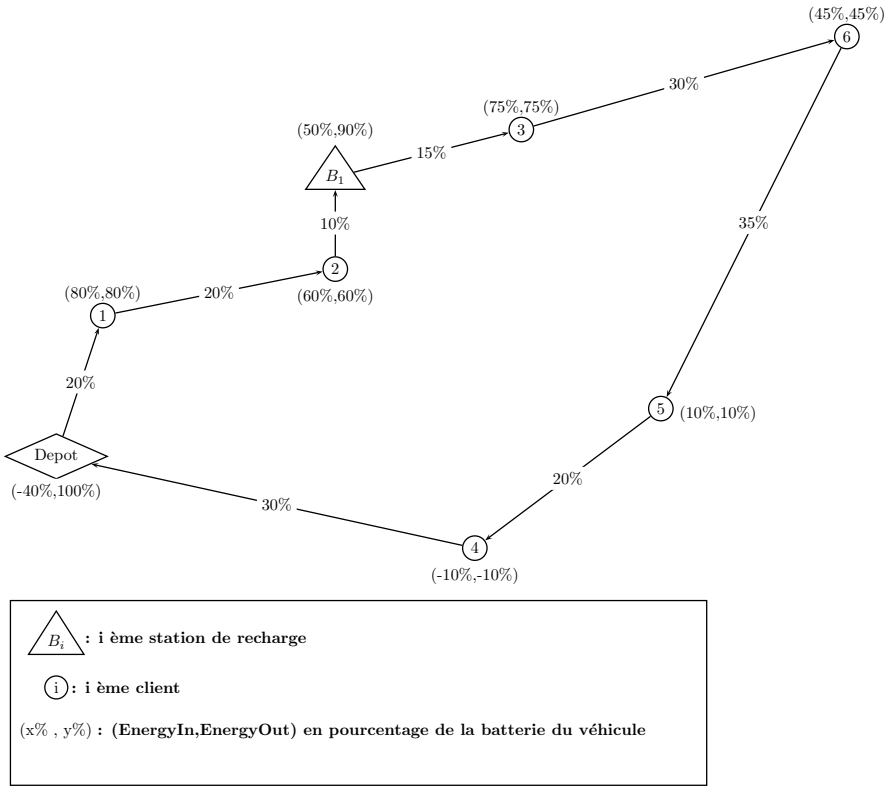


FIGURE 3.4 – Exemple d'une tournée violant la contrainte d'énergie

la réparation doivent vérifier deux critères : le premier est d'injecter assez d'énergie dans la tournée, et le deuxième est celui de ne pas causer un dépassement de la durée maximale. Dans notre algorithme, nous avons choisi de classer les stations selon le taux d'énergie injecté dans la tournée, si la première station ne vérifie pas le premier critère, aucune autre station ne peut le faire. Si elle vérifie le premier critère mais cause un dépassement du T_{max}, nous considérons la prochaine station dans la liste classée, jusqu'à obtenir une station vérifiant les deux critères (réparation possible) où une station violant le premier critère (pas de réparation possible à cette position). Au delà de trouver la station qui répare si elle existe, cette procédure avantage les stations permettant d'injecter le plus d'énergie et donc de faciliter, en termes d'énergie, l'insertion future de nouveaux clients dans la tournée.

Pour vérifier si une station b_k à une position k répare la tournée en termes d'énergie, l'algorithme calcule la quantité d'énergie e_k à recharger et l'état de la batterie après cette recharge $EnergyOut(b_k)$. Si la recharge nécessaire excède la capacité de la batterie ($EnergyOut(b_k) \geq MaxEnergy$), la réparation n'est pas possible à cette position (Algorithme 3, ligne 12). Si le temps nécessaire pour atteindre la station et recharger fait dépasser la durée de la tournée T_{max} (Algorithme 3, ligne 17), l'algorithme supprime b_k de B_k et teste la station suivante de B_k en termes d'énergie injectée dans la tournée. Si une station réparant la tournée est trouvée, l'algorithme calcule son coût (le coût de recharge et le coût de parcours pour l'atteindre). Entre les stations réparant la tournée, l'algorithme choisit la station la moins coûteuse, si elle existe, et l'insère à la position correspondante.

Dans le tableau 3.5, nous avons itéré l'algorithme sur l'exemple 3.6. A partir de B_1, il est donc

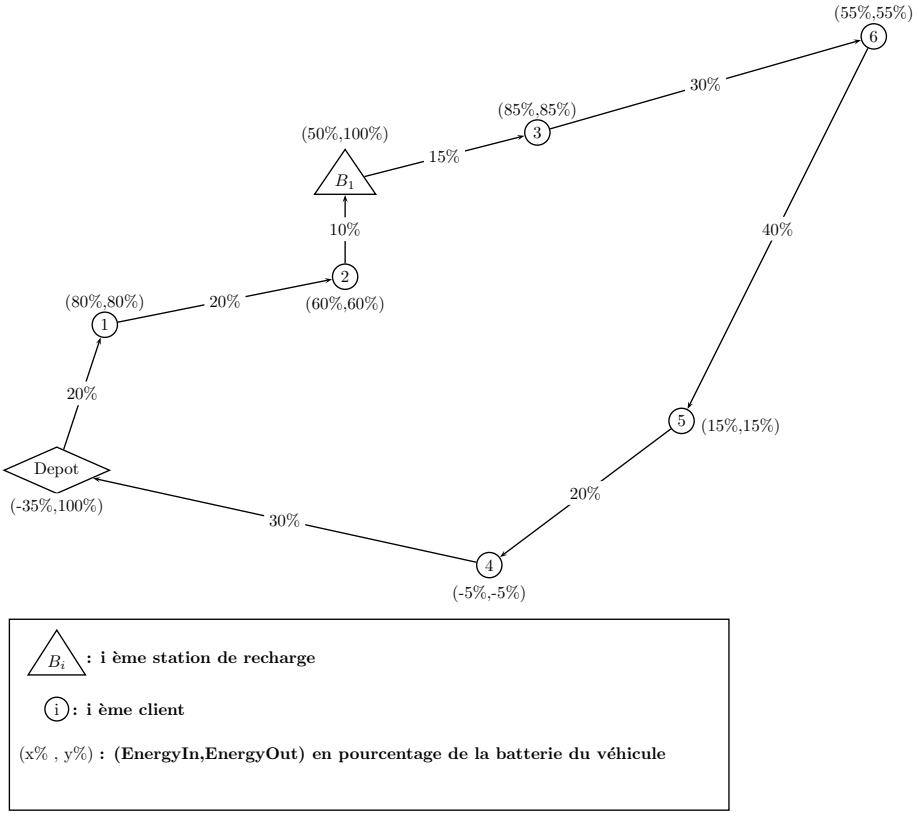


FIGURE 3.5 – Exemple d'une augmentation de recharge non suffisante

possible d'atteindre B_2 et B_3. Le choix se fait sur B_2 puisque cette station permet d'arriver au sommet $succ(B_1) = 3$ avec un niveau de batterie de 90% (100%−10%) si on recharge au maximum à B_2, en comparaison avec B_3 qui ferait au maximum arriver le véhicule avec un recharge de 85%. En calculant le $EnergyOut(B_2) = 130\%$ nécessaire si B_2 est inséré entre B_1 et 3, nous pouvons conclure que la réparation n'est pas possible à cette position ($130\% > MaxEnergy = 100\%$). La même conclusion est constatée pour l'insertion d'une station après le client 3. Pour $k = 6$, l'insertion de la station B_4 après le client 6 permettrait d'injecter assez d'énergie pour réparer la tournée ($70\% < 100\%$) mais donnerait une tournée dont la durée dépasserait T_{max}. La station suivante B_3 est donc testée à la même position. L'insertion de B_3 après le client 6 vérifie les deux critères (énergie et temps), son coût est donc calculé. A partir du client 5 ($k = 5$), aucune station n'est atteignable. La seule solution de réparation est donc l'insertion de B_3 entre les clients 6 et 5, avec une recharge allant jusqu'à 75%.

3.5 Heuristique de meilleure insertion (BIH)

L'heuristique de meilleure insertion (BIH) construit en parallèle les tournées pour chaque jour. Globalement, m tournées, initialement vides, sont définies pour chaque jour. À chaque itération et pour chaque jour, nous considérons l'insertion dans toutes les tournées disponibles sans dépasser m tours par jour. Pour chaque client i et pour chacun de ses jours de disponibilité $d \in D(i)$, l'algorithme simule l'insertion de i dans toutes les positions possibles de tous les tours de

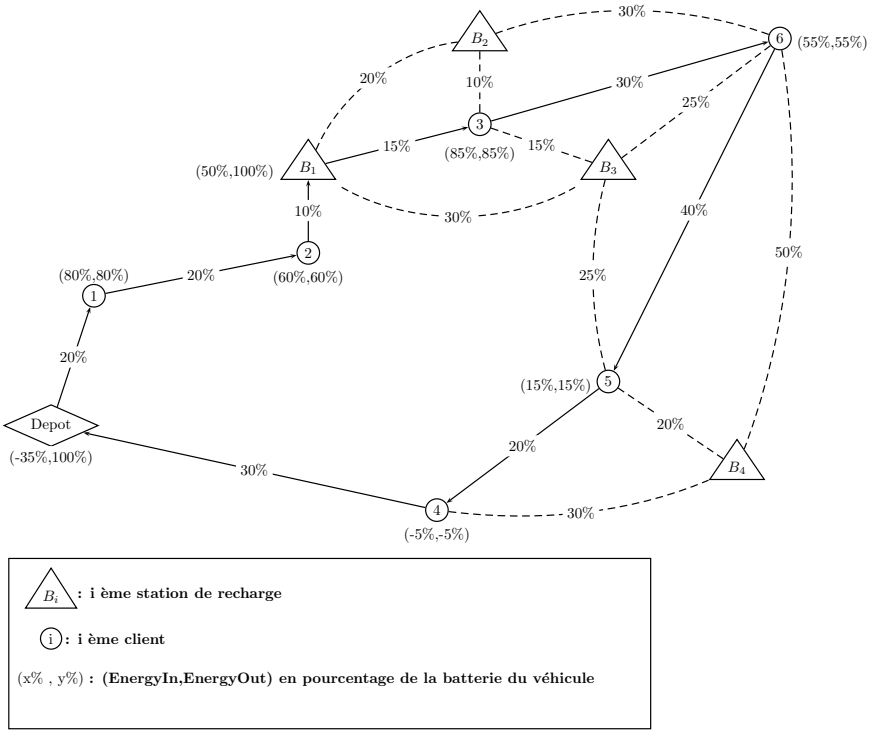


FIGURE 3.6 – Stations atteignables pour réparation de la tournée

	B_k									
Sommet k	B_2	B_3	B_4	b_k	δ	e_k	$EnergyOut(b_k)$	Réparation	T_{max}	$Cost_k$
B_1	✓	✓	×	B_2	15%	50%	130%	×	-	∞
3	✓	✓	×	B_3	10%	45%	115%	×	-	∞
6	✓	✓	✓	B_4	30%	65%	70%	✓	×	-
6	✓	✓	×	B_3	10%	45%	75%	✓	✓	300
5	×	×	×	×	×	×	×	×	-	∞

TABLE 3.5 – Déroulé de l'algorithme sur l'exemple 3.6

la journée considérée d. La variation du coût total est calculée pour chaque insertion vérifiant les différentes contraintes. Le client (et éventuellement la station de recharge) avec le coût d'insertion minimum est inséré à la fin de chaque itération à sa meilleure position. L'heuristique s'arrête si tous les clients sont insérés ou si aucune insertion n'est possible.

3.5.1 Test d'insertion *SimulInsert*

Afin de simuler l'insertion d'un client i dans une tournée Tr nous utilisons la fonction $SimulInsert(Tr, i)$. La fonction permet de tester l'insertion du client i dans toutes les positions de la tournée Tr en vérifiant les différentes contraintes et retourne le coût de la meilleure insertion respectant les contraintes et la tournée correspondante. Si l'insertion n'est pas réalisable la fonction retourne un coût infini et une tournée vide.

L'Algorithme 4 commence par vérifier s'il reste un véhicule avec assez de capacité pour répondre à la demande du client i (Algorithme 4, ligne 6) sinon l'algorithme s'arrête et retourne

Algorithm 3 $InsertChargingStation(Tr')$

1: **Input :** Tournée Tr'
2: **Output :** Tr'
3: $u^- := firstNodeNegativeNode(Tr')$, $b^- = PrecStation(u^-)$
4: **for all** sommet k entre b^- et $prec(u^-)$ **do**
5: $B'_k = \{b \in B | (EnergyOut(k) - e_{k,b}) > 0\}$, ▷ Les bornes atteignables
6: $cost_k = \infty$
7: **if** $B'_k \neq \emptyset$ **then**
8: $b_k = \arg\min_{b \in B'} .\{e_{b,succ(k)} - EnergyOut(succ(k))\}$ ▷ borne injectant le plus d'énergie
9: $\delta = e_{k,b_k} + e_{b_k,succ(k)} - e_{k,succ(k)}$ ▷ Énergie consommée pour atteindre la borne
10: $z_k = -GetMinEnergy(Tr') + \delta$ ▷ Énergie à recharger à la borne pour réparer
11: $EnergyOut(b_k) = z_k + EnergyOut(k) - e_{k,b_k}$ ▷ $EnergyOut(b_k)$ éventuel et nécessaire pour réparer la tournée
12: **if** $EnergyOut(b_k) \geq MaxEnergy$ **then**
13: Pas de réparation possible à cette position.
14: $cost_k = \infty$, Aller à 23
15: **end if**
16: Soit t' le temps supplémentaire nécessaire pour visiter et recharger à b_k
17: **if** t' engendre dépassement du T_{max} **then**
18: $B'_k = B'_k - \{b'\}$, ▷ Permet de tester une autre borne moins chronophage
19: Aller à l'étape 7
20: **end if**
21: Calculer le coût d'insertion $cost_k$
22: **end if**
23: **end for**
24: **if** $\min\{Cost_k\} < \infty$ **then** ▷ Réparation possible
25: $k^* = \arg\min\{Cost_k\}$
26: Insérer la borne b^*_k à la position k^* avec une recharge z_{k^*}
27: **end if**
28: Retourner Tr'

un coût infini. Une fois la contrainte de capacité vérifiée, l'algorithme teste l'insertion du client i dans chaque position de la tournée Tr. Pour chaque position k, une tournée Tr' copie de Tr est créée, puis le sommet i est inséré dans Tr' après le sommet k (Algorithme 4, ligne 13). Si la durée de la tournée dépasse T_{max}, l'algorithme passe à la position suivante. Il reste donc à vérifier la contrainte d'énergie (Algorithme 4, ligne 15), tenter la réparation sinon (Algorithme 4, ligne 16) puis comparer la tournée résultante à la meilleure enregistrée. Si la tournée vérifie la contrainte d'énergie (avant ou après réparation) et son coût est inférieur au meilleur coût enregistré, le coût et la tournée sont enregistrés.

A la fin de l'algorithme, la tournée réalisable contenant i la moins coûteuse est retournée, en plus de son coût.

3.5.2 L'algorithme BIH

Pour construire la solution, l'algorithme BIH part de tournées vides et insert les clients itérativement, en choisissant à chaque itération le client engendrant la moindre augmentation du

Algorithm 4 $SimulInsert(Tr, i)$: Test d'insertion du client i dans la tournée Tr

1: **Input :** Tournée Tr, client i
2: **Output :** $SimulInsert(Tr, i).CostMin$: Coût d'insertion minimum
3: $SimulInsert(Tr, i).Tour$: la tournée correspondante
4: $SimulInsert(Tr, i).CostMin = \infty$
5: $SimulInsert(Tr, i).Tour = \emptyset$
6: **if** $Demande(Tr) + q_i > Q$ **then**
7: Client i ne peut être inséré dans la tournée
8: Retourne $SimulInsert(Tr, i).CostMin$ et $SimulInsert(Tr, i).Tour = Tr$
9: **end if**
10: $k = \bar{0}$
11: **while** $k \neq \underline{0}$ **do**
12: $Tr' = Tr$
13: Insérer i après le sommet k dans Tr'
14: **if** $Temps(Tr') \leq T_{max}$ **then**
15: **if** $GetMinEnergy(Tr') < 0$ **then**
16: $repar(Tr')$
17: **end if**
18: **if** $(GetMinEnergy(Tr') \geq 0 \quad \& \ TourCost(Tr') < CostMin)$ **then**
19: $SimulInsert(Tr, i).CostMin = TourCost(Tr')$
20: $SimulInsert(Tr, i).Tour = Tr'$
21: **end if**
22: **end if**
23: **end while**
24: Retourne $SimulInsert(Tr, i).CostMin$ et $SimulInsert(Tr, i).Tour = Tr$

coût de la solution.

L'objectif de l'heuristique BIH est donc de choisir la meilleure insertion à chaque itération jusqu'à insérer tous les clients. A chaque itération de l'heuristique, l'insertion de chaque client non traité est testée dans chacun de ses jours de disponibilités et dans chaque tournée de ces jours. Entre toutes les insertions faisables des différents clients, celle provoquant le minimum d'augmentation de coût est appliquée.

L'Algorithme 5 initialise la solution S à m tournées vides pour chaque jour de l'horizon (ligne 6). L'ensemble C' contient les clients à insérer dans la solution, puis une fois un client inséré dans la solution, celui ci est supprimé de C' et les tests d'insertion des clients restants sont relancés. Il n'est pas nécessaire de refaire tous les tests d'insertion après chaque insertion, puisqu'il suffit de refaire seulement ceux concernant la tournée modifiée à l'itération précédente. Pour des raisons de simplification et de compréhension, nous avons relancé tous les tests d'insertion dans l'Algorithme 5. L'algorithme s'arrête si tous les clients sont visités ou si aucun client dans C' ne peut être inséré dans la solution. $CostMin$ est initialisé à ∞ à chaque itération, puis prend la valeur de chaque $SimulInsert(T_j^d, i).Cost$ lui étant inférieur strictement. $SimulInsert(T_j^d, i).Cost = \infty$ signifie que l'insertion n'est pas faisable. Le coût de l'insertion ne suffit pas à identifier le client à insérer ni la tournée de son insertion, l'algorithme sauvegarde donc le client, le jour et le véhicule concernés. D'autre part, pour éviter de relancer $SimulInsert$ une fois l'insertion identifiée, la meilleure tournée $BestTour$ est sauvegardée.

Si après avoir testé l'insertion de tous les clients dans C', aucune insertion n'est faisable ($CostMin = \infty$), l'algorithme supprime les clients dans C' et retourne la solution obtenue même

Algorithm 5 Algorithme BIH

Input : C : ensemble des clients, m : le nombre de véhicules disponibles
Output : La solution S
T_j^d : tournée du véhicule j le jour d (vide initialement).
$BestTour$ le meilleur tour à chaque itération
i_{best}, d_{best}, j_{best} meilleurs client, jour, tournée à chaque itération
Initialiser $S := \cup_{j\in\{1,..,m\}}^{d\in H} T_j^d$
$C' := C$ Les clients à visiter
$Stop := false$
while $C' \neq \emptyset$ & $Stop = false$ **do**
    $CostMin := \infty$
    for all $(i \in C')$ **do**
        for all $d \in D(i)$ **do**
            for all $j \in \{1, .., m\})$ **do**
                if $(SimulInsert(T_j^d, i).CostMin < CostMin)$ **then**
                $CostMin := SimulInsert(T_j^d, i).CostMin$
                $BestTour := SimulInsert(T_j^d, i).Tour$
                $i_{best} := i$, $d_{best} := d$, $j_{best} := j$
                end if
            end for
        end for
    end for
    if $CostMin < \infty$ **then**
        $T_{j_{best}}^{d_{best}} := BestTour$
        Supprimer le client i_{best} de C'
    else ▷ Plus d'insertion possible
        $Stop = true$
    end if
end while
Retourner S

si elle est incomplète.

3.6 Première heuristique de clustering CLH1

Cette heuristique s'inspire des heuristiques de type "cluster first, route second" où la résolution se fait en deux étapes, la première étape consiste à regrouper les clients par cluster (un cluster par véhicule), et la deuxième consiste à construire une tournée dans chaque cluster. Autrement dit, on utilise le principe de "diviser pour mieux règner" en traitant l'aspect affectation du problème puis l'aspect construction des tournées, le tout en respectant les contraintes liées aux véhicules électriques.

L'algorithme CLH1 fonctionne en quatre étapes. La première étape vise à créer m clusters initiaux de clients pour chaque jour de l'horizon, un pour chaque véhicule disponible. Dans la deuxième étape, pour chaque jour d, l'algorithme CLH1 essaie de répartir le nombre maximum de clients non affectés dans les m clusters disponibles du jour d, sans insérer de station de recharge, et en utilisant comme critère l'équilibrage de la distance dans tous les clusters du jour

d. À la troisième étape, les clients qui n'ont pas été insérés à l'étape 2 en raison des contraintes énergétiques liées à la batterie du véhicule seront pris en compte. Dans cette étape, l'objectif d'insertion est la minimisation de la consommation d'énergie supplémentaire pour chaque cluster. Enfin, dans la quatrième étape, une heuristique du problème du voyageur de commerce est utilisée pour construire une tournée réalisable par cluster par jour.

Pour illustrer l'heuristique, nous appliquerons successivement les étapes sur l'exemple graphique 3.7 d'une instance sur deux jours et avec trois véhicules. Dans l'exemple, nous avons trois types de clients, clients pouvant être visités le premier jour seulement, ceux pouvant être visités le deuxième jour, et les clients non exclusifs.

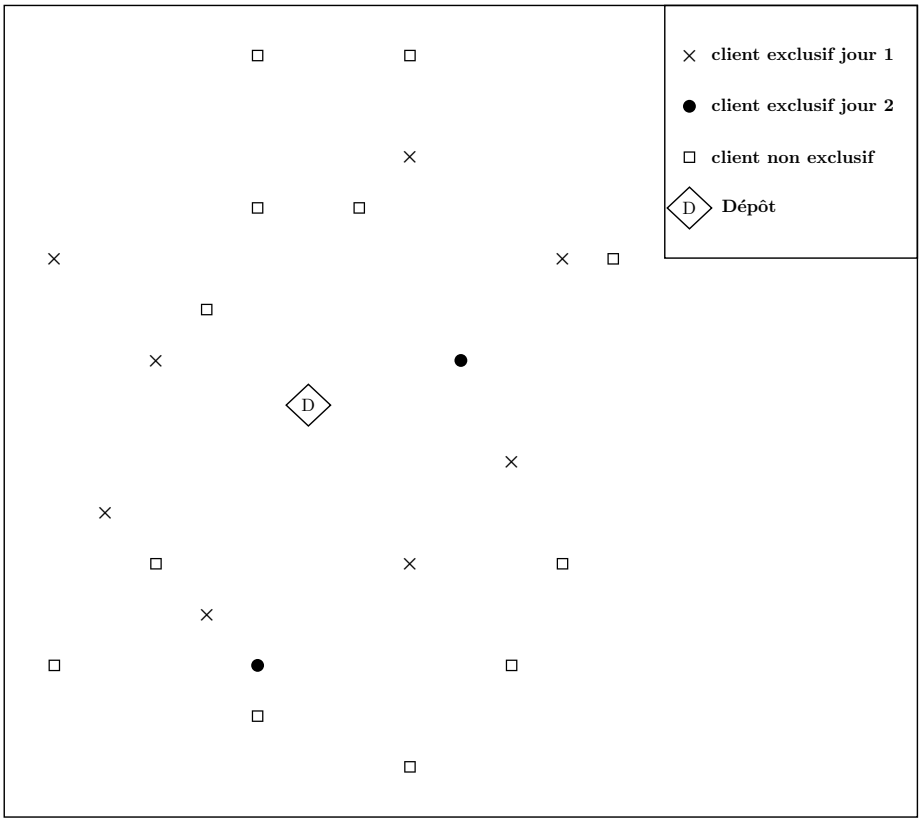


FIGURE 3.7 – Exemple pour application de l'heuristique CLH1 avec $m = 3$

Avant de détailler toutes les étapes de l'algorithme CLH1, nous précisons les notations. Pour chaque cluster cl_d, nous calculons une estimation de :

— $Distance(cl_d)$: est une estimation de la durée d'une tournée à partir du dépôt, visitant tous les clients dans cl_d et se terminant au dépôt. Il est possible d'obtenir la durée réelle de cette tournée mais cela nécessiterait la construction de la tournée ce qui est coûteux et de plus à l'encontre du principe de l'heuristique qui doit ignorer la construction des tournées dans les premières étapes. Nous avons donc choisi d'utiliser des estimations simples et de complexité faible pour avoir une heuristique rapide, sachant qu'une fonction d'estimation plus rigoureuse ne garantirait pas pour autant de meilleurs résultats à la construction des tournées. La distance est donc estimée selon (3.4) (3.5) (3.6).

$$Distance(cl_d) = \min(Distance_1(cl_d), Distance_2(cl_d)) \tag{3.4}$$

— $Distance_1(cl_d)$: est une surestimation calculée selon l'arc le plus coûteux dans cl_d. En multipliant ce coût par la taille du cluster, on obtient une borne max de distance de la tournée.

$$Distance_1(cl_d) = |cl_d - 1| \times \max_{i,j \in cl_d} d_{i,j} \tag{3.5}$$

— $Distance_2(cl_d)$ utilise une approximation basée sur la distance de chaque client au centre du cluster. En utilisant la propriété des distances euclidiennes, on obtient une nouvelle borne supérieure du coût de la tournée.

$$Distance_2(cl_d) = 2 \times \sum_{i \in cl_d} d_{center(cl_d),i} \tag{3.6}$$

— $Energy(cl_d)$: Estimation de la consommation d'énergie du véhicule pour réaliser la tournée, calculée à partir de la distance parcourue :

$$Energy(cl_d) = r \times Distance(cl_d)$$

— $Time_1(cl_d)$ (respectivement $Time_2(cl_d)$) est calculé comme $Distance_1(cl_d)$(respectivement $Distance_2(cl_d)$) en remplaçant $d_{i,j}$ par $t_{i,j}$.
— $Time(cl_d)$: Estimation du temps nécessaire pour effectuer une tournée à partir du dépôt :

$$Time(cl_d) = \min(Time_1(cl_d), Time_2(cl_d))$$

— $Load(cl_d)$: La quantité de produits nécessaires pour répondre à la demande de l'ensemble des clients affectés au cluster cl_d :

$$Load(cl_d) = \sum_{i \in cl_d} q_i$$

Une fois ces estimations définies, nous pouvons donc vérifier la faisabilité d'un cluster, c'est-à-dire vérifier s'il est transformable en une tournée réalisable. Pour cela l'algorithme CLH1 utilise les trois fonctions suivantes :

— $CheckEnergyFeasibility(cl_d)$: Cette fonction vérifie la faisabilité énergétique sans recharge de la tournée qui sera obtenue en cl_d et retourne $Vrai$ si $Energy(cl_d) \leq E$.
— $CheckConstraintsFeasibility(cl_d)$: Cette fonction vérifie que les demandes des clients appartenant au cluster cl_d ne dépassent pas la capacité du véhicule et que la durée maximale d'une tournée n'est pas dépassée. Elle retourne $Vrai$ si $Load(cl_d) \leq Q$ et $Time(cl_d) \leq T_{max}$.
— $CheckFeasibility(cl_d)$: Cette fonction réuni les deux précédentes fonctions, elle est égale à $CheckConstraintsFeasibility(cl_d)$ ET $CheckEnergyFeasibility(cl_d)$

Étape 1. Initialisation du cluster

L'initialisation du cluster commence par affecter tous les clients ayant seulement un jour de visite autorisé ($\forall i \in V$, avec $|D(i)| = 1$) qu'on nommera "clients exclusifs". Le choix est fait sur ces clients car ils n'y a pas de décision à faire sur leurs jours d'affectation et donc pas de regrets a posteriori. De plus, ils sont les plus contraints en terme de planification et donc les plus difficiles à insérer s'ils sont considérés plus tard.

Soit L_d la liste des clients du jour d, et $List_d = \{i \in V, |D(i)| = 1 \text{ et } D(i) = \{d\}\}$ la liste des clients exclusifs pour le jour d. Soit $\gamma_d = |List_d|$ le nombre de clients exclusifs du jour d. L'algorithme d'initialisation des cluster pour un jour d est présenté dans l'Algorithme 6.

Algorithm 6 Initialisation des clusters pour un jour d

1: **Input :** $List_d$: la liste des clients exclusifs du jour d. m : le nombre de véhicules
2: **Output :** Cl_d : ensemble des m clusters
3: **for all** client $i \in List_d$ **do**
4: Créer un cluster $\{i\}$, l'ajouter à l'ensemble Cl_d
5: **end for**
6: $Nbcluster_d := |Cl_d|$
7: **if** $(Nbcluster_d \leq m)$ **then**
8: Créer $(m - Nbcluster_d)$ clusters vides et les insérer dans Cl_d
9: Retourner Cl_d.
10: **end if**
11: $Pairs$: ensemble de paires fusionables de clusters
12: **while** $(Nbcluster_d > m)$ **do**
13: $Pairs = \emptyset$ ▷ Ensemble de paires de clusters fusionnables
14: **for all** $(a, b) \in Cl_d$ **do** ▷ a, b : deux clusters
15: **if** $CheckConstraintsFeasibility(a \cup b)$ **then** ▷ $a \cup b$: cluster fusion de a et b
16: Insérer la paire (a, b) dans $Pairs$
17: **end if**
18: **end for**
19: **if** $(Pairs \neq \emptyset)$ **then**
20: $(a^*, b^*) = \min\{Energy(a \cup b) \quad | \quad \forall (a, b) \in Pairs\}$
21: Créer le cluster $c = a^* \cup b^*$, ajouter c à Cl_d
22: Supprimer $a*$ et $b*$ de Cl_d
23: $Nbcluster_d := Nbcluster_d - 1$.
24: **else**
25: **while** $(Nbcluster_d > m)$ **do**
26: $a^* = \arg\max_{a \in Cl_d}\{(Energy(a)/|a|)\}$
27: Supprimer a^* de Cl_d, $Nbcluster_d := Nbcluster_d - 1$
28: **end while**
29: **end if**
30: **end while**
31: Retourner Cl_d.

L'Algorithme 6 reçoit en entrée tous les clients exclusifs d'un jour d et retourne Cl_d, ensemble contenant exactement m clusters. Pour chaque client exclusif, un cluster est créé le contenant (ligne 3). Selon le nombre de clusters résultants, nombre égale au nombre de clients exclusifs γ_d, l'algorithme procède comme suit :

— Cas $\gamma_d = m$: Nous avons créé autant de clusters un jour d que de véhicules, chaque cluster contenant un seul client exclusif. L'ensemble Cl_d de ces clusters est retourné.
— Cas $\gamma_d < m$ (ligne 7) : Nous avons plus de véhicules que de clusters créés. Des clusters vides viennent compléter le nombre de clusters nécessaires (ligne 8) .
— Cas $\gamma_d > m$: Dans ce cas, l'algorithme agit en deux étapes :

1. **Fusion de deux clusters** (ligne 14) : l'algorithme cherche les paires de clusters $Pairs$ pouvant être fusionnées en un seul cluster vérifiant les différentes contraintes. Parmi ces paires, celle dont la fusion donne le cluster consommant le moins d'énergie est fusionné en un seul cluster. Cette étape est répétée jusqu'à obtenir le nombre de clusters nécessaires m. Si aucune fusion n'est possible ($(Pairs \neq \emptyset)$), l'algorithme

passe à l'étape suivante :

2. **Supression des clusters** (ligne 24) : Puisque aucune fusion n'est possible, certains clusters doivent être supprimés. Les clients les composant seront donc non affectés et traités à l'étape suivante. Entre les clusters existants, ceux ayant une consommation d'énergie par client la plus haute seront supprimés.

Pendant l'étape de fusion des clusters, nous vérifions pour chaque fusion éventuelle que les contraintes sont respectées sur le cluster résultant (Algorithme 6, ligne 15). La contrainte d'énergie pouvant être levée par une visite à une station de recharge, nous considérons l'énergie consommée comme critère de sélection et non comme contrainte. Donc entre toutes les fusions faisables en termes de contraintes, nous choisissons celle qui engendre le minimum d'énergie (Algorithme 6, ligne 20). Dans le cas où aucune fusion n'est faisable, l'algorithme détruit les clusters engendrant le plus de consommation d'énergie par client jusqu'à obtenir le nombre nécessaire de clusters (Algorithme 6, ligne 26). Les clients exclusifs non affectés à un cluster seront traités en priorité lors de la prochaine étape.

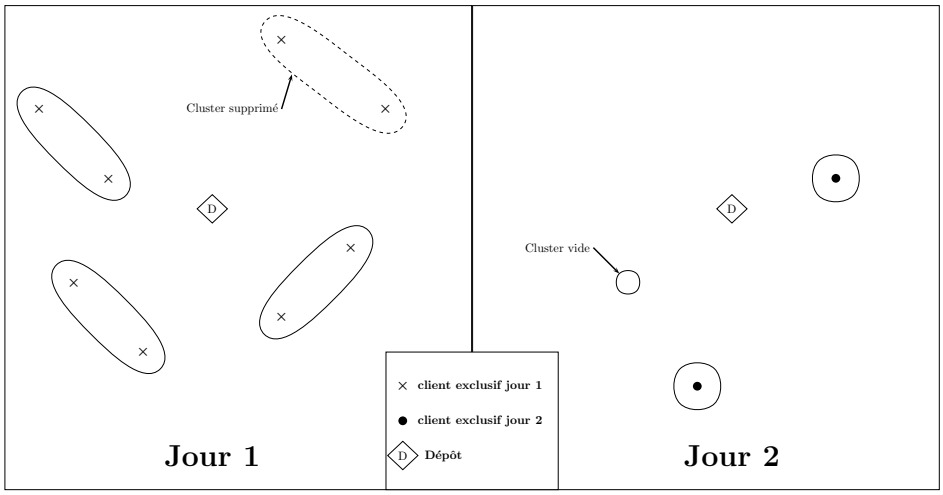


FIGURE 3.8 – Initialisation des clusters

Dans la Figure 3.8, nous avons appliqué l'initialisation des clusters sur l'exemple 3.7. Nous ne considérons donc que les clients exclusifs sur les deux jours. Ayant trois véhicules, il est nécessaire d'avoir trois clusters pour chaque jour. Le premier jour, nous avons 8 clients exclusifs, donc 8 clusters initialement. L'étape de fusion s'arrête à quatre clusters, puisqu'aucune autre fusion n'est possible. Un cluster doit donc être supprimé (cluster en tiret). Donc nous obtenons trois clusters et deux clients exclusifs non affectés. Le deuxième jour, nous avons deux clients exclusifs, donc deux clusters initialement. Le nombre nécessaire de clusters étant de trois, un cluster vide est créé. Nous avons donc trois clusters dont un vide.

Étape 2. Insertion des clients sans recharges

Dans cette étape, nous affectons les clients aux clusters en considérant que le véhicule fera la tournée sans passer par une station de recharge, l'autonomie du véhicule devient donc une limite à ne pas dépasser. Cette hypothèse nous permet de prioriser des tournées sans visites aux stations et donc d'éviter des coûts de recharge supplémentaires.

Soit $Cl = \bigcup_{d=1}^{|H|} Cl_d$ l'ensemble des clusters disponibles sur tout l'horizon H de planification, clusters créés et initialisés à l'étape précédente. Et soit L la liste des clients i non affectés à un cluster, triés par ordre croissant selon $|D(i)|$. Cette ordre permet de traiter les clients ayant le moins de jours de disponibilité en premier puisqu'ils sont les plus contraints en terme d'affectation.

Algorithm 7 Insertion des clients sans recharges

1: **Input :** $Cl = \bigcup_{d=1}^{|H|} Cl_d$: l'ensemble des clusters sur l'ensemble de l'horizon
2: L Liste des clients non affectés à un cluster triée dans l'ordre de leurs fréquences
3: **Output :** L'ensemble Cl actualisé, L_1 : Liste des clients non affectés à la fin de l'algorithme
4: $L_1 = \emptyset$
5: **for all** client $i \in L$ **do**
6: $Feasible_i = \emptyset$ ▷ Ensemble des clusters où l'insertion de i est possible
7: **for all** $a \in Cl_d | d \in D(i)$ **do** ▷ a un cluster
8: **if** $CheckFeasibility(a \cup \{i\})$ **then**
9: Ajouter a à $Feasible_i$
10: **end if**
11: **end for**
12: **if** $Feasible_i \neq \emptyset$ **then**
13: $a^* = \arg\min_{a \in Feasible_i}\{Energy(a \cup \{i\})\}$
14: Insérer le client i dans le cluster a^*
15: Supprimer i de la liste L
16: **else**
17: Insérer i dans L_1, Supprimer i de la liste L.
18: **end if**
19: **end for**

L'Algorithme 7 répète les deux étapes suivantes pour chaque client i dans L :

— la première étape teste l'insertion de i dans chaque jour $d \in D(i)$ et dans chaque cluster $a \in Cl_d$ de ce jour d (ligne 5). Si l'insertion du client i dans le cluster a est faisable, ce cluster est rajouté dans $Feasible_i$. $Feasible_i$ est donc l'ensemble des clusters où le client i peut être inséré en respectant les différentes contraintes.

— La deuxième étape insère i dans le cluster qui minimise l'énergie parmi tous les clusters dans $Feasible_i$ (ligne 12). Dans le cas où aucune insertion n'est faisable pour un client i ($Feasible_i = \emptyset$), le client est supprimé de L et inséré dans la liste L_1 (ligne 16).

A la fin de cet algorithme, la liste L_1 contient tous les clients éjectés pour cause de dépassement de capacité, de contrainte de temps ou du non-respect de la contrainte énergétique. Tous les clients en L_1 sont considérés à l'étape suivante.

Dans la Figure 3.9, nous avons donc appliqué l'algorithme d'insertion des clients dans les clusters existants. La liste L contient dans cet ordre là, les deux clients exclusifs du premier jour, les 12 clients non exclusifs. Les clients non exclusifs sont donc considérés et peuvent être insérés dans un des deux jours. Les deux clients exclusifs du premier jour non affectés à l'étape précédente sont traités en premier, ce qui permet d'en insérer un dans le cluster le plus proche. A la fin de l'algorithme, la liste L_1 contient quatre clients dont un client exclusif du premier jour.

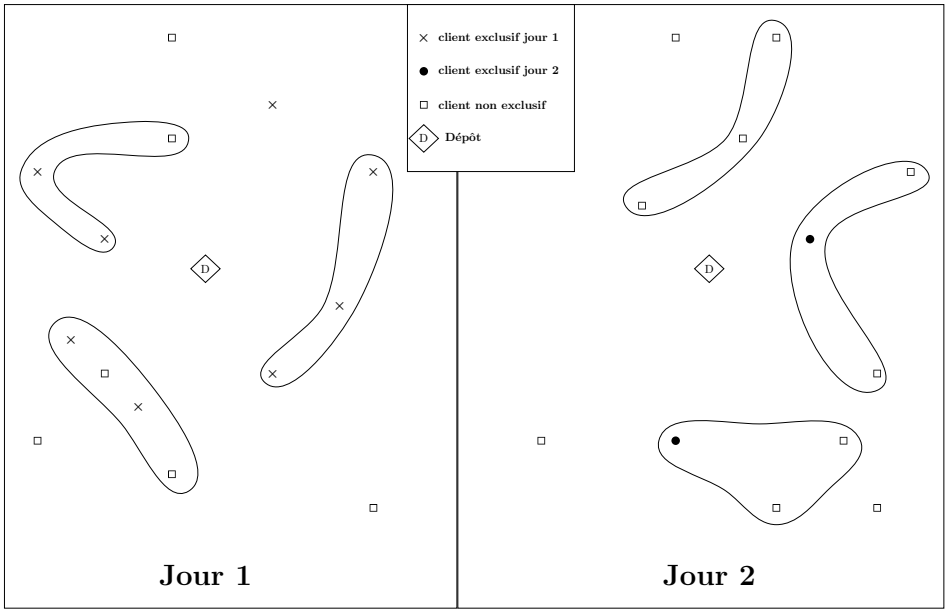


FIGURE 3.9 – Insertion des clients sans recharges

Étape 3. Insertion du client avec considération de la recharge

Nous savons que si la recharge d'un véhicule ne suffit pas à réaliser une tournée, il est possible de "réparer" la tournée en insérant une/des stations de recharges. La contrainte d'énergie exprimée par $CheckEnergyFeasibility$ est donc suffisante mais pas nécessaire. D'un autre côté, il est difficile de simuler une insertion de stations à cette étape de notre algorithme puisque l'efficacité (gain d'énergie) de l'insertion d'une station dépend du choix de la station et de son emplacement dans la tournée. La tournée n'étant pas encore construite, il est impossible de prédire quelles stations peuvent être insérées et avec quel gain d'énergie. Dans cette étape, nous considérons donc la possibilité d'insérer une station en permettant à la consommation énergétique (fonction $Energy()$) des clusters de dépasser l'autonomie d'un véhicule tout en minimisant l'énergie supplémentaire pour augmenter les probabilités de pouvoir réparer la tournée.

L'Algorithme 8 vise à insérer tous les clients à partir de L_1 tout en minimisant la consommation d'énergie. De même que lors de l'étape précédente, L_1 est trié selon la valeur $|D(i)|$. Itérativement, pour chaque client $i \in L_1$ l'algorithme sélectionne les clusters où l'insertion vérifie les contraintes de capacité et de durée maximale (ligne 7). Parmi toutes les possibles insertions ($Feasible_i$), l'algorithme choisit celle qui minimise l'augmentation énergétique du cluster (ligne 13).

Dans la Figure 3.10, les clients dans la liste L_1 (cercle en tiret dans la figure) ont été insérés dans les clusters en utilisant l'Algorithme 8.

Étape 4. Construction d'itinéraires

Dans cette étape, chaque cluster sera considéré pour construire une tournée en utilisant une heuristique de meilleure insertion. Pour chaque cluster, cette phase considère d'abord une tournée vide qui commence et termine au dépôt, puis insère les clients de manière séquentielle. A chaque itération, pour chaque client i, l'algorithme simule l'insertion de i dans chaque position de la

Algorithm 8 Insertion des clients avec considération de la recharge

Input : $Cl = \bigcup_{d=1}^{|H|} Cl_d$: l'ensemble des clusters sur l'ensemble de l'horizon
        L_1 Liste des clients non affectés à un cluster triès selon la fréquence
Output : L'ensemble Cl actualisé
        L_1 Liste des clients non affectés actualisée
for all client $i \in L_1$ **do**
    $Feasible_i = \emptyset$ ▷ Ensemble des clusters où l'insertion de i est possible
    for all $a \in Cl_d | d \in D(i)$ **do** ▷ a un cluster
        if $CheckConstraintsFeasibility(a \cup \{i\})$ **then**
            Ajouter a à $Feasible_i$
        end if
    end for
    if $Feasible_i \neq \emptyset$ **then**
        $a^* = \arg\min_{a \in Feasible_i}\{Energy(a \cup \{i\})\}$
        Insérer le client i dans le cluster a^*
        Supprimer le client i de la liste L_1
    end if
end for

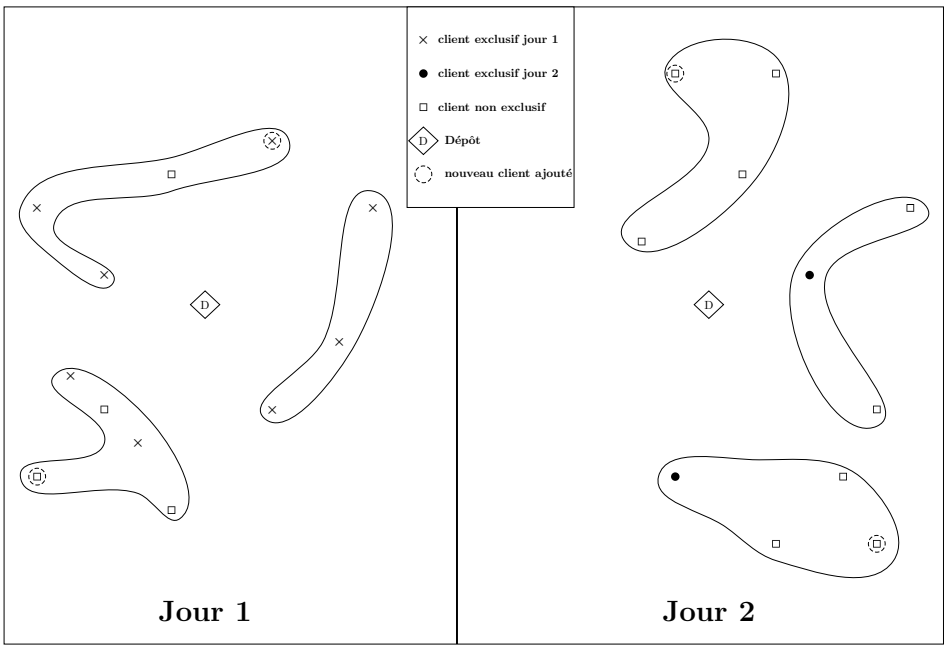


FIGURE 3.10 – Insertion des clients avec considération de la recharge

tournée. Si l'énergie résiduelle du véhicule n'est pas suffisante pour ajouter i, l'algorithme simule simultanément l'insertion de i et d'une station de recharge $b \in B$. La variation du coût total $f(x)$ est calculée pour chaque simulation. Le client i avec le coût d'insertion minimum est inséré à sa meilleure position à la fin de chaque itération. le coût d'insertion d'un client comprend le coût d'insertion des stations de recharge si nécessaire.

Soit la tournée $Tour$ et i un client. La fonction $SimulInsert(Tour, i)$ (explicitée précédemment dans la section 3.5.1) teste l'insertion du client i dans la tournée $Tour$ et retourne le coût engendré et la tournée résultante par cette insertion et par l'éventuelle insertion d'une station. Si aucune insertion n'est possible, la fonction retourne la valeur ∞.

Algorithm 9 Construction d'une tournée à partir d'un cluster cl

1: **Input :** cl : Cluster de clients
2: **Output :** Tournée $Tour$
3: Soit $Tour$ tournée vide ▷ partant de $\bar{0}$ à $\underline{0}$
4: **while** $cl \neq \emptyset$ **do**
5: $\quad i^* = \arg\min\{\forall i \in cl : SimulInsert.Cost(Tour, i)\}$
6: $\quad$ **if** $SimulInsert.Cost(Tour, i^*) \neq \infty$ **then**
7: $\quad\quad Tour = SimulInsert.Tour(Tour, i^*)$
8: $\quad\quad$ Supprimer i^* de cl
9: $\quad$ **else**
10: $\quad\quad cl = \emptyset$
11: $\quad$ **end if**
12: **end while**

L'Algorithme 9 choisit donc à chaque itération le client dont l'insertion dans la tournée augmente le moins le coût et l'insère (ligne 6). Si aucune insertion n'est possible, l'algorithme s'arrête à la tournée partielle obtenue (ligne 9).

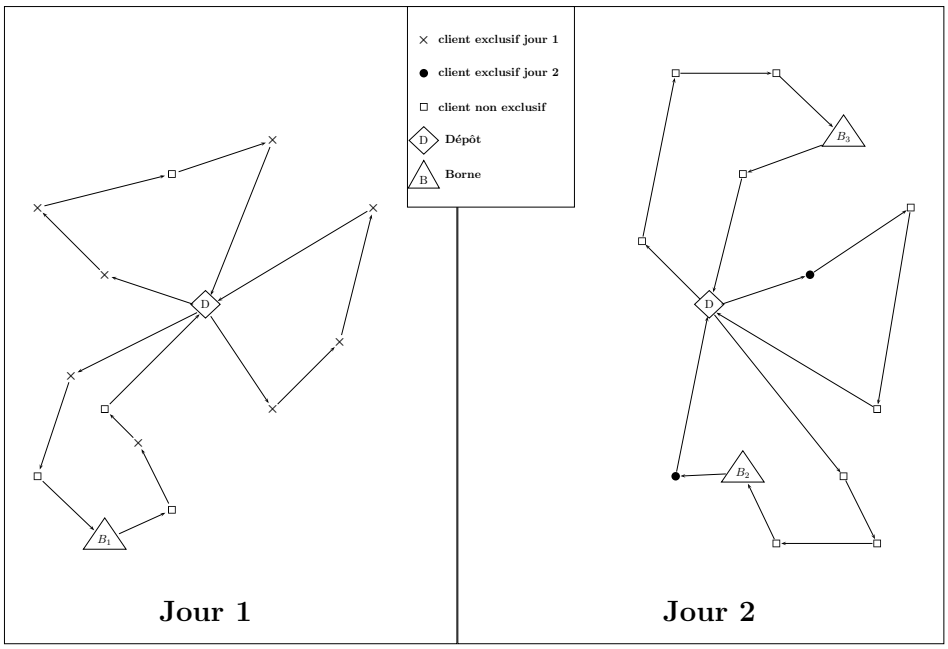


FIGURE 3.11 – Construction des tournées

Dans la Figure 3.11, nous construisons les tournées à partir des clusters obtenus dans l'étape précédente (Figure 3.10). Dans l'exemple, un des clusters modifiés à l'étape 4 a été construit sans insertion d'une station de recharge, ce qui est possible ayant utilisé des fonctions ($Distance$, $energy$...) d'estimations surévalués.

3.7 Heuristique de clustering CLH2

L'heuristique CLH2 est une variante de CLH1, elle procède comme CLH1 sauf pour les étapes (2) et (3) où les insertions de clients sont évalués de manière récursive. Au lieu d'insérer les clients dans l'ordre donné, nous favorisons ici la meilleure insertion possible entre tous les clients ayant le même nombre de jours de disponibilité. Cette considération sera plus coûteuse en temps de calcul mais permet éventuellement une meilleure clustérisation.

Nous avons vu dans l'étape 2 (respectivement étape 3) de $CLH1$ que tous les clients non insérés sont groupés dans la liste L (respectivement L_1) et sont triés dans l'ordre croissant de leurs jours de disponibilité $|D(i)|$ puis insérés dans cet ordre. Ici, nous rassemblons les clients ayant le même nombre de jours de disponibilité β dans une liste L^β dans l'étape 2 ($L = \bigcup_{\beta=1}^{|H|} L^\beta$) et L_1^β pour l'étape 3 ($L_1 = \bigcup_{\beta=1}^{|H|} L_1^\beta$). On traite chaque liste séparément dans l'ordre croissant de d. Ci-dessous, le détail du traitement dans les deux étapes modifiées.

Étape 2. Insertion (Best insertion) des clients sans recharges

L'Algorithme 10 traite les listes $\{L^1, L^2, ..\}$ dans l'ordre de la fréquence, chaque liste est traitée complètement avant le passage à la suivante.

Pour chaque liste L^f, l'insertion de tous les clients dans tous les clusters est testée et évaluée puis la meilleure insertion est choisie (Ligne 19) ; le client ayant la meilleure insertion est inséré dans le cluster correspondant. Les insertions considérées respectent évidemment les jours de disponibilité des clients et les autres contraintes ($CheckFeasibility$). Le processus est répété jusqu'à ce que tous les clients soient insérés ou qu'aucune insertion n'est plus possible (ligne 23), puis on passe à la liste suivante $L^{\beta+1}$. Les clients non insérés d'une liste L^f sont insérés dans L_1^β pour l'étape suivante.

Étape 3. Insertion (Best insertion) du client avec considération de la recharge

Dans l'Algorithme 10, nous considérons dans l'ordre croissant des jours de disponibilité les ensembles $\{L_1^1, L_1^2...\}$. Parmi toutes les insertions respectant les jours de disponibilité et vérifiant $CheckConstraintsFeasibility$, celle minimisant l'énergie supplémentaire est retenue.

La modification de ces deux étapes pourrait permettre d'avoir des clusters mieux construits même si la démarche reste gloutonne et donc pas forcément efficace.

3.8 Expérimentations

L'objectif de cette section est d'évaluer la qualités des heuristiques proposées. Nous commençons par décrire les instances générées. Ensuite, nous présentons et analysons les résultats de chacune des heuristiques sur ces instances et terminons par une étude comparative des résultats de ces heuristiques.

3.8.1 Les instances

Nous proposons de nouvelles instances pour PEVRP inspirées des instances fournies par [33] pour le EVRP. Les instances de [33] sont divisées en deux types : instances de type A, où le dépôt est situé au centre de la carte, et il y a neuf stations de recharge. Les instances de type B où le dépôt est situé dans un coin de la carte et il n'y a que cinq stations de recharge. Dans les deux cas, le dépôt possède une station de recharge accessible pendant la journée. Ces instances

Algorithm 10 Insertion des clients sans recharge pour CLH2

Input : $Cl = \bigcup_{d=1}^{|H|} Cl_d$: l'ensemble des clusters pour chaque jour de l'horizon
$L = \bigcup_{f=1}^{|H|} L^f$ ensemble des clients non affectés, regroupés par leurs nombre de jours de disponibilité f
Output : L'ensemble Cl actualisé,
$L_1 = \bigcup_{f=1}^{|H|} L_1^f$: Liste des clients non affectés, regroupés par leurs fréquences f
$\forall f \in \{1, .., |H|\} : L_1^f := \emptyset$
for all $f \in \{1, .., |H|\}$ **do**
  while $L^f \neq \emptyset$ **do**
    for all client $i \in L^f$ **do**
      $BestCluster_i = \emptyset$ ▷ Cluster où l'insertion de i est la meilleure
      $BestEnergie_i = \infty$ ▷ Énergie minimum pour l'insertion de i
      for all $(a \in Cl_d \quad | \quad d \in D(i))$ **do** ▷ a un cluster
        if $(CheckFeasibility(a \cup \{i\}))$ & $(Energie(a \cup \{i\}) < BestEnergie_i)$ **then**
          $BestCluster_i = a$,
          $BestEnergie_i = Energie(a \cup \{i\})$
        end if
      end for
    end for
    if $(\min\{BestEnergie_i : \forall i \in L^f\} \neq \infty)$ **then**
      $i^* = \arg\min_{i \in L^f}\{BestEnergie_i\}$
      Insérer i^* dans $BestCluster_{i^*}$
      Supprimer i^* de L^f
    else ▷ Aucune insertion n'est possible dans L^f
      $L_1^f = L^f$, $L^f = \emptyset$
    end if
  end while
end for

nous fournissent aussi la position géographique et la demande de chaque client. Nous ajustons les autres paramètres de l'article [33] comme suit :

- Nombre de clients : n=100 et n=200.
- Nombre de véhicules électriques fixe $\in \{3, 4, 5, 6\}$.
- Puissance de la batterie : 20 kWh (équivalent à une autonomie de 160 km).
- La consommation d'énergie : 0,125 kWh/km.
- Vitesse moyenne : 80 km/h.
- Capacité du véhicule : 2500 Kg.
- Durée maximale d'une tournée : 8 h.
- Puissance de recharge : 10 kW et un coût fixe de service de recharge de 2,5 euros

Les instances [33] ne considèrent pas d'horizons, ni de jours de disponibilité. Nous avons utilisé des fonctions aléatoires pondérées pour générer les jours de disponibilité tout en respectant une distribution. Cette distribution est définie par le nombre de jours de disponibilité de chaque client, l'objectif étant que le nombre de clients ayant d jours de disponibilités soit réaliste et

Algorithm 11 Insertion des clients avec recharge pour CLH2

Input : $Cl = \bigcup_{d=1}^{|H|} Cl_d$: l'ensemble des clusters pour chaque jour de l'horizon
$L_1 = \bigcup_{f=1}^{|H|} L_1^f$ ensemble des clients non affectés, regroupés par leurs fréquences f
Output : L'ensemble Cl actualisé,
for all $f \in \{1, .., |H|\}$ **do**
  $Stop := false$
  while $L_1^f \neq \emptyset$ & $Stop = false$ **do**
    for all client $i \in L_1^f$ **do**
      $BestCluster_i = \emptyset$ ▷ Cluster où l'insertion de i est la meilleure
      $BestEnergie_i = \infty$ ▷ Énergie minimum pour l'insertion de i
      for all $(a \in Cl_d \mid d \in D(i))$ **do** ▷ a un cluster
        if $(CheckConstraintsFeasibility(a \cup \{i\}))$ & $(Energie(a \cup \{i\}) < BestEnergie_i)$ **then**
          $BestCluster_i = a$,
          $BestEnergie_i = Energie(a \cup \{i\})$
        end if
      end for
    end for
    if $(\min\{BestEnergie_i : \forall i \in L_1^f\} \neq \infty)$ **then**
      $i^* = \arg\min_{i \in L_1^f}\{BestEnergie_i\}$
      Insérer i^* dans $BestCluster_{i^*}$
      Supprimer i^* de L_1^f
    else ▷ Par contraintes de temps ou/et de capacité du VE
      $Stop := true$;
    end if
  end while
end for

homogène sur toute les instances.

La taille de l'horizon peut prendre la valeur de 2 jours, 3 jours ou 5 jours. Nous avons alors proposé 3 grands groupes d'instances (ligne du Tableau 3.6). Nous pouvons constater les distributions retournées pour chaque groupe.

Une instance sera définie par trois indices $(H, m, indice)$, $H \in \{2, 3, 5\}$ étant le nombre de jours, $m \in \{3, 4, 5, 6\}$ le nombre de véhicules, et $indice \in \{1, .., 100\}$ étant l'indice de cette instance. (H, m) désignera toutes les instances du groupe.

Le choix du nombre de véhicule influe sur la difficulté des instances, c'est pour cela que nous l'avons fait varier pour les différentes catégories. Pour fixer le nombre de véhicules nous avons effectué une étude préalable en simulant plusieurs valeurs de nombre de VE pour chaque instance. Nous avons donc pour chaque instance de 100 clients, trois instances différentes par leurs nombres de véhicules. Nous avons utilisé les trois heuristiques pour résoudre chaque instance, les résultats sont présentés dans le Tableau 3.7. Pour mettre en évidence la variation de difficulté due à la variation du nombre de véhicules indépendamment de l'heuristique utilisée, le tableau montre, pour chaque sous-catégorie, la moyenne du nombre de clients non visités (C.n.v) sur toutes les solutions et le pourcentage de solutions incomplètes (ayant des clients non visités) parmi toutes

	Nombre de jours de disponibilités					**Nombre**
Instances	1	2	3	4	5	**d'instances**
100 clients et 2 jours	55%	45%	\	\	\	300
100 clients et 3 jours	25%	55%	20%	\	\	300
200 clients et 5 jours	10%	20%	35%	20%	15%	300

TABLE 3.6 – Distribution utilisée pour la génération des instances

les solutions considérées.

Dans le Tableau 3.7, les lignes représentent les différents groupes d'instances (H, m) décrits précédemment. L'objectif du tableau est de montrer l'effet de la variation du nombre de véhicules sur la capacité des heuristiques à trouver des solutions complètes, le nombre de solutions incomplètes (%$\mathbf{Sol}^-$) permet donc de constater l'efficacité des heuristiques. Le nombre de clients non visités ($C.n.v$ minimum, maximum et moyen par groupe) permet de constater la "distance" par rapport à une solution complète et donc le niveau de difficulté de chacun des groupes d'instances. Nous remarquons dans le tableau 3.7 que la tendance reste la même, en augmentant le nombre de VE, le $C.n.v$ et %$\mathbf{Sol}^-$ diminuent considérablement.

Instances	**C.n.v min**	**C.n.v moyen**	**C.n.v max**	%$\mathbf{Sol}^-$
(2,4)	6	10,48	18	100%
(2,5)	0	0,39	6	18%
(2,6)	0	0,02	4	1%
(3,3)	0	2,90	14	71%
(3,4)	0	0,11	3	8%
(3,5)	0	0,00	0	0%
(5,3)	23	30,42	47	100%
(5,4)	0	1,3	9	39%
(5,5)	0	0,22	3	16%

TABLE 3.7 – Impact de la variation du nombre de véhicules

Dans ce qui suit, nous utilisons ces métriques pour analyser et comparer les résultats :
- **Coût.s** : coût simple (voir Section 3.3.2)
- **C.n.v** nombre de clients non visités.
- **Coût.c** : coût complet (voir Section 3.3.2)
- **CPU** : temps d'exécution en secondes sur un ordinateur Intel Core (TM) i7-5600U CPU, avec un processeur de 2.60 GHz.
- **Demande Moy** : Demande moyenne d'une tournée
- %$\mathbf{Sol}^-$: pourcentage de solutions incomplètes (ayant des clients non visités) sur l'ensemble des solutions considérés (100 par sous catégories).

3.8.2 Évaluation des heuristiques CLH1, CLH2

Dans les tableaux 3.8 et 3.9 nous analysons les résultats de l'heuristique CLH1 sur les instances à 100 clients et celle à 200 clients respectivement. Les tableaux 3.10 et 3.11 présentent les mêmes

résultats pour l'heuristique CLH2.

	inst(2,4)	**inst(2,5)**	**inst(2,6)**	**inst(3,3)**	**inst(3,4)**	**inst(3,5)**
Coût.s Max	2835,91	3121,97	3269,00	3280,45	3689,90	3598,78
Coût.s Min	1507,18	1720,18	1798,05	1693,89	1995,08	2216,92
Coût.s Moy	2185,54	2349,66	2420,50	2557,97	2719,30	2837,86
C.n.v Max	18,00	1,00	0,00	8,00	1,00	0,00
C.n.v Min	6,00	0,00	0,00	0,00	0,00	0,00
C.n.v Moy	10,54	0,03	0,00	1,52	0,01	0,00
Coût.c Max	9428,37	4285,78	3413,00	7224,21	4645,29	3778,78
Coût.c Min	3240,49	1840,18	1942,05	2098,67	2139,08	2396,92
Coût.c Moy	4873,00	2506,45	2564,50	3690,59	2875,43	3017,86
CPU Max	60,00	47,00	33,00	58,00	36,00	22,00
CPU Min	31,00	28,00	19,00	38,00	20,00	14,00
CPU Moy	44,69	37,65	25,74	48,60	28,02	17,56
Demande Moy	1927,31	1725,86	1438,66	1888,29	1438,48	1150,93
%**Sol**$^{-}$	100%	3%	0%	56%	1%	0%

TABLE 3.8 – Évaluation de l'heuristique CLH1 sur les instances à 100 clients

Sur les instances ayant un horizon de deux jours et quatre véhicules (instances (2,4)), l'heuristique CLH1 enregistre ses pires résultats avec aucune solution complète trouvée et une moyenne de $10,54$ clients non visités. Le temps d'exécution moyen (44,69 secondes) est le deuxième le plus élevé parmi tous les groupes d'instances et ceci en ayant le moins de clusters/tournées (8) à considérer et donc théoriquement le moins de tests à faire. Ce nombre réduit de tournées disponibles couplé à la proximité de la demande moyenne d'une tournée $(1927,31)$ avec la capacité d'un véhicule (2000), peuvent expliquer les mauvais résultats sur ces instances. On peut même conclure qu'il est très difficile, voire impossible, de visiter tous les clients dans ces instances. Sur les instances (2,5), l'heuristique trouve une solution complète dans 97 instances, et n'a qu'un client non visité sur les trois autres instances (Figure 3.12). L'ajout de véhicules permet logiquement une meilleure répartition des demandes des clients sur les différentes tournées des différents jours. L'heuristique trouve une solution complète à toutes les instances dans (2.6). On remarque la même progression inversée entre le nombre de véhicules et la capacité de l'heuristique à trouver des solutions complètes dans les instances ayant trois jours comme taille d'horizon et celles ayant cinq.

L'augmentation du nombre de jours/véhicules réduit visiblement le temps d'exécution de l'heuristique pour le même nombre de clients (Figure 3.13). Il est donc plus rapide pour l'heuristique de construire plusieurs clusters/tournées réduites qu'un nombre réduit de clusters/tournées plus conséquents. Pour la première étape de l'heuristique CLH1, un nombre de véhicule élevé permet d'avoir moins de fusions à tester et faire. La troisième étape est potentiellement obsolète si le nombre de véhicules permet d'affecter tous les clients à l'étape 2. Dans l'étape de construction des tournées, l'augmentation du nombre de véhicules permet d'éviter les tests de réparation,

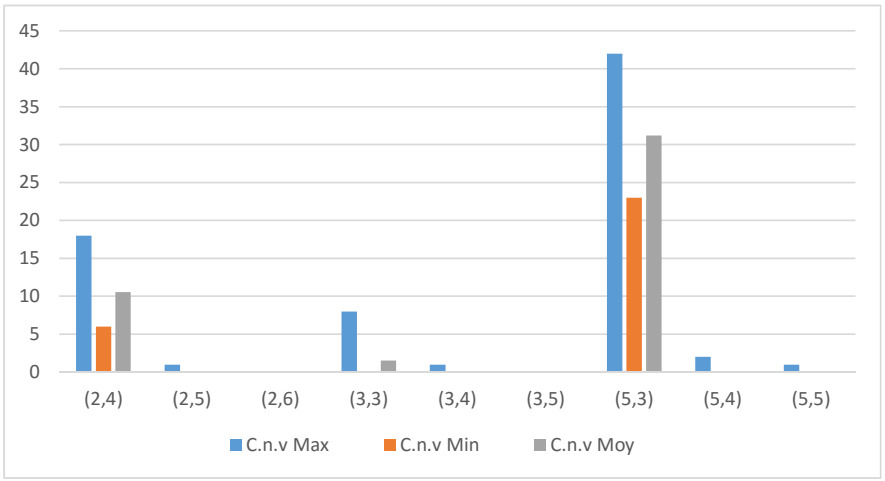


FIGURE 3.12 – Évolution du nombre de clients non visités par groupe d'instances pour l'heuristique CLH1

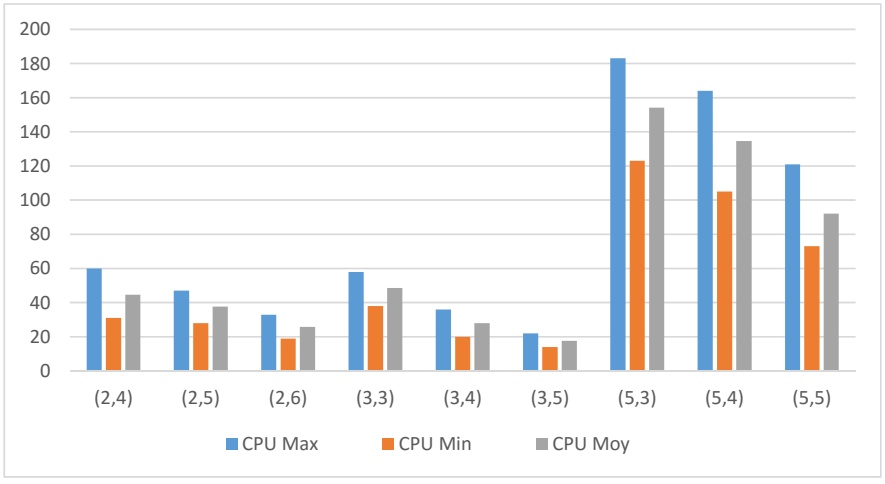


FIGURE 3.13 – Évolution du temps d'exécution par groupe d'instances pour l'heuristique CLH1

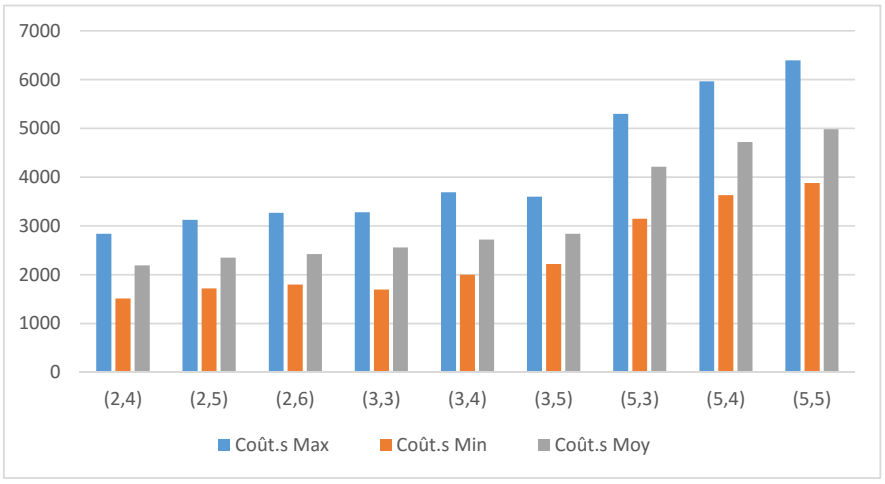


FIGURE 3.14 – Évolution du coût simple par groupe d'instances pour l'heuristique CLH1

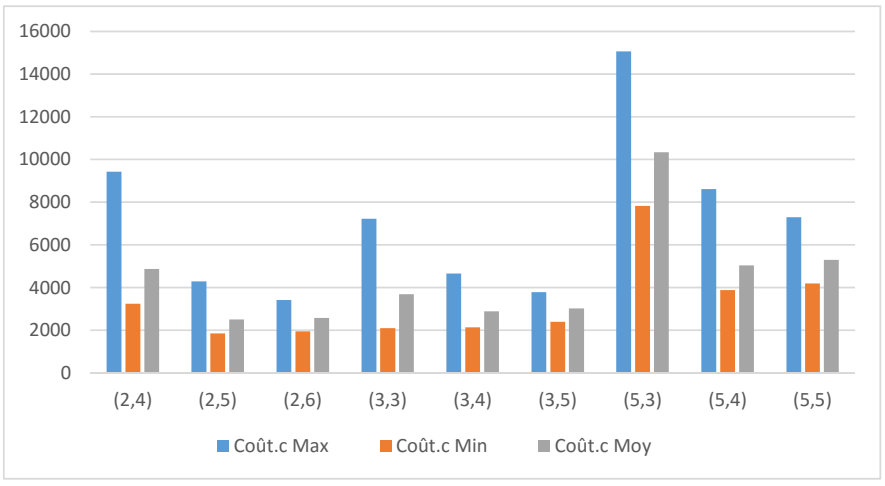


FIGURE 3.15 – Évolution du coût complet par groupe d'instances pour l'heuristique CLH1

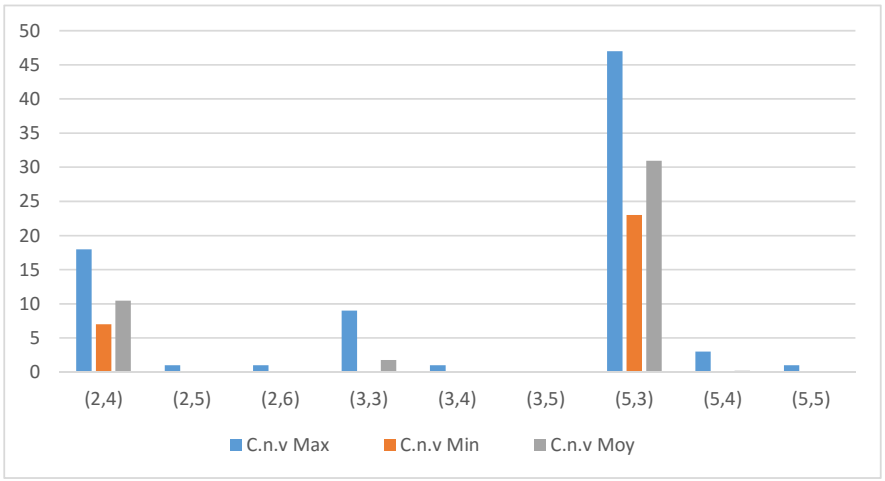


FIGURE 3.16 – Évolution du nombre de clients non visités par groupe d'instances pour l'heuristique CLH2

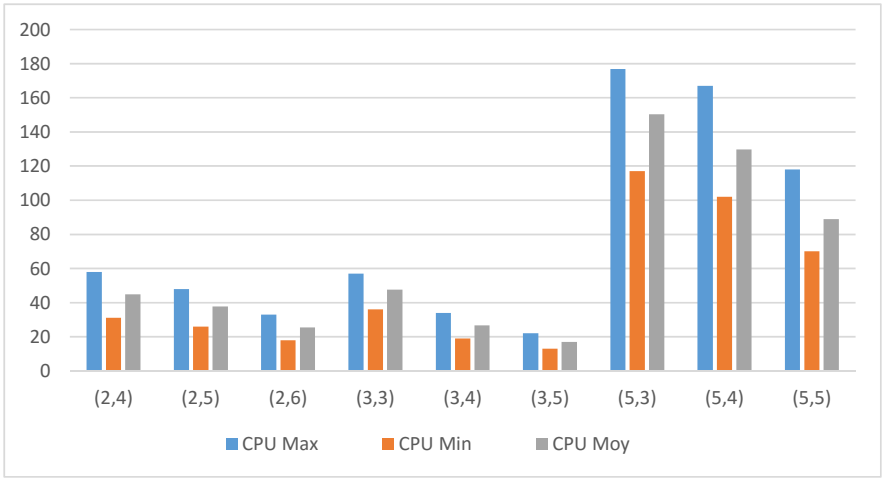


FIGURE 3.17 – Évolution du temps d'exécution par groupe d'instances pour l'heuristique CLH2

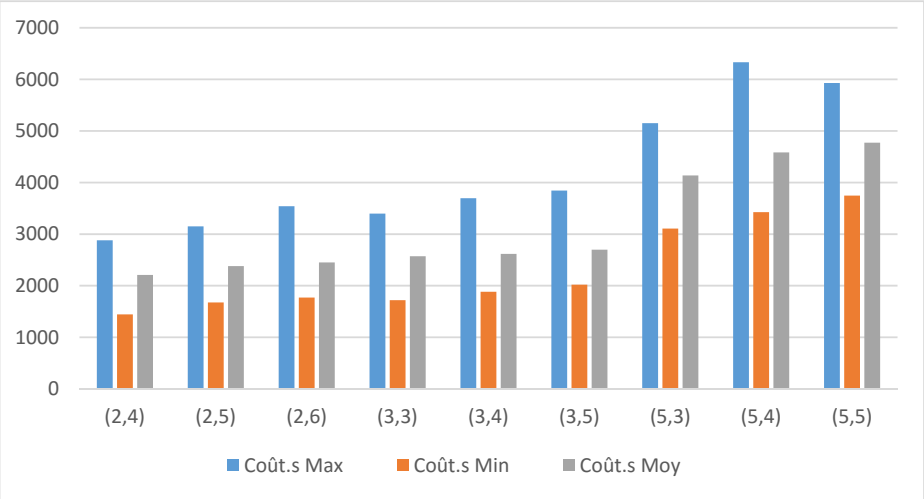


FIGURE 3.18 – Évolution du coût simple par groupe d'instances pour l'heuristique CLH2

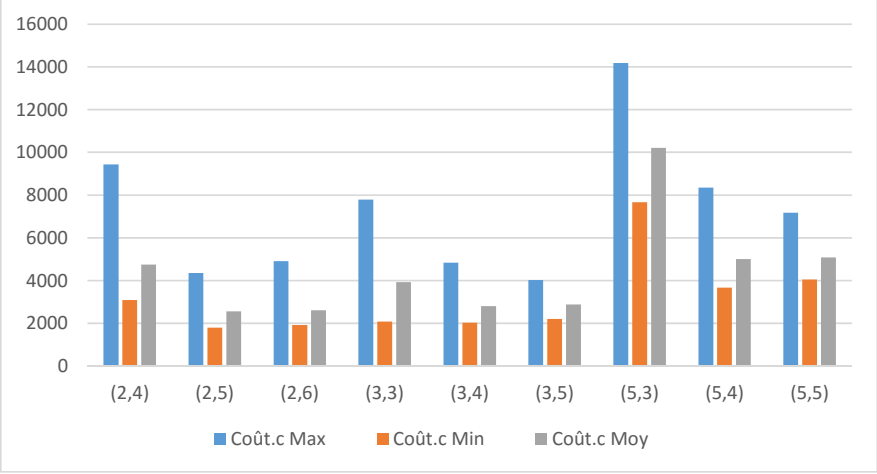


FIGURE 3.19 – Évolution du coût complet par groupe d'instances pour l'heuristique CLH2

inst(5,3)	**inst(5,4)**	**inst(5,5)**	
Coût.s Max	5296,68	5964,06	6391,55
Coût.s Min	3145,05	3630,55	3882,64
Coût.s Moy	4216,09	4718,41	4983,34
C.n.v Max	42,00	2,00	1,00
C.n.v Min	23,00	0,00	0,00
C.n.v Moy	31,18	0,07	0,01
Coût.c Max	15064,30	8617,25	7285,51
Coût.c Min	7820,49	3870,55	4182,64
Coût.c Moy	10347,39	5038,14	5293,91
CPU Max	183,00	164,00	121,00
CPU Min	123,00	105,00	73,00
CPU Moy	154,12	134,53	92,08
Demande Moy	1932,01	1718,49	1375,18
%**Sol**$^{-}$	100%	6%	1%

TABLE 3.9 – Évaluation de l'heuristique CLH1 sur les instances à 200 clients

tests coûteux en temps d'exécution.

Les résultats de l'heuristique CLH2 sont sensiblement similaires aux résultats de l'heuristique CLH1. Sur la moyenne des clients non visités, l'heuristique CLH2 obtient des résultats entre 0.25% plus mauvais et égale à CLH1. Sur les coûts complets, les résultats de CLH1 sont plus disparates avec des intervalles [min,max] plus larges sur chacune des sous-catégories (Figure 3.19). Ce qui place le coût complet moyen dans un intervalle $[-6\%, 5\%]$ selon les sous-catégories d'instances. En revanche, l'heuristique CLH2 est plus rapide sur 4 des 6 sous-catégories d'instances (Figure 3.17).

3.8.3 Évaluation de l'heuristique BIH

Nous avons précédemment avancé la difficulté des instances (2.4) liées à la capacité limitée des véhicules, cette supposition se renforce avec les résultats de l'heuristique BIH sur ces instances (Tableau 3.12). En revanche, les résultats sur les autres sous-catégories d'instances (spécialement (2.5) et (3.3)) montrent une difficulté spécifique à l'heuristique BIH à trouver des solutions complètes. Cette difficulté peut s'expliquer par le fait les choix dans BIH sont dictés par la minimisation du coût sans anticipation sur les différentes contraintes (disponibilité, capacité, durée maximale) dans le choix des clients à insérer, et de se retrouver avec des clients "difficiles" à insérer dans des tournées déjà complètes. En effet, un client éloigné du dépôt et isolé des autres clients aura un coût d'insertion énorme et sera inséré dans les dernières itérations quand la majorité des tournées sont "complètes" (capacité du véhicule consommé, durée maximale de la tournée atteinte). Si ce client est exclusif et/ou a une demande considérable, l'heuristique n'anticipera pas sur sa difficulté et son insertion sera compliquée voire impossible vu le peu de tournées en état de l'intégrer.

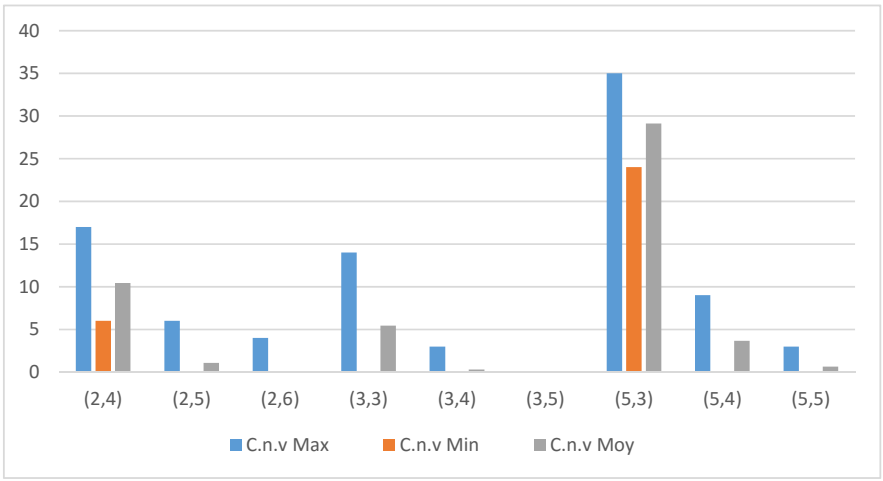


FIGURE 3.20 – Évolution du nombre de clients non visités par groupe d'instances pour l'heuristique BIH

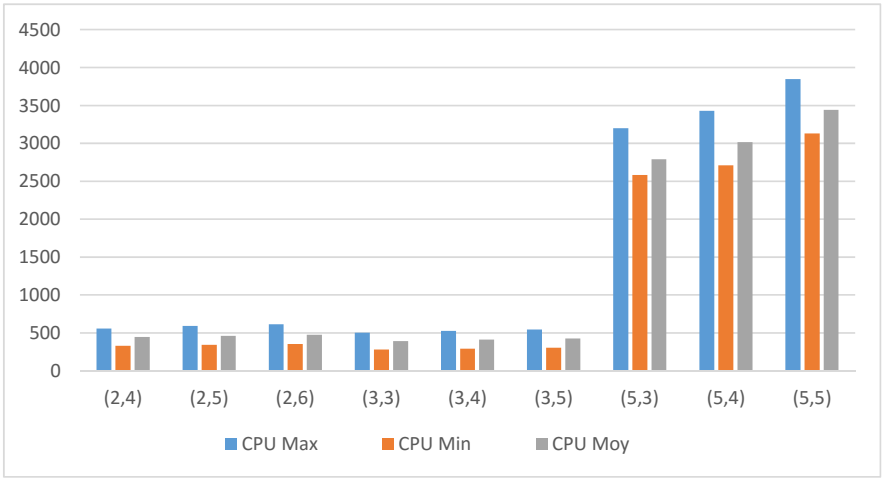


FIGURE 3.21 – Évolution du temps d'exécution par groupe d'instances pour l'heuristique BIH

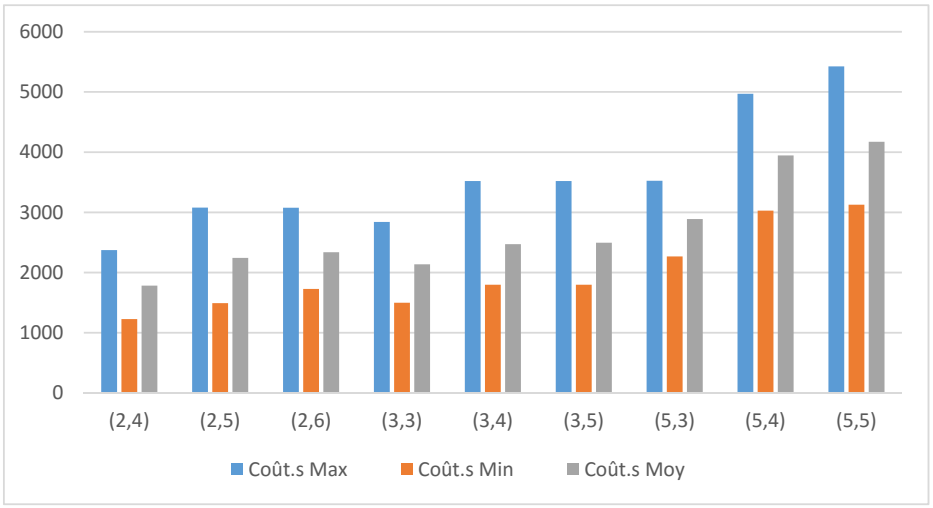


FIGURE 3.22 – Évolution du coût simple par groupe d'instances pour l'heuristique BIH

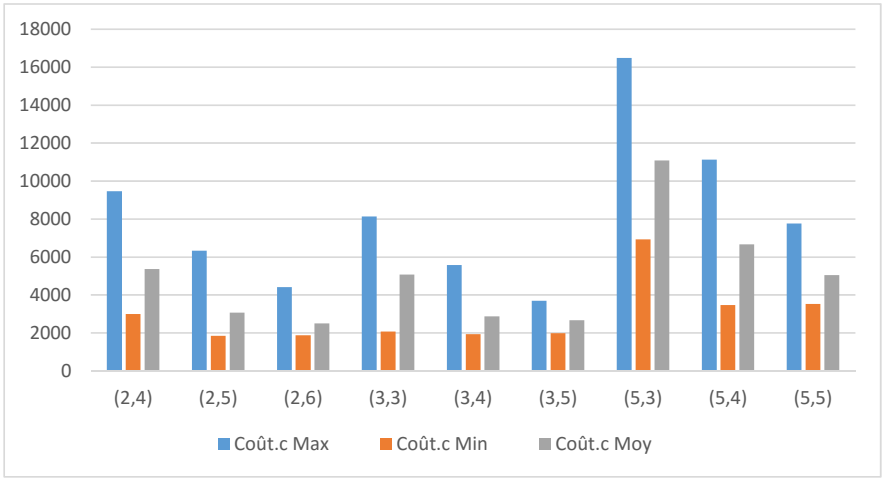


FIGURE 3.23 – Évolution du coût complet par groupe d'instances pour l'heuristique BIH

	inst(2,4)	inst(2,5)	inst(2,6)	inst(3,3)	inst(3,4)	inst(3,5)
Coût.s Max	2880,17	3149,53	3542,10	3398,34	3696,09	3844,20
Coût.s Min	1444,65	1674,50	1771,93	1720,95	1882,85	2022,31
Coût.s Moy	2210,22	2382,02	2450,23	2572,73	2616,87	2698,17
C.n.v Max	18,00	1,00	1,00	9,00	1,00	0,00
C.n.v Min	7,00	0,00	0,00	0,00	0,00	0,00
C.n.v Moy	10,47	0,04	0,01	1,77	0,03	0,00
Coût.c Max	9428,37	4344,70	4899,24	7783,12	4840,55	4024,20
Coût.c Min	3078,39	1794,50	1915,93	2070,67	2026,85	2202,31
Coût.c Moy	4742,48	2550,10	2606,36	3921,59	2797,21	2878,17
CPU Max	58,00	48,00	33,00	57,00	34,00	22,00
CPU Min	31,00	26,00	18,00	36,00	19,00	13,00
CPU Moy	44,90	37,68	25,50	47,61	26,69	16,88
Demande Moy	1926,4	1725,7	1438,48	1883,9	1438,21	1150,93
%**Sol**$^{-}$	100%	4%	1%	57%	3%	0%

TABLE 3.10 – Évaluation de l'heuristique CLH2 sur les instances à 100 clients

En terme de temps d'exécution, le test et la comparaison à chaque itération de toutes les insertions possibles à chaque position de toutes les tournées est inévitablement très coûteux ce qui implique des temps d'exécution considérables et pouvant atteindre 1 heure de temps d'exécution (figure 3.21). On remarque une légère corrélation entre le temps d'exécution et le nombre de véhicules (donc de tournées) sur les instances ayant le même horizon, mais cette corrélation ne tient pas entre deux instances avec des horizons différents. En effet, le temps d'exécution des instances ayant trois jours est inférieur à celui des instances ayant deux jours. Cela s'explique partiellement par le fait que sur les instances de trois jours, les tests d'insertion se font seulement sur les jours de disponibilité des clients et non sur tout l'horizon.

3.8.4 Étude comparative des heuristiques

Évaluation basée sur la moyenne

Pour comparer les heuristiques, nous allons présenter la moyenne sur trois métriques (coût complet, CPU, pourcentage de solutions incomplètes) sur les différentes groupes d'instances.

Dans le premier Tableau 3.14, nous présentons le coût complet des trois heuristiques par groupes d'instances. Les résultats montrent que pour des instances critiques en termes de nombre de véhicules, les heuristiques CLH1 et CLH2 donnent de meilleurs résultats que sur les instances avec un nombre plus large de véhicules et donc de tournées. Les heuristiques CLH1 et CLH2 gèrent donc mieux les instances contraintes ce qui peut être expliqué par la priorisation des clients "difficiles" en termes de fréquence. Aussi, le clustering permet de répartir de manière plus intelligente les clients dans les tournées. A l'inverse, l'heuristique BIH construit les tournées de manière séquentielle (sans vision globale) et sur un critère de coût seulement, ce qui engendre

	inst(5,3)	**inst(5,4)**	**inst(5,5)**
Coût.s Max	5151,93	6332,68	5929,32
Coût.s Min	3108,37	3427,18	3745,30
Coût.s Moy	4136,42	4585,12	4773,72
C.n.v Max	47,00	3,00	1,00
C.n.v Min	23,00	0,00	0,00
C.n.v Moy	30,94	0,19	0,01
Coût.c Max	14183,80	8340,90	7174,78
Coût.c Min	7665,15	3667,18	4045,30
Coût.c Moy	10213,45	4998,04	5084,09
CPU Max	177,00	167,00	118,00
CPU Min	117,00	102,00	70,00
CPU Moy	150,37	129,81	88,91
Demande Moy	1932,70	1717,39	1375,20
%**Sol**$^{-}$	100%	14%	1%

TABLE 3.11 – Évaluation de l'heuristique CLH2 sur les instances à 200 clients

de bonnes solutions en termes de coût. De plus, la flexibilité de BIH sur l'utilisation ou non des véhicules disponibles est bénéfique en termes de coût sur les instances ayant un nombre large de véhicules.

Dans le Tableau 3.15, nous présentons le temps d'exécution des trois heuristiques par groupe d'instances. Les temps d'exécution dans le Tableau 3.15 montrent l'avantage des heuristiques CLH1 et CLH2 qui gagnent à décomposer la construction de la solution ("diviser pour régner"). En effet, les heuristiques CLH1 et CLH2 sont vingt fois plus rapides que l'heuristique BIH en moyenne et trouvent la solution 37 fois plus rapidement sur certaines instances (instances (5.5)). En revanche, il est étonnant de voir que l'heuristique CLH2 performe aussi rapidement que CLH1 malgré le traitement récursif des listes de clients ayant la même fréquence dans les étapes d'insertion des clients sans recharge et avec considération de la recharge (étape 2 et 3). Ceci s'explique premièrement par le fait que le traitement dans ses étapes reste élémentaire et la différence entre traitement itératif simple ou récursif est minime. Aussi, le traitement récursif donne potentiellement de meilleurs clusters et facilite donc la construction des tournées.

Dans le Tableau 3.16, nous présentons le nombre moyen de clients non visités des trois heuristiques par groupe d'instances. En terme d'efficacité pour trouver des solutions complètes, l'heuristique CLH1 donne les meilleurs résultats suivie de très près par l'heuristique CLH2 ($\leq 1\%$) sur toutes les instances. Comme explicité précédemment, cette performance revient au traitement prioritaire des clients les plus difficiles en terme de planification et à la clustérisation qui permet une répartition plus équilibrée des clients.

	inst(2,4)	inst(2,5)	inst(2,6)	inst(3,3)	inst(3,4)	inst(3,5)
Coût.s Max	2371,85	3078,47	3075,97	2841,02	3519,73	3519,73
Coût.s Min	1225,94	1490,41	1727,27	1499,27	1796,72	1798,03
Coût.s Moy	1782,24	2242,52	2337,83	2135,78	2471,82	2496,67
C.n.v Max	17,00	6,00	4,00	14,00	3,00	0,00
C.n.v Min	6,00	0,00	0,00	0,00	0,00	0,00
C.n.v Moy	10,43	1,09	0,05	5,42	0,30	0,00
Coût.c Max	9459,53	6329,81	4414,38	8131,34	5574,61	3699,73
Coût.c Min	2995,86	1847,27	1871,27	2069,16	1942,03	1978,03
Coût.c Moy	5364,37	3068,92	2507,50	5081,78	2871,42	2676,67
CPU Max	559,00	591,00	617,00	504,00	528,00	547,00
CPU Min	330,00	345,00	356,00	283,00	295,00	304,00
CPU Moy	445,30	462,96	476,88	394,44	411,40	427,03
Demande Moy	1929,67	1707,64	1437,95	1812,51	1434,33	1150,93
%**Sol**$^-$	100%	47 %	2%	99%	20%	0%

TABLE 3.12 – Évaluation de l'heuristique BIH sur les instances à 100 clients

Évaluation basée sur la méthode des profils de performances

Pour mieux comparer les résultats des différentes heuristiques, nous avons adapté et utilisé la méthode "performance profiles" présenté dans [28] qui permet de comparer plus efficacement plusieurs méthodes de résolution sur une métrique. Nous avons choisi de considérer le coût complet (Coût.c) comme métrique de comparaison. Soit $\mathbb{H}$ l'ensemble des trois heuristiques et $\mathbb{I}$ l'ensemble des instances, on posera :

$$p_{h,i} = \text{Le Coût.c de la solution obtenu par l'heuristique } h \in \mathbb{H} \text{ sur l'instance } i \in \mathbb{I}$$

Le gap de ce coût par rapport à la meilleure performance :

$$r_{h,i} = \frac{p_{h,i} - \min\{p_{h',i} : h' \in \mathbb{H}\}}{\min\{p_{h',i} : h' \in \mathbb{H}\}}$$

Puis une fonction qui permet de donner le pourcentage d'instances où une heuristique h est à une distance τ de la meilleure solution obtenue :

$\rho_h(\tau) = \frac{1}{|\mathbb{I}|}|\{i \in I : r_{h,i} \leq \tau\}$

Cette méthode permet de voir quelle heuristique donne les meilleurs résultats dans l'absolue ($\tau = 0$), mais permet aussi de voir quelle heuristique performe le mieux avec une marge de tolérance $\tau > 0$. Pour mieux comparer les heuristiques, la méthode suggère d'utiliser une courbe représentant l'évolution du $\rho_h(\tau)$ à la variation du τ pour chaque heuristique. L'heuristique ayant la courbe la plus haute est logiquement la meilleure. Dans la Figure 3.24 nous présentons la courbe de la fonction ρ sur les sous-catégories d'instances en initialisant τ à zéro, puis le variant

	inst(5,3)	**inst(5,4)**	**inst(5,5)**
Coût.s Max	3525,62	4970,03	5425,26
Coût.s Min	2266,69	3028,96	3127,77
Coût.s Moy	2888,83	3946,61	4173,05
C.n.v Max	35,00	9,00	3,00
C.n.v Min	24,00	0,00	0,00
C.n.v Moy	29,13	3,65	0,65
Coût.c Max	16496,20	11132,10	7768,93
Coût.c Min	6925,27	3473,14	3526,03
Coût.c Moy	11093,85	6659,39	5038,45
CPU Max	3198,00	3429,00	3848,00
CPU Min	2583,00	2709,00	3132,00
CPU Moy	2789,12	3016,10	3441,62
Demande Moy	1950,34	1687,37	1370,70
%**Sol**$^-$	100%	98%	45%

TABLE 3.13 – Évaluation de l'heuristique BIH sur les instances à 200 clients

en une suite géométrique dont le premier terme est 2% avec une raison de $1,2$. Dans la Figure 3.25, nous étudions la même méthode mais sur l'ensemble des instances considérées, τ initialisé à zéro, puis variant en une suite géométrique dont le premier terme est $1,5\%$ avec une raison de $1,2$.

Les graphes dans la Figure 3.24 montrent plus clairement la corrélation entre le type d'instances et la performance des différentes heuristiques. Sur les instances ayant un nombre de véhicules réduit, les heuristiques CLH1 et CLH2 trouvent plus facilement des solutions complètes à la différence de la BIH ce qui influe grandement sur le coût complet de la solution et donc sur le graphe. Inversement, l'heuristique BIH donne de meilleur solutions sur les instances moins contraintes du fait de son seul critère de décision qui est le coût.

CLH1 performe mieux sur trois groupes d'instances $\{(2.5), (2.6), (3.3)\}$ et donc sur des instances avec peu de tournées (jours * véhicules). L'heuristique CLH2 performe donc mieux sur le reste des instances. De plus, sur les instances (3.4) et (3.5) on remarque une plus grande différence entre les deux à l'avantage de l'heuristique CLH2, contrairement aux autres instances où la différence est moindre. Cet avantage de l'heuristique CLH2 peut s'expliquer par un impact plus conséquent de la construction récursive des clusters quand le nombre des tournées est assez grand.

Dans la Figure 3.25 on constate que l'heuristique CLH2 est la meilleure puisque son tracé domine le tracé des deux autres heuristiques. Initialement, l'heuristique BIH commence aussi bien que CLH2 mais sa courbe de performance ne croit pas assez rapidement et se fait rattraper (à $taux = 4\%$) par l'heuristique CLH.

Instances	**BIH**	**CLH1**	**CLH2**
(2,4)	5364,37	4873	**4742,48**
(2,5)	3068,92	**2506,45**	2550,1
(2,6)	**2507,5**	2564,5	2606,36
(3,3)	5081,78	**3690,59**	3921,59
(3,4)	2871,42	2875,43	**2797,21**
(3,5)	**2676,67**	3017,86	2878,17
(5,3)	11093,85	10347,39	**10213,45**
(5,4)	6659,39	5038,14	**4998,04**
(5,5)	**5038,45**	5293,91	5084,09
Moyenne	4929,15	4467,48	**4421,28**

TABLE 3.14 – Évaluation comparative du coût complet par heuristique/groupe d'instances

3.8.5 Etude de l'impact d'une variation de l'énergie de départ des véhicules électriques

Dans tous les résultats présentés précédemment, nous avons supposé que les véhicules partent avec une batterie complètement rechargée. Dans ce qui suit, nous allons étudier l'impact d'une contrainte de recharge au dépôt qui limite l'énergie à recharger pendant la nuit et donc le niveau de batterie au départ le matin. Nous allons considérer que les bornes au dépôt permettent de recharger 75 % de la capacité totale de la batterie des véhicules disponibles pendant la nuit. Nous proposons deux distributions différentes des recharges initiales pour chaque véhicule au matin de chaque jour de l'horizon. La première distribution consiste à recharger chaque véhicule à 75%, et la deuxième consiste à recharger complètement le maximum de véhicules tout en vérifiant que le reste des véhicules partent à au moins 50 % de leurs batteries. Pour évaluer l'impact de ces deux distributions indépendamment des autres contraintes, nous les avons testées sur les instances les moins "difficiles" et donc le groupe d'instances ayant trois jours d'horizons et cinq véhicules et comparées aux solutions obtenues précédemment, où tous les véhicules sont chargés complètement. Nous comparons donc les trois scénarios suivants, chacun défini par la charge initiale des véhicules :

— **Var0** : Tous les véhicules sont chargés à 100% au départ du dépôt.
— **Var1** : Tous les véhicules sont chargés à 75%.
— **Var2** : Deux véhicules chargés à 100%, deux chargés à 50% et le dernier chargé à 75%.

Pour les heuristiques CLH, nous avons ajusté les contraintes d'énergie en redéfinissant $CheckEnergyFeasi$ non plus sur la capacité de la batterie mais sur l'énergie initiale du véhicule auquel le cluster sera affecté.

La première constatation sur les solutions obtenues, est que la plupart des véhicules n'ayant pas une recharge complète commencent leurs tournées en visitant la borne localisée au dépôt puisque cette borne ne nécessite pas de détour.

Dans le Tableau 3.17, on voit que les heuristiques ont besoin d'environ 35% de plus de temps pour résoudre les deux variantes par rapport à Var0. Cette augmentation du temps de résolution s'explique logiquement par l'utilisation plus prématurée et plus fréquente de la fonction

Instances	**BIH**	**CLH1**	**CLH2**
(2,4)	445,3	**44,69**	44,9
(2,5)	462,96	**37,65**	37,68
(2,6)	476,88	25,74	**25,5**
(3,3)	394,44	48,6	**47,61**
(3,4)	411,4	28,02	**26,69**
(3,5)	427,03	17,56	**16,88**
(5,3)	2789,12	154,12	**150,37**
(5,4)	3016,1	134,53	**129,81**
(5,5)	3441,62	92,08	**88,91**
Moyenne	1318,32	64,78	**63,15**

TABLE 3.15 – Évaluation comparative du temps de calcul par heuristique/groupe d'instances

de réparation dûe à la recharge initiale inférieure des véhicules. Entre les deux variantes, la première nécessite plus de temps (1.5% en moyenne) pour obtenir des solutions. Cette différence s'explique en partie par le fait que les deux véhicules ayant une recharge complète nécessitent moins de visites aux bornes de recharges.

Sur l'ensemble des solutions obtenues pour les deux scénarios, trois ont un client non visité. Deux de ces solutions incomplètes sont obtenues sur la même instance par l'heuristique CLH. La dernière solution incomplète est obtenue par BIH avec le scénario Var2. Pour les deux solutions incomplètes obtenues par CLH, il est probable que l'ordre des clients ne permette pas de construire des clusters/tournées réalisables puisque CLH2 réussit à visiter tous les clients sur cette instance. La solution incomplète obtenue par BIH étant seulement sur le scénario Var2, il est probable que les recharges nécessaires pour visiter le dernier client cause un dépassement de la durée maximale des tournées potentielles.

Le coût des solutions augmente de $1,8\%$ en moyenne pour les deux scénarios en comparaison au scénario initial. L'heuristique BIH donne en moyenne les meilleures solutions sur les deux scénarios avec un avantage de $0,6\%$ pour le premier scénario. L'heuristique CLH2 enregistre la plus grande dégradation sur les deux scénarios (3,6 et 2,9 %) par rapport à Var0 mais donne toujours de meilleures résultats que CLH1 qui a une dégradation inférieure (0,5 et 1,5 %).

On remarque aussi que CLH2 est la seule heuristique des trois à mieux performer avec le deuxième scénario. Une construction plus rigoureuse et méthodique des clusters permet donc de respecter plus facilement la distribution non uniforme des recharges initiales et en prendre avantage pour construire des tournées adéquates.

L'heuristique BIH garde donc son avantage malgré la baisse de la recharge initiale pour trouver les meilleures solutions au risque (très bas sur les tests faits) de ne pas trouver de solutions complètes. L'heuristique CLH2 est la plus robuste à cette baisse de recharge initiales et à ses différentes variantes puisqu'elle est la seule heuristique à n'avoir aucune solution incomplète. Contrairement au résultats précédents, l'heuristique CLH1 a le plus mauvais résultats sur le nombre de solutions incomplètes ce qui démontre l'avantage des insertions récursives pour une meilleure considération de la contrainte d'énergie.

Instances	BIH		CLH1		CLH2	
	C.n.v	**%Sol$^-$**	**C.n.v**	**%Sol$^-$**	**C.n.v**	**%Sol$^-$**
(2,4)	**10,43**	100%	10,54	100%	10,63	100%
(2,5)	1,09	47%	**0,03**	**3%**	0.04	4%
(2,6)	0,05	2%	**0**	**0%**	0,01	1%
(3,3)	5,42	99%	**1,52**	**56%**	1,77	57%
(3,4)	0,3	20%	**0,01**	**1%**	0,03	3%
(3,5)	0	0%	0	0%	0	0%
(5,3)	**29,13**	**100%**	31,18	**100%**	30,94	**100%**
(5,4)	3,65	98%	**0,07**	**6%**	0,19	14%
(5,5)	0,65	45%	**0,01**	**1%**	**0,01**	**1%**
Moyenne	5,64	57%	4,82	**30%**	**4,81**	31%

TABLE 3.16 – Évaluation comparative du (nombre de clients non visités, pourcentage de solutions incomplètes) par heuristique/groupe d'instances

	BIH	**CLH1**	**CLH2**
Var0	427,03	17,56	16,88
Var1	577,85	24,9	24,62
Var2	560,06	24,75	24,44

TABLE 3.17 – Comparatif du CPU(secondes) par heuristique et par variante

3.9 Conclusion

Dans ce chapitre, nous avons proposé la problématique de planification de tournées de véhicules électriques avec fréquence unique des clients. Nous avons modélisé le problème en programme linéaire en nombres entiers. Le problème étant NP-Complet, nous avons proposé trois heuristiques de résolution. Deux de ces heuristiques utilisent une politique "cluster first, route second", alors que la troisième est inspirée des heuristiques de meilleure insertion. Pour gérer la contrainte d'énergie, une procédure est proposée. Cette procédure permet de réparer des tournées non faisables en termes d'énergie.

Les trois heuristiques ont été développées puis testées sur une batterie d'instances. Les tests nous ont permis de constater l'efficacité de l'heuristique BIH sur des instances peu contraintes en termes de véhicules. Inversement, les heuristiques CLH1 et CLH2 sont plus efficaces pour visiter plus de clients sur les instances plus difficiles. Globalement, l'heuristique CLH2 permet d'avoir un bon équilibre entre vérification des contraintes et minimisation du coût de la solution.

	BIH	**CLH1**	**CLH2**
Var0	0	0	0
Var1	0	0,01	0
Var2	0,01	0,01	0

TABLE 3.18 – Comparatif du nombre de clients non visités par heuristiques et par variantes

	BIH	**CLH1**	**CLH2**	**Moyenne**
Var0	2676,67	3017,86	2878,17	2857,57
Var1	2700,19	3033,68	2982,97	2905,61
Var2	2716,75	3063,98	2960,93	2913,89

TABLE 3.19 – Comparatif du coût complet par heuristiques et par variantes

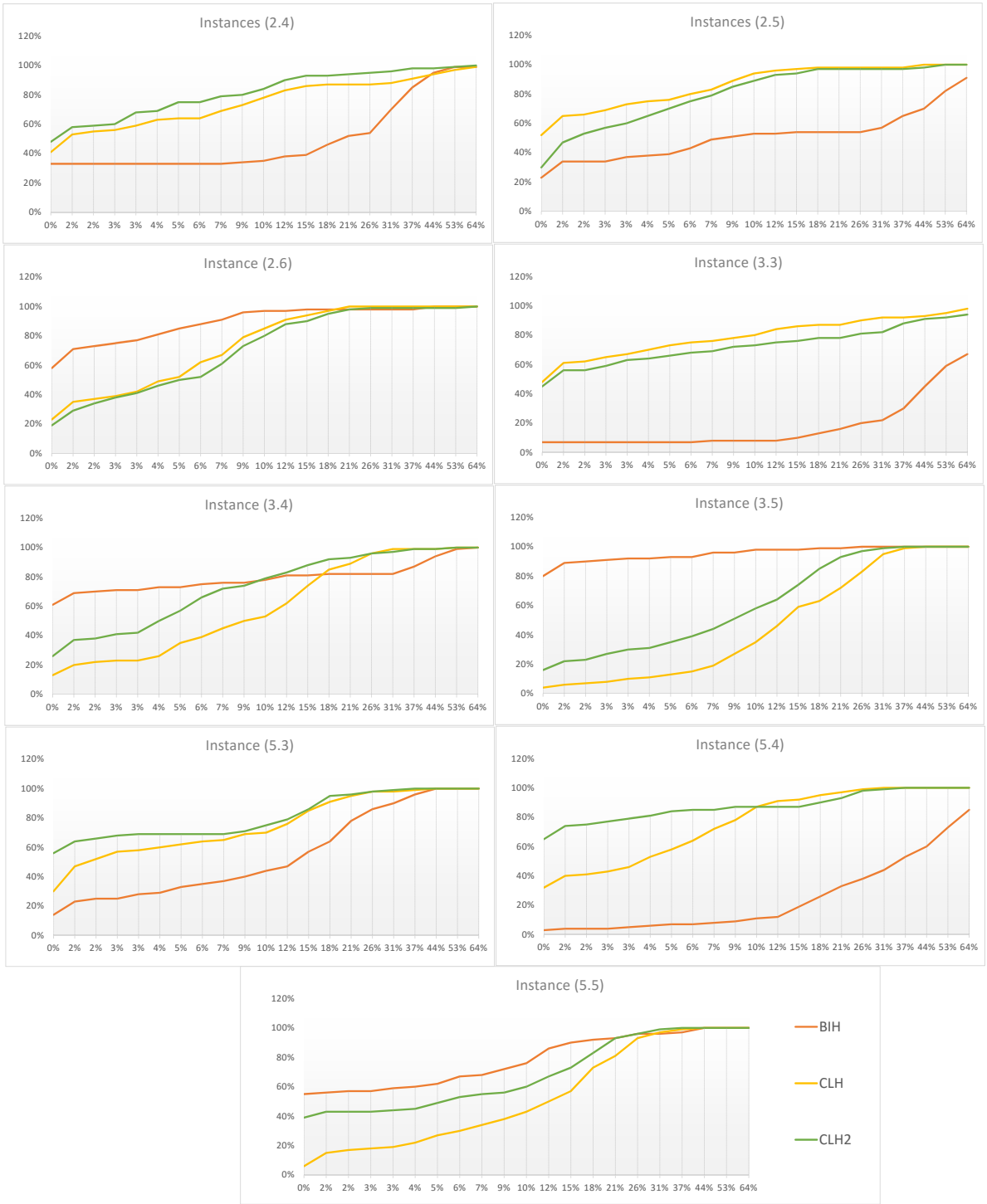


FIGURE 3.24 – Performance profiles par sous-catégories d'instances

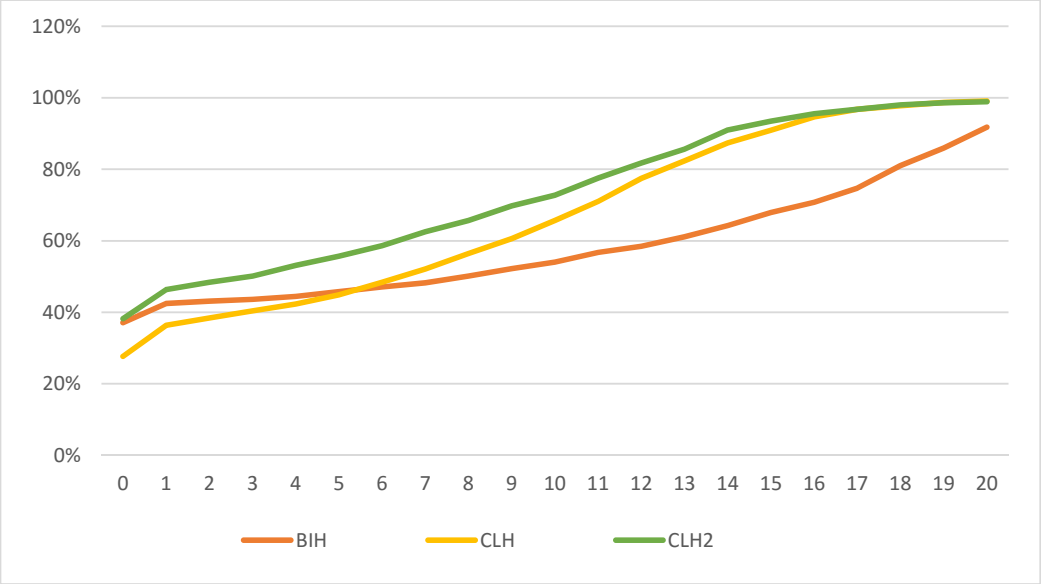


FIGURE 3.25 – Performance profiles sur toute les instances

Chapitre 4

Problème de planification des tournées de véhicules électriques avec mono-visite sur l'horizon : métaheuristiques

Sommaire

4.1 Introduction

Nous avons étudié dans le chapitre précédent le problème PEVRP et la difficulté de trouver des solutions complètes avec des heuristiques fiables. Dans ce chapitre nous développons des métaheuristiques de type de recherche à large voisinage (Large Neighborhood Search LNS) et de recherche adaptative à large voisinage (Adaptive Large Neighborhood Search ALNS).

La ALNS a d'abord été proposé par [89] et peut être décrit comme un algorithme de recherche de grand voisinage avec une couche adaptative, où un ensemble d'opérateurs de suppression/insertion se font concurrence pour modifier la solution courante à chaque itération de

l'algorithme. Par rapport au LNS, l'ALNS gère plusieurs opérateurs et choisit les deux opérateurs (suppression, insertion) à appliquer à chaque itération selon des critères de performances. Par cette gestion interne de différents opérateurs, la méthode apporte des modifications plus diversifiées à la solution courante en explorant un espace de recherche encore plus grand.

Dans ce chapitre nous proposons des opérateurs de suppression et d'insertion nécessaires pour l'algorithme LNS et ALNS. Ces opérateurs utilisent principalement deux fonctions pour faire face aux contraintes énergétiques et de recharge.

La première fonction, appelée $AdjustDecreaseCharging(Tr)$ est appliquée sur chaque tour Tr où des clients ont été supprimés par un opérateur de destruction. Le but de cette fonction est d'estimer l'énergie inutilisée de la solution modifiée Tr, et de décider, quand c'est possible, quelles stations devront réduire leur recharges, et/ou quelles stations doivent être supprimées de la tournée.

La deuxième fonction appelé $repar(Tr)$, définit dans le chapitre précédent (Section 3.4), est utilisée pour chaque tour modifié Tr après avoir inséré un nouveau client. Cette procédure permet de vérifier si une tournée respecte la contrainte d'énergie et tente de la réparer sinon en injectant de l'énergie (augmentation de la recharge existante, insertion de stations de recharges). Cette fonction est utilisée dans les opérateurs d'insertion.

4.2 Métaheuristiques à base de grand voisinage pour les problèmes de véhicules électriques

4.2.1 Recherche à large voisinage LNS

La LNS est une métaheuristique de recherche de grand voisinage (Large Neighborhood Search, LNS). LNS a d'abord été proposé par [97], puis adapté par [41]. La LNS part d'une solution réalisable donnée et l'améliore à l'aide de la stratégie destruction-construction en utilisant un opérateur de suppression et un opérateur d'insertion. En effet, LNS retire un nombre relativement important de clients de la solution courante et réinsère ces clients dans des positions différentes. Cela conduit à une solution différente et permet d'échapper à l'optimum local.

A chaque itération, l'algorithme du LNS va supprimer, par l'opérateur de suppression, un nombre fixe de clients de la solution courante, puis réinsérer ces clients dans la solution selon un opérateur d'insertion. Ces deux étapes sont répétées jusqu'à atteindre le critère d'arrêt.

L'organigramme 4.1 présente le fonctionnement général du LNS. La métaheuristique commence par générer la solution initiale S_0 qui sera considérée comme solution courante S initialement. A chaque itération et jusqu'à atteindre le critère d'arrêt, une solution S_1 est construite par l'application d'un opérateur de destruction puis de construction sur la solution courante S. Cette solution S_1 est considérée comme la nouvelle solution courante si elle passe le critère d'acceptation ($Accepter(S_1, S)$?). Initialement dans les travaux [97], le critère prenait seulement les solutions améliorantes. On trouve d'autres critères d'acceptation dans les travaux ultérieurs. La solution S^* est la meilleure solution atteinte par la métaheuristique et sera la solution retournée à la fin de l'algorithme.

La LNS est une métaheuristique fréquemment utilisé sur les problèmes de transport pour ses bons résultats et pour sa capacité à explorer un voisinage plus large avec des opérateurs simples et faciles à adapter aux différents problèmes et à leurs caractéristiques. Cette conclusion reste plausible pour les problèmes de transport impliquant des véhicules électriques.

Dans [91], une métaheuristique LNS est utilisée pour résoudre un problème de tournées de véhicules avec une flotte mixte de véhicules thermiques et électriques. Les auteurs dans [91], ont

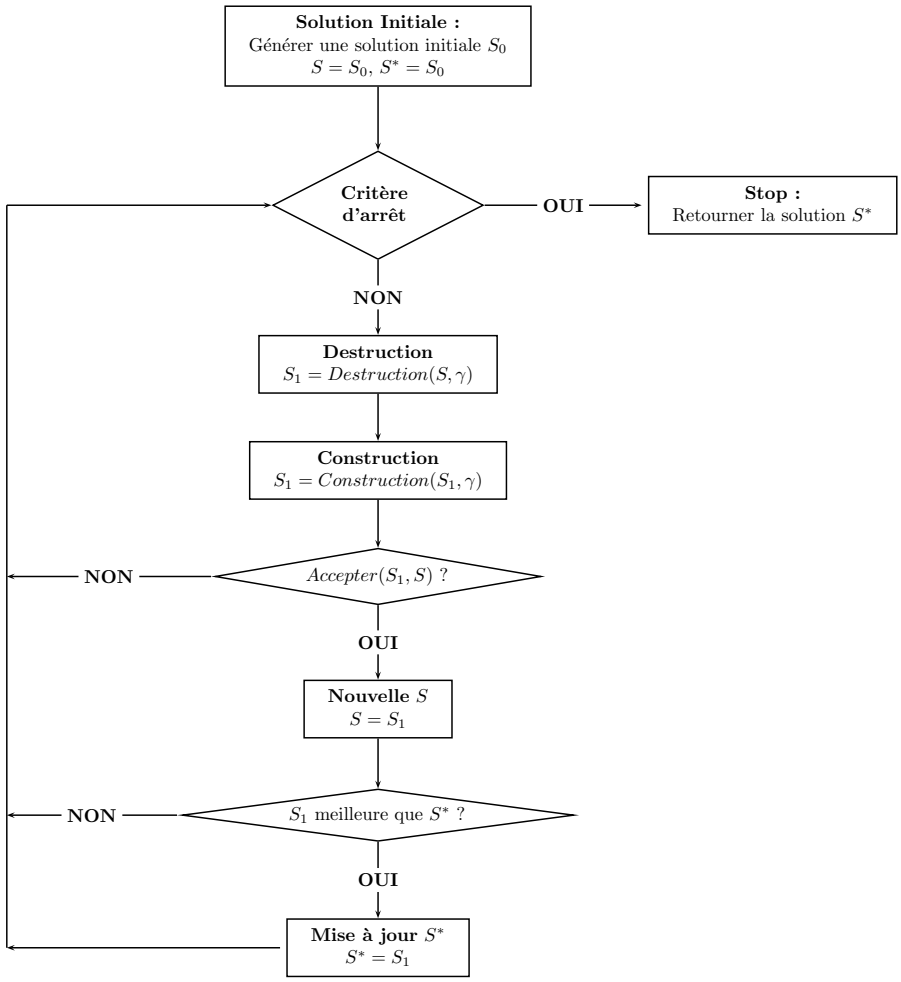


FIGURE 4.1 – Organigramme LNS

initialement proposé une méthode LNS avec :

- un critère d'acceptation classique où une solution n'est acceptée que si elle est meilleure que la solution courante
- un opérateur de destruction qui éjecte un nombre précis de sommets de la solution, clients et stations de recharge confondus.
- trois opérateurs de construction, chacun réinjectant les clients supprimés selon des politiques d'insertion différentes

L'objectif de ce travail était de tester la méthode LNS avec différents paramètres (choix de l'opérateur de construction, nombre de sommets éjectés, temps de calcul..) et de proposer une méthode permettant d'améliorer une solution donnée. Les résultats furent probants puisque la LNS a permis d'améliorer la qualité des solutions générées de plus de 50% dans la majorité des instances et ceci en un temps de calcul raisonnable.

Dans un deuxième temps, les auteurs de [91] ont proposé deux méthodes de résolution hybrides, chacune considérant un algorithme LNS comme voisinage dans une méthode de recherche local (ILS, VNS). Pour tester ces méthodes hybrides, elles ont été comparées à une méthode classique LNS et les résultats ont montré que la méthode LNS classique donne de meilleurs résultats

dans la majorité des instances testées.

Dans [74] un problème de tournées de véhicules électriques avec incertitude sur la consommation d'énergie est étudié. L'objectif est de trouver une solution robuste minimisant les différents coûts. Une solution robuste étant une solution composée de tournées, appelées robustes, pouvant être réalisées, en termes de consommation d'énergie, avec une probabilité forte par un véhicule électrique sans effectuer de recharges pendant la tournée. Les auteurs proposent une résolution en deux phases, une première phase basée sur un algorithme LNS pour obtenir un ensemble de tournées robustes, la deuxième phase construit à partir de l'ensemble obtenu, la meilleure solution possible. Dans la première phase, l'algorithme LNS est lancé sans critère de robustesse sur les solutions, chaque nouvelle solution obtenue est analysée pour repérer les tournées robustes et les stocker dans un ensemble. Un problème de partition (Set Partitioning Problem) est résolu sur l'ensemble des tournées robustes pour construire la meilleure solution possible. Les auteurs utilisent une variante de l'algorithme LNS où plusieurs opérateurs (suppression, insertion) sont utilisés. Chacun des opérateurs a une probabilité d'être choisi au début de chaque itération du LNS, cette probabilité ne change pas durant l'algorithme (contrairement à l'algorithme ALNS présenté ultérieurement).

4.2.2 Recherche adaptative à large voisinage ALNS

La méthode ALNS est une variante du LNS proposée initialement dans [89]. La ALNS a une fonctionnalité d'adaptation permettant de gérer et d'utiliser un ensemble d'opérateurs de suppression et d'insertion, contrairement au LNS qui fonctionne avec seulement un opérateur de chaque (suppression, insertion). Par rapport au LNS, l'ALNS apporte des modifications plus diversifiées à la solution en explorant un espace de recherche encore plus grand.

Les opérateurs de suppression (respectivement d'insertion) se font concurrence pour modifier la solution courante selon leurs historiques de performances. Chaque opérateur o a une probabilité d'être choisi calculée selon son poids w_o. Le poids w_o est calculé selon la performance passée de l'opérateur o pour favoriser les opérateurs ayant été les plus efficaces.

L'algorithme ALNS est fractionné en plusieurs segments, chaque segment est un nombre fixé Seg d'itérations. À chaque segment j, le poids w_o^j est actualisé selon la performance de l'opérateur o dans le segment $j-1$. Ce fractionnement en segments permet à tous les opérateurs d'avoir une chance d'être utilisé avant d'actualiser leurs poids et donc leurs chances d'être réutilisés.

Pour valoriser la performance des opérateurs dans un segment, l'ALNS calcule un score θ_o pour chaque opérateur dans chaque segment. À chaque début de segment, les scores sont initialisées à zéro, puis à chaque fois qu'une opération de destruction/réparation est effectuée, les scores des opérateurs concernés sont augmentés. Le score peut être augmenté de :

- σ_1 si une nouvelle et meilleure solution globale est trouvée,
- σ_2 si une nouvelle solution est trouvée et que cette solution est de meilleure qualité que la solution courante,
- σ_3 si une nouvelle solution est trouvée et que cette solution est plus mauvaise que la solution courante

Le poids w_o^{j+1} d'un opérateur o dans un segment $j+1$ est donc calculé selon son poids précédent w_o^j et son score θ_o dans le segment j selon la formule suivante :

$w_o^{j+1} = w_o^j \times (1-\lambda) + \lambda \times \frac{\theta_o}{\pi_o}$, où λ est un paramètre contrôlant la balance d'influence entre le score θ_o et le poids précédent w_o^j, π_o représente le nombre de fois que l'opérateur o a été utilisé au cours du segment j.

La sélection des opérateurs est ensuite effectuée, à chaque itération du segment $j+1$, en utilisant le principe de la sélection par la roulette. Ainsi, l'opérateur o, qui a un poids élevé,

aura plus de chances d'être sélectionné. Plus précisément, la probabilité de sélectionner un tel opérateur est de $(w_o^{j+1}/\sum_{a\in O} w_a^{j+1})$, où O est l'ensemble de tous les opérateurs du même type (suppression ou insertion).

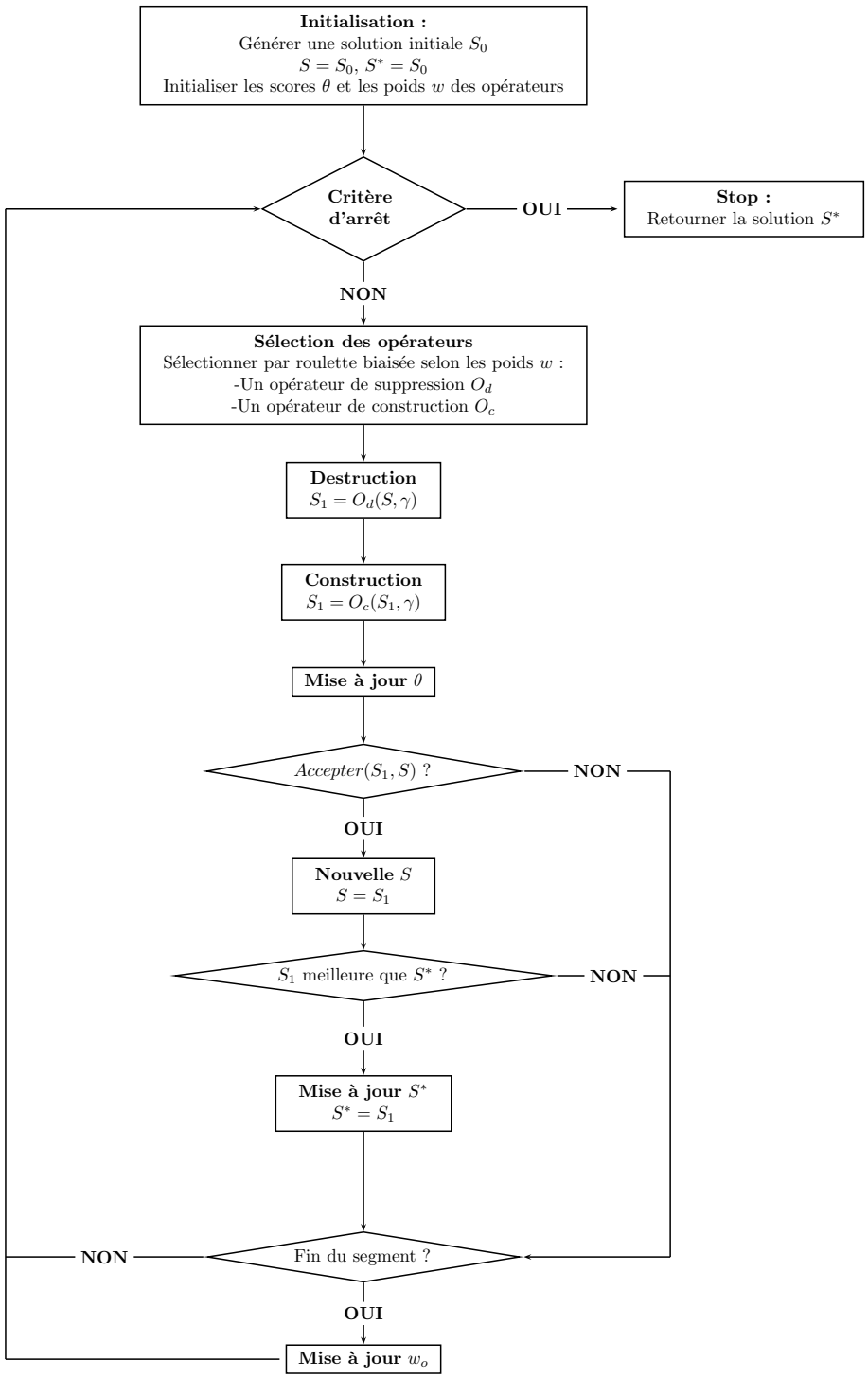


FIGURE 4.2 – Organigramme ALNS

La Figure 4.2 présente le fonctionnement général du ALNS. La métaheuristique commence par générer la solution initiale S_0 et initialiser les scores des opérateurs θ et le poids des opérateurs w_o. A chaque itération, un opérateur de destruction et un opérateur de construction sont tirés au sort par roulette biaisée puis appliqués sur la solution S pour obtenir la solution S_1. Selon la qualité de la solution, les opérateurs utilisés sont augmentés du σ correspondant. Le critère d'acceptation permet de choisir si la solution S_1 sera la nouvelle solution courante S. A la fin de chaque segment, les poids des opérateurs sont mis à jour selon leurs poids précédents et leurs scores sur le segment précédent.

Dans [51] les auteurs étudient un problème de transport avec véhicules électriques à deux échelons, le premier échelon portant sur le transport de marchandises entre le dépôt et des sites satellites, le deuxième sur le transport des marchandises entre les sites satellites et les clients. Lors du premier échelon, des véhicules avec une capacité (marchandises) et une autonomie (batterie) plus larges sont utilisés puis des véhicules plus petits et moins autonomes pour le deuxième échelon. Pour répondre à l'éventualité de manque d'autonomie, les auteurs considèrent des stations où les véhicules peuvent remplacer leurs batteries par une batterie complètement chargée. Pour résoudre ce problème, les auteurs résolvent le deuxième échelon qui correspond à un problème de transport à plusieurs dépôts (les sites satellites) avec une métaheuristique ALNS puis résolvent le premier échelon selon la solution du deuxième échelon avec une méthode exacte de génération de colonnes. Pour la métaheuristique ALNS appliquée lors du deuxième échelon, les auteurs acceptent les solutions irréalisables pour permettre une diversification lors de l'exploration tout en pénalisant ces solutions irréalisables pour garantir l'obtention d'une solution réalisable finalement. Les auteurs proposent huit opérateurs de destruction, deux de ces opérateurs choisissent directement les clients à supprimer, trois opérateurs modifient les satellites (ouvrir, fermer..) puis suppriment les clients affectés, et trois autres opérateurs suppriment des routes entières selon l'utilisation considérée non optimale des stations d'échange de batteries. Six opérateurs de construction sont proposés. Pour permettre à tous les opérateurs d'être utilisés avant d'actualiser leurs scores, les auteurs actualisent les scores des opérateurs au début de chaque segment, le segment étant un nombre fixé d'itérations, comme proposé dans [89].

4.3 Considération de la contrainte d'énergie

Nous avons présenté dans le chapitre précédant la procédure $Repar(Tr)$, cette procédure permet de rendre réalisable une tournée ne respectant pas la contrainte d'énergie. Cette procédure sera utilisée par les opérateurs d'insertion dans les algorithmes LNS et ALNS.

Nous détaillerons ici une autre procédure, la procédure $AdjustDecreaseCharging(Tr)$ qui sera appliquée à une tournée après l'opérateur de suppression.

Soit Tr une tournée réalisable et Tr^- une copie de Tr dont une partie des clients a été annulée ou supprimée. La tournée Tr^- reste réalisable mais peut contenir des passages par des stations recharges qui ne sont plus nécessaires. La suppression de ces visites superflues permettrait d'optimiser la tournée en réduisant le coût de la tournée et sa durée. La procédure $AdjustDecreaseCharging(Tr)$ s'applique à une tournée pour examiner les visites aux stations de recharge, puis réduire les recharges non nécessaires. Cette réduction de la recharge peut se traduire par la suppression de visites aux stations de recharge et/ou la réduction du temps de recharge dans les stations nécessaires.

Notre objectif est donc de supprimer les visites non nécessaires aux stations de recharges, puis de réduire au maximum la recharge des stations restantes. Il est évident que la suppression d'une station de recharge est plus intéressante qu'une simple réduction de recharge, puisqu'elle

permet un plus grand gain en termes de coût et de temps.

La fonction *AdjustDecreaseCharging* (voir Algorithme 12) reçoit en entrée la tournée Tr où des clients ont été supprimés. L'objectif de l'algorithme est d'ajuster les recharges d'une tournée, en supprimant les visites aux stations superflues, puis en réduisant méthodiquement les recharges des stations restantes.

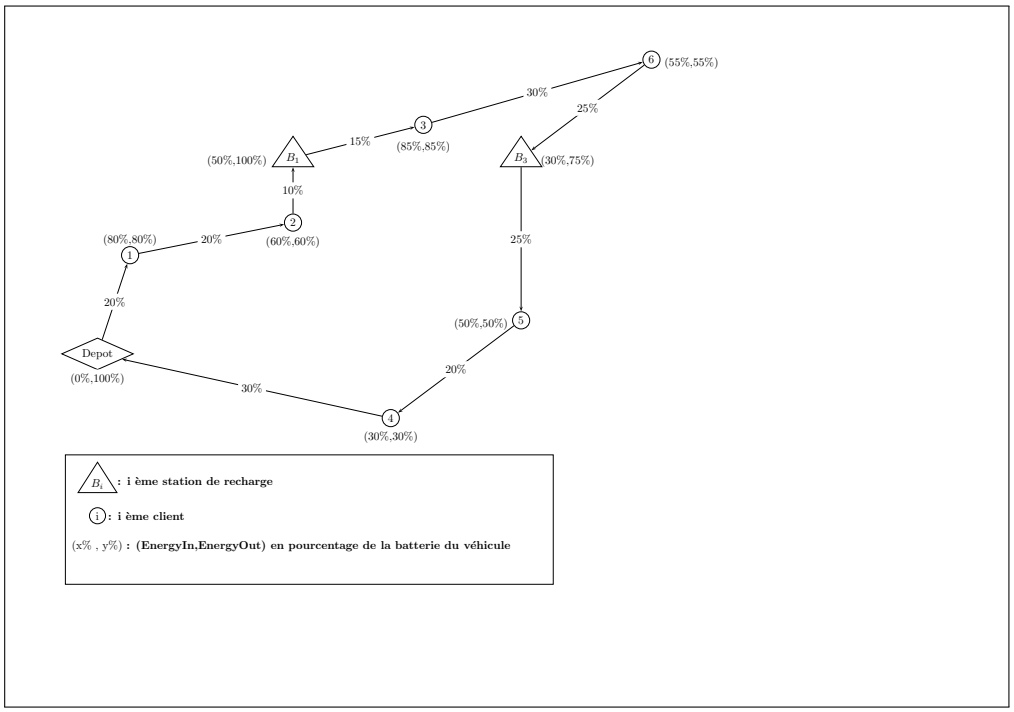


FIGURE 4.3 – Exemple d'une tournée avant suppression de clients

Pour illustrer cette fonction, nous avons pris une tournée (Figure 4.3) dont on va supprimer différents clients pour obtenir trois exemples de tournées chacune avec un surplus d'énergie. La premier tournée (Figure 4.4) résulte de la suppression du client (1), la deuxième tournée (Figure 4.5) résulte de la suppression du client (4), et le troisième exemple (Figure 4.6) résulte de la suppression des clients (1),(4) et (6).

La suppression des clients engendre évidemment un réajustement du niveau de batterie à l'arrivée et au départ de chaque sommet restant dans la tournée (*EnergyIn* et *EnergyOut*). Avant le réajustement nous allons mémoriser la quantité d'énergie rechargée dans chaque station b, que nous appellerons $Recharge(b)$. Puis le réajustement se fait en partant du dépôt ($\bar{0}$) avec une recharge entière et par paire de sommets se suivant, le $EnergyIn$ du sommet suivant est calculé selon la consommation d'énergie de l'arc le reliant à son précédent. Pour les clients et les deux copies du dépôt ($\bar{0}$ et $\underline{0}$), le niveau de la batterie au départ ($EnergyOut$) est logiquement égale au niveau de la batterie à l'arrivée ($EnergyIn$). La valeur de ($EnergyOut$) d'une station b est calculée pour que la quantité de recharge $Recharge(b)$ reste la même sans dépasser la capacité de la batterie $MaxEnergy$ (formule 4.1). On voit dans les exemples que par cette limite de charge d'une batterie ($MaxEnergy$) la quantité de recharge baisse dans les stations de recharge.

Dans le premier exemple (Figure 4.4), le réajustement induit par la limite de charge d'une batterie ($MaxEnergy$) suffit à réduire efficacement les recharges puisque le véhicule revient avec une batterie déchargée au dépôt. Dans ce cas là, la procédure $AdjustDecreaseCharging(Tr)$ ne modifie pas la tournée.

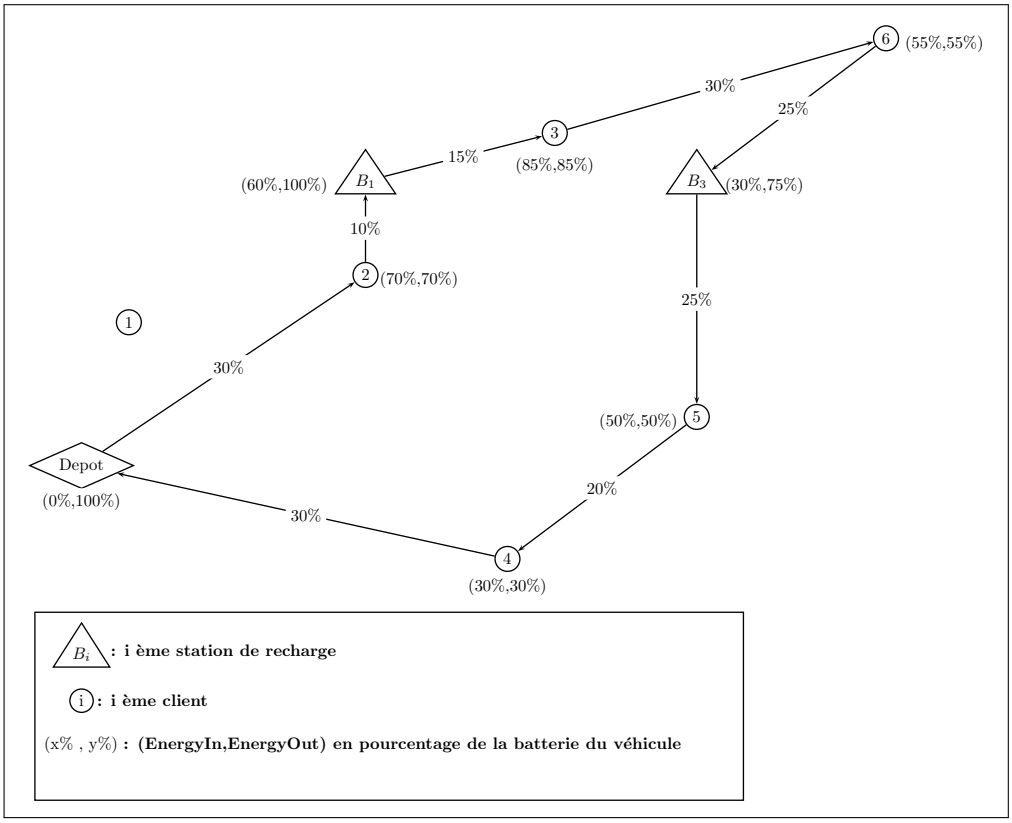


FIGURE 4.4 – Premier exemple d'une tournée après suppression de clients

Dans les deux autres exemples, le véhicule ayant encore de la recharge en revenant au dépôt, il est donc intéressant d'appliquer la procédure $AdjustDecreaseCharging(Tr)$.

$$EnergyOut(b) = \min(MaxEnergy, EnergyIn(b) + recharge(b)) \tag{4.1}$$

L'Algorithme 12 commence par vérifier que la tournée contient des stations de recharge, sinon il retourne la tournée telle quelle (ligne 3). Puis l'algorithme parcourt la tournée à la recherche des stations de recharge dont la suppression ne fera pas violer la contrainte d'énergie. Nous considérons qu'une station b est supprimable si le véhicule peut atteindre la prochaine station ou dépôt sans recharger à b. À chaque station supprimable trouvée (ligne 10), il calcule le gain de coût obtenue par cette suppression. La station dont la suppression permet le plus grand gain est supprimée, puis la tournée est actualisée (ligne20). Cette procédure est répétée jusqu'à ce l'algorithme ne trouve plus de stations à supprimer.

Une fois toutes les stations non nécessaires supprimées, l'algorithme atteint la dernière station visitée et y réduit la recharge pour arriver au dépôt avec un niveau de batterie nul (ligne 26). Réduire la recharge sur les autres stations n'est pas efficace puisque cela provoquera juste un effet de compensation sur la prochaine station.

Dans le deuxième exemple (Figure 4.5), la suppression de la station (B_1) ou de la station (B_3) n'est pas possible puisque ça rendrait la tournée irréalisable en termes de contrainte d'énergie. L'algorithme réduit donc la recharge à la dernière station (B_3) de 10% pour que le véhicule arrive au dépôt avec un niveau de batterie nul (figure 4.7).

Dans le troisième exemple (Figure 4.6, l'algorithme commence par lister les stations supprimables, ici les deux stations sont supprimables. en termes de coût (principalement coût de

Algorithm 12 *AdjustDecreaseCharging*

Input : Tournée Tr après la suppression des clients
Output : Tournée Tr Mise à jour.
if $(PrecStation(\underline{0}) = \bar{0})$ **then**
    Retourner Tr ▷ Pas de stations de recharge dans Tr
end if
while $(GetMinEnergy(Tr) \neq 0)$ **do**
    $b^* := \underline{0}$, $MinCost := \infty$
    $u := \underline{0}$, $b := PrecStation(u)$
    while $(b \neq \bar{0})$ **do** ▷ Recherche d'une station à supprimer
        if $(Recharge(b) <= EnergyIn(u))$ **then** ▷ Station supprimable
            $x = prec(b)$, $y = succ(b)$
            $Cost = c_{xb} + c_{by} - c_{xy} + Cc$
            if $(Cost < MinCost)$ **then**
                $b^* := b$, $MinCost := Cost$
            end if
        end if
        $u := b$;$b := PrecStation(u)$
    end while
    if $(MinCost \neq \infty)$ **then** ▷ Possible de supprimer une station
        Supprimer b^* ; $UpdateCharging(Tr)$.
    else
        if $(PrecStation(\underline{0}) = \bar{0})$ **then**
            Retourner Tr ▷ Plus de stations de recharge dans Tr
        else
            $b := PrecStation(\underline{0})$
            $EnergyOut(b) := EnergyOut(b) - EnergyIn(\underline{0})$
            $UpdateCharging(Tr)$
        end if
    end if
end while

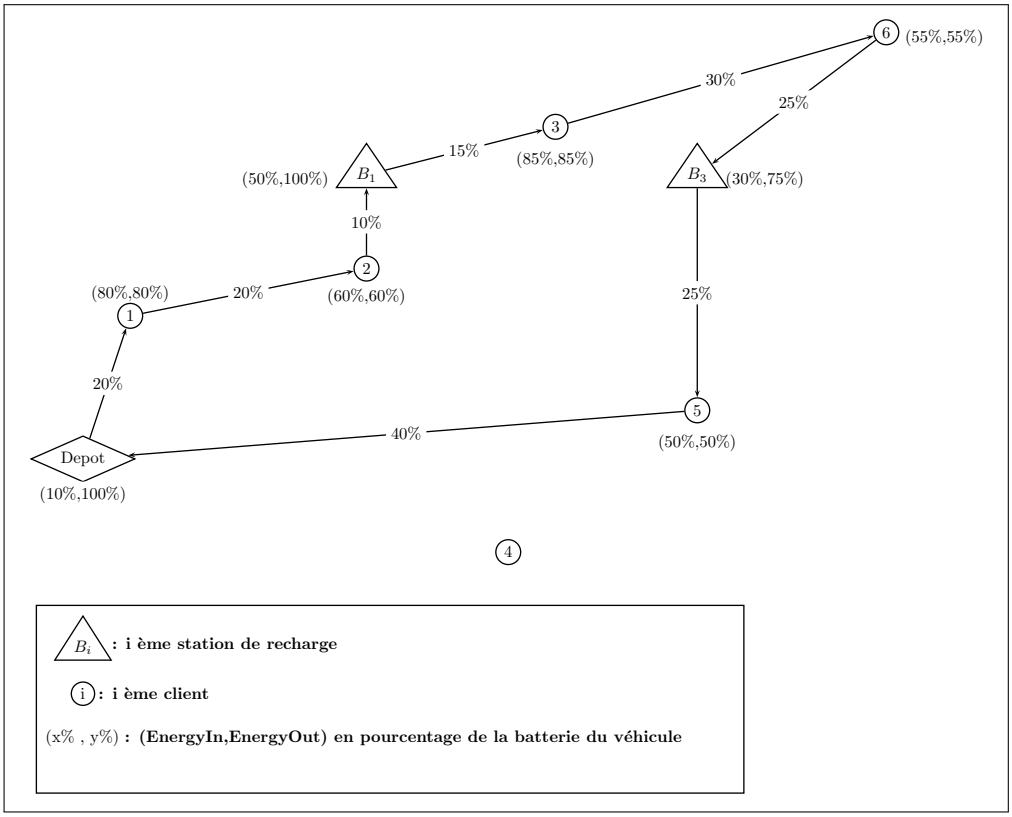


FIGURE 4.5 – Deuxième exemple d'une tournée après suppression de clients

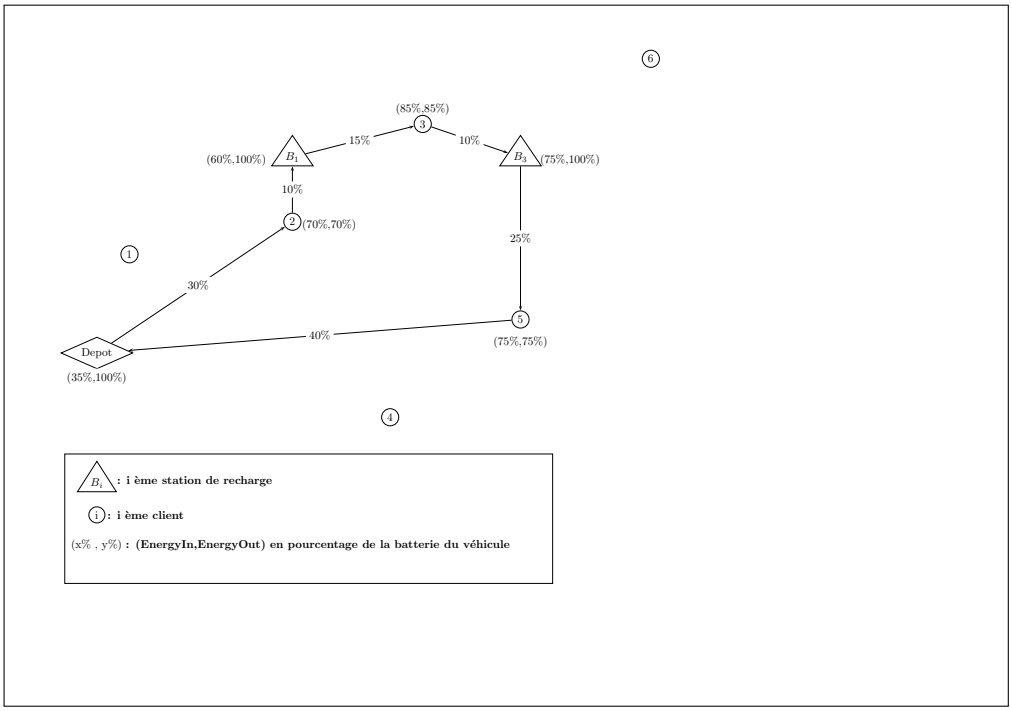


FIGURE 4.6 – Troisième exemple d'une tournée après suppression de clients

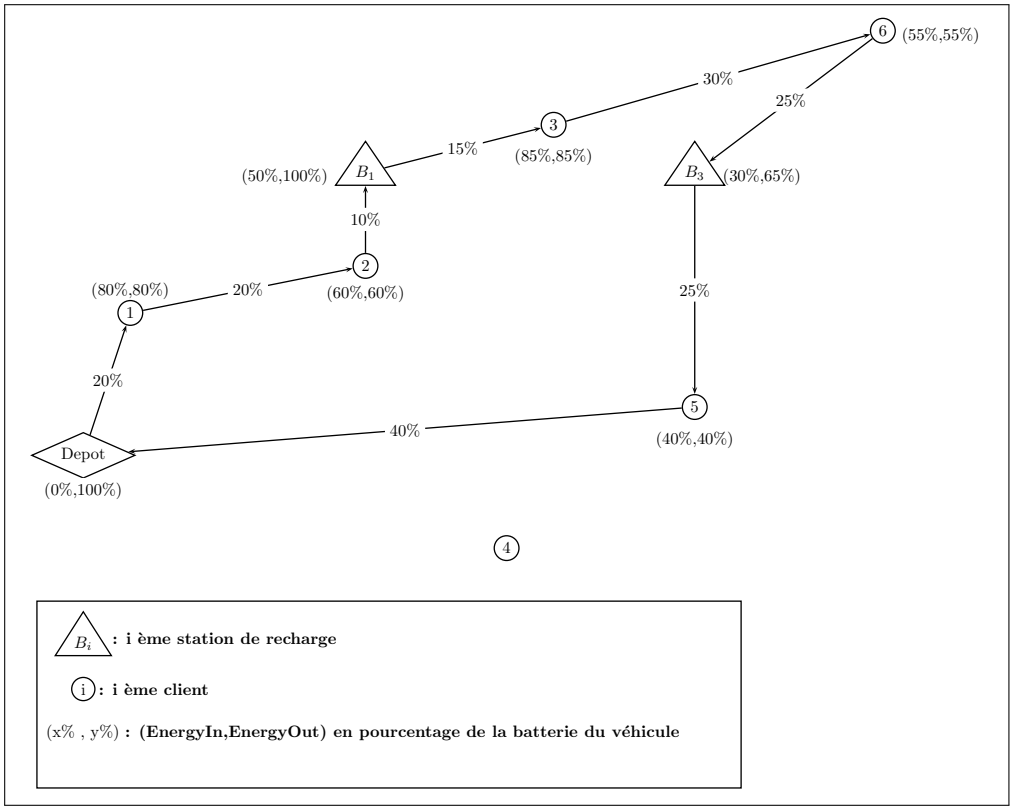


FIGURE 4.7 – Deuxième exemple : réduction des recharges

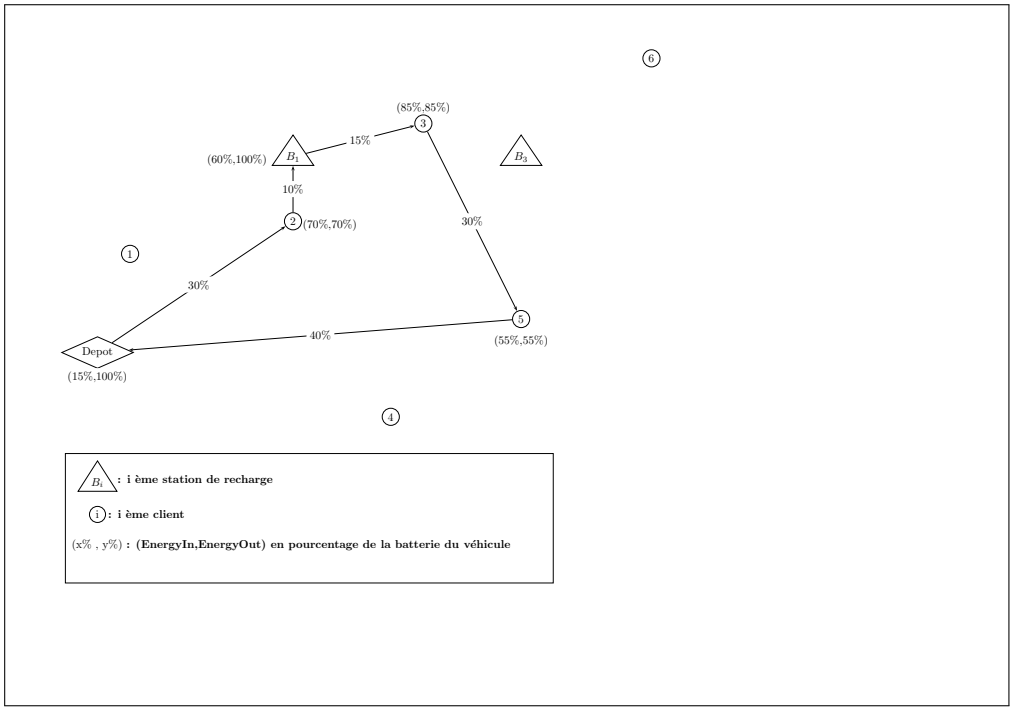


FIGURE 4.8 – Troisième exemple : suppression de stations de recharge

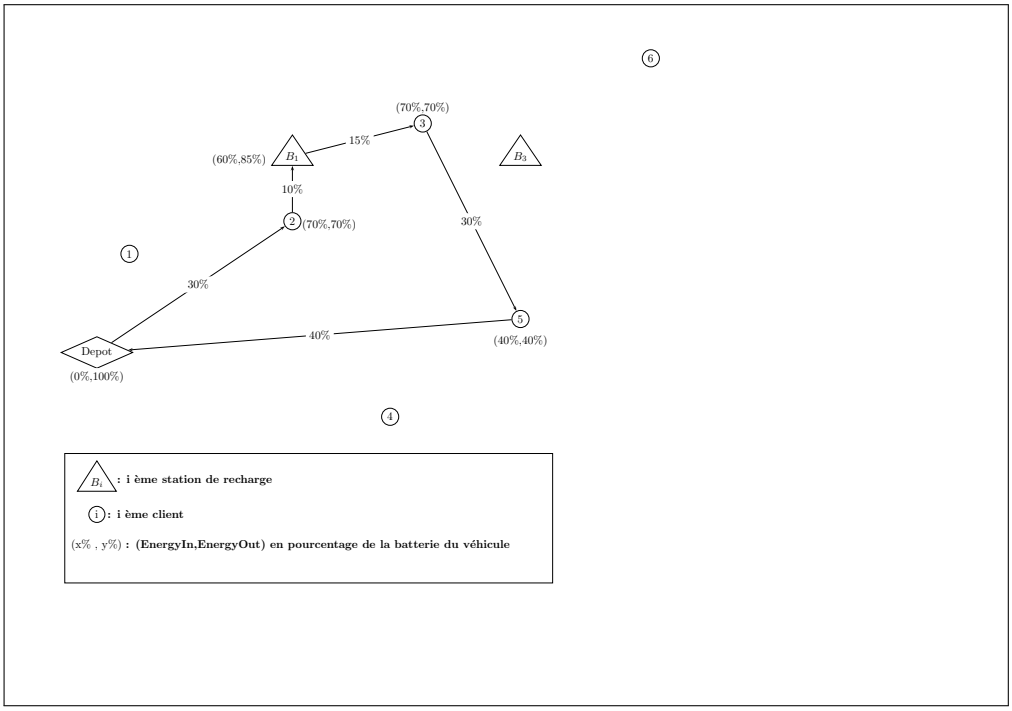


FIGURE 4.9 – Troisième exemple : réduction des recharges

détour), il est plus intéressant de supprimer la station B_3 ce qui donne la tournée dans la Figure 4.8. Puis sur la tournée résultante (Figure 4.8), la recharge à la station B_1 est réduite de 15% pour que le véhicule arrive avec un niveau de 0% à $\underline{0}$ (figure 4.9).

4.4 Principe de fonctionnement de notre LNS

Pour notre problème, nous avons adapté la métaheuristique LNS décrite précédemment en changeant le critère d'acceptation et en utilisant la fonction de coût complet pour évaluer la solution.

L'utilisation de la fonction de coût complet permet à l'algorithme d'envisager les solutions irréalisables comme solutions courantes tout en les pénalisant. Ce choix est justifié par une volonté de diversification dans l'exploration de l'espace des solutions.

Nous proposons d'utiliser le principe du recuit simulé comme critère d'acceptation. Ainsi, une solution est acceptée si elle est meilleure que la solution courante ou si la règle de Metropolis est respectée. En d'autres termes, les mauvaises solutions sont acceptées avec la probabilité donnée par $\exp^{\frac{cout.c(s)-cout.c(s')}{T}}$. où $cout.c(.)$ représente la fonction de coût complet, s' et s sont respectivement la nouvelle et la solution courante, et $T > 0$ indique la température actuelle. A chaque itération, la température actuelle T est diminuée en utilisant la vitesse de refroidissement $cooling \in]0,1[$ selon l'expression $T = T \times cooling$ Nous décidons d'initialiser la température de telle sorte qu'aux premières itérations du LNS, une solution 20% pire que la précédente soit acceptée avec une probabilité de 0.5. La vitesse de refroidissement est initialisée à $0,9995$.

Nous lancerons l'algorithme LNS trois fois, chacune avec la solution d'une des trois heuristiques (BIH,CLH1 et CLH2). Ce choix nous permet à la fois de diversifier la solution initiale, mais aussi de juger quel type de solution est la plus efficace comme solution initiale. La LNS

commence donc par appeler l'une des trois heuristiques de construction que nous avons proposées dans le chapitre précédent.

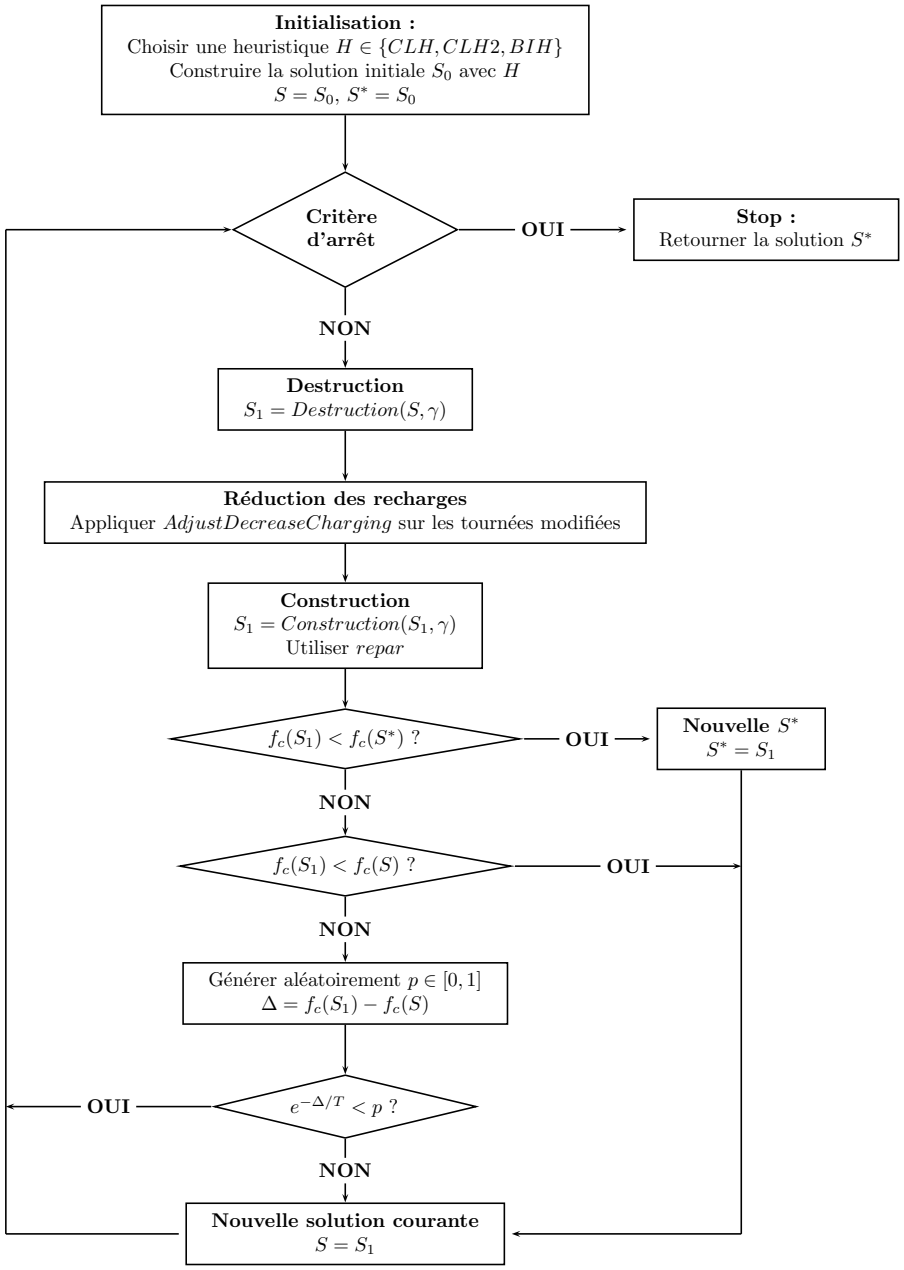


FIGURE 4.10 – Organigramme du LNS utilisé

La Figure 4.10 explicite l'algorithme LNS appliqué à notre problème. L'algorithme commence par construire une solution initiale S_0 avec les trois heuristiques $(CLH, CLH2, BIH)$, cette solution est considérée comme la solution courante S. L'opérateur de destruction est appliqué sur la solution courante S puis la procédure $AdjustDecreaseCharging$ est appliquée sur toutes les tournées où des clients ont été supprimés. Les clients supprimés sont réeinsérés dans la solution à

l'aide de l'opérateur de construction, opérateur qui utilisera la procédure *Repar* pour les tournées violant la contrainte d'énergie. Si la solution S_1 a un moindre coût complet (f_c) que la solution S, elle est automatiquement acceptée et considérée comme solution courante. Sinon, le critère d'acceptation permet de choisir entre les deux solutions.

4.4.1 Opérateurs de destruction

Les opérateurs de destruction suppriment un nombre donné de clients (notés γ) de la solution courante. Nous proposons trois opérateurs de destruction inspirés des opérateurs de suppression présentés dans [41]. Ces opérateurs sont le random removal, le worst removal, et le cluster removal. Dans ce qui suit, nous donnons une description de ces opérateurs.

Soit $S = \{S_1, ..., S_h\}$ la solution initiale, avec S_d l'ensemble des tournées de la journée d, $S_d = \{T_{1d}...T_{kd}...T_{md}\}$. Après avoir supprimé des clients γ, une nouvelle solution partielle S' est générée. $S' = \{T'_{1d}...T'_{kd}, ..., T'_{md}\}$, tel que $T'_{kd} = T_{kd}$ si aucun client n'a été supprimé de T_{kd} et $T'_{kd} = T^-_{kd}$ si un ou plusieurs clients ont été supprimé de T_{kd}.

Random removal $(S, S', f(S'))$

Cet opérateur supprime aléatoirement les clients γ de la solution. La procédure $AdjustDecreaseCharging(Tr)$ est appliquée après la suppression de tous les clients sur les tournées où des clients ont été supprimés.

Cluster removal $(S, S', f(S'))$

Cet opérateur commence par choisir aléatoirement un client i, puis $\gamma - 1$ des clients supplémentaires les plus proches de i (en termes de distances) sont sélectionnés (les voisins sélectionnés peuvent être sur des tournées différentes et des jours différents et ne sont pas forcément dans $trip(i)$, sachant que $trip(i)$ est le tour qui inclut i). La procédure $AdjustDecreaseCharging(Tr)$ est appliquée à chaque tournées impactées par l'opérateur $T^-_{kp}. \in S'$.

Worst removal $(S, S', f(S'))$

Pour chaque client $i \in S$, l'algorithme simule la suppression de i, en supprimant i de S, et en appliquant $AdjustDecreaseCharging(trip(i)^{-i})$. Le coût simple $Cost.s(trip(i)^{-i})$ est calculé pour chaque simulation de suppression. Le client i^* dont la suppression produit la plus grande baisse $(Cost.s(trip(i)) - Cost.s(trip(i)^{-i})$ est choisie. L'algorithme répète la procédure jusqu'à supprimer γ clients.

4.4.2 Opérateurs d'insertion

Nous avons mis en place trois méthodes d'insertion, à savoir, First Improvement, Best Improvement et Regret Insertion. Nous appliquons $Repar(Tr)$ pour ajuster la recharge dans chaque tournée violant la contrainte d'énergie après l'insertion d'un client. S'il n'est pas possible d'insérer un sommet éjecté dans une tournée déjà construite, une nouvelle tournée contenant ce sommet et le dépôt peut être créé si le nombre de véhicules utilisés est inférieur à m. Lorsque tous les clients ont été réinsérés dans la solution en utilisant une méthode d'insertion prédéfinie, la nouvelle solution est comparée à la solution originale par la fonction de coût complet. La solution est considérée même si elle ne visite pas tous les clients. Nous considérons deux stratégies d'insertion d'un client i dans une tournée Tr :

— $InsertC^{All}$ qui teste l'insertion du client i à toutes les positions de la tournée $Tour$, nous avons donc $|Tr| - 1$ tests d'insertion.
— $InsertC^{2}$ qui choisit les deux clients visités les plus proches de i dans la tournée Tr, et teste l'insertion du client i avant et après chacun d'eux.

Le croisement entre les trois méthodes d'insertion et les deux stratégies d'insertion, nous donne donc six opérateurs d'insertion. Dans ce qui suit nous expliquons les trois méthodes d'insertion.

First Improvement insertion

Cette méthode sélectionne un sommet dans la liste L des sommets éjectés et l'insère dans la tournée qui génère l'augmentation minimale des coûts. L'insertion du client i est donc testée dans toutes les tournées de ces jours de disponibilité $D(i)$. Si l'insertion du client i dans une position donnée d'une tournée t entraîne une violation de la capacité du véhicule ou des contraintes de temps total, cette insertion ne sera pas considérée. Cependant, si l'insertion d'un client dans une position d'une tournée donnée satisfait les contraintes de capacité du véhicule et de temps total mais entraîne une violation des contraintes énergétiques (dans le cas où le VE a besoin de plus d'énergie pour servir ce client), $Repar(t)$ est utilisée pour réparer la solution.

Algorithm 13 First Improvement insertion

Input : Solution partielle S, L liste des clients à insérer
Output : Solution S.
T_j^d : tournée du véhicule j le jour d.
$BestTour$ le meilleur tour pour chaque client
d_{best}, j_{best} indices du meilleure jour, tournée pour chaque client
for all $(i \in L)$ **do**
    $CostMin = \infty$ ▷ Coût d'insertion du client i
    for all $d \in D(i)$ **do**
        for all $j \in \{1, .., m\}$) **do**
            if $(SimulInsert(T_j^d, i).CostMin < CostMin)$ **then**
                $CostMin := SimulInsert(T_j^d, i).CostMin$
                $BestTour := SimulInsert(T_j^d, i).Tour$
                $d_{best} := d$, $j_{best} := j$
            end if
        end for
    end for
    if $CostMin < \infty$ **then**
        $T_{j_{best}}^{d_{best}} := BestTour$
    end if
    Supprimer le client i_{best} de L
end for
Retourner S

L'Algorithme 13 résume l'opérateur en utilisant la fonction $SimulInsert(T, i)$. Cette fonction, définie précédemment, retourne la tournée $Tour$ résultante de l'insertion du client i dans la tournée T et le coût d'insertion $CostMin$. la fonction retourne $CostMin = \infty$ si l'insertion n'est pas possible. L'algorithme sélectionne le premier client non traité, teste son insertion dans toutes les tournées puis choisit la tournée où son insertion est la moins coûteuse et l'insère. Il

est possible qu'un client ne puisse être inséré dans aucune tournées de la solution, l'algorithme passera donc au client suivant et la solution obtenue ne sera pas complète.

Best Improvement insertion

Soit L la liste des clients éjectés. Cette procédure répète les trois étapes ci-dessous jusqu'à ce que $L = \emptyset$. L'étape 1 trouve l'insertion de coût minimum de chaque client $i \in L$ avec utilisation de *Repar* si nécessaire. Dans l'étape 2, le client i^* (et éventuellement la station de charge b^*) avec le coût d'insertion minimum est insérée à sa meilleure position, et finalement dans l'étape 3 L et S sont mis à jour.

Algorithm 14 Best Improvement insertion

Input : Solution partielle S, L liste des clients à insérer
Output : Solution S.
T_j^d : tournée du véhicule j le jour d.
$BestTour$ le meilleur tour pour chaque itération
i_{best}, d_{best}, j_{best} indices du meilleur client, jour et tournée pour chaque itération
$Stop := false$
while $(L \neq \emptyset \ \ \& \ \ Stop = false)$ **do**
    $CostMin := \infty$
    for all $(i \in L)$ **do**
        for all $d \in D(i)$ **do**
            for all $j \in \{1, .., m\})$ **do**
                if $(SimulInsert(T_j^d, i).CostMin < CostMin)$ **then**
                    $CostMin := SimulInsert(T_j^d, i).CostMin$
                    $BestTour := SimulInsert(T_j^d, i).Tour$
                    $i_{best} := i$, $d_{best} := d$, $j_{best} := j$
                end if
            end for
        end for
    end for
    if $CostMin < \infty$ **then**
        $T_{j_{best}}^{d_{best}} := BestTour$
        Supprimer le client i_{best} de L
    else ▷ Plus d'insertion possible
        $Stop = true$
    end if
end while
Retourner S

L'Algorithme 14 présente le fonctionnement de l'opérateur "Best Improvement insertion". A chaque itération, l'algorithme testera et calculera le coût d'insertion de tous les clients non visités dans toutes les tournées. L'insertion réalisable la moins coûteuse sera choisie et appliquée. L'algorithme s'arrête si tous les clients sont visités, et retourne donc une solution complète ou si aucune insertion n'est réalisable et retourne donc une solution non complète.

Regret insertion

Soit L la liste des clients éjectés. Cette procédure répète les deux étapes ci-dessous jusqu'à ce que $L = \emptyset$. L'étape 1 teste l'insertion de chaque client $i \in L$ dans toutes les tournées de ces jours de disponibilité, les deux insertions dans des tournées différentes avec le coût le plus bas sont retenus, et la différence δ_i entre leurs deux coûts est calculée. Si un client j n'est insérable que dans une seule tournée, le coût de la deuxième insertion est ∞, et donc $\delta_j = \infty$, ce qui permettra d'insérer ce client en priorité. Dans l'étape 2, le client i^* (et éventuellement la station de charge b^*) avec le maximum δ_{i^*} est inséré dans sa meilleure position/tournée trouvée. Si plusieurs clients ont la même valeur du δ (∞ par exemple), celui dont la meilleure insertion est la moins coûteuse est choisi.

L'Algorithme 15 présente le fonctionnement de l'opérateur "Regret insertion". A chaque itération et pour chaque client, l'algorithme testera et calculera le coût d'insertion dans toutes les tournées, les deux insertions réalisables les moins coûteuses seront retenues. Le client dont la différence entre ces deux meilleures insertions est la plus grande, sera inséré dans la tournée engendrant le minimum de coût. Si l'algorithme ne trouve aucune insertion réalisable pour tous les clients non visités, il retourne une solution non complète.

4.5 Principe de fonctionnement de notre ALNS

De même que pour l'algorithme LNS, nous utiliserons comme critère d'acceptation la règle de Metropolis (recuit simulé) et comme fonction d'évaluation des solutions la fonction de coût complet.

La Figure 4.11 détaille l'algorithme ALNS appliqué à notre problème. L'algorithme commence avec la solution S_0 d'une des trois heuristiques ($CLH, CLH2, BIH$) et initialise les différents paramètres. La roulette biaisée par les poids des opérateurs, choisit un opérateur de destruction et de construction puis les applique sur la solution. La procédure $AdjustDecreaseCharging$ est appliquée sur toutes les tournées où des clients ont été supprimés, et la procédure $repar$ est utilisée par l'opérateur de construction pour réparer des tournées où la contrainte d'énergie est violée. Une fois la nouvelle solution S_1 obtenue, les scores des opérateurs utilisés sont actualisés. Le critère d'acceptation permet de choisir la nouvelle solution courante entre celle de l'itération précédente S et celle obtenue à cette itération S_1. Les poids des opérateurs sont actualisés à la fin de chaque segment.

4.6 Expérimentations

L'objectif de cette section est de paramétrer puis d'évaluer la qualité des métaheuristiques proposées. Nous commencerons par rappeler les instances générées dans le chapitre précédant. Ensuite, nous ferons des tests pour fixer les paramètres des méthodes de résolution. Finalement, nous analyserons les résultats de chacune des métaheuristiques sur ces instances et terminerons par une étude comparative des résultats.

Nous avons proposé de nouvelles instances pour le PEVRP dans le chapitre précédent. Pour rappel, nos instances sont réparties en 9 groupes, chaque groupe contient 100 instances. Chacun des groupes est défini par sa taille d'horizon $H \in \{2, 3, 5\}$ et par le nombre de véhicules $m \in \{3, 4, 5, 6\}$. Ce qui nous donne les 9 groupes : **(2,4)**, **(2,5)**, **(2,6)**,**(3,3)**,**(3,4)**,**(3,5)**,**(5,3)**,**(5,4)** et **(5,5)**.

Algorithm 15 Regret insertion

Input : Solution partielle S, L liste des clients à insérer
Output : Solution S.
T_j^d : tournée du véhicule j le jour d.
$Tour_i$ le meilleur tour pour client i
$BestTour$ le meilleur tour pour chaque itération
d_i, j_i indices du meilleur jour et tournée pour client i
i_best , d_{best}, j_{best} indices du meilleur client, jour et tournée pour chaque itération
$Stop := false$
while $(L \neq \emptyset \ \& \ Stop = false)$ **do**
    $RegretMax := -1$
    for all $(i \in L)$ **do**
        $CostMin = \infty$; $CostMin2 = \infty$
        for all $d \in D(i)$ **do**
            for all $j \in \{1, .., m\}$) **do**
                if $(SimulInsert(T_j^d, i).CostMin < CostMin)$ **then**
                $CostMin2 = CostMin$;
                $CostMin := SimulInsert(T_j^d, i).CostMin$
                $Tour_i := SimulInsert(T_j^d, i).Tour$
                $d_i := d$, $j_i := j$
                else
                if $(SimulInsert(T_j^d, i).CostMin < CostMin2)$ **then**
                $CostMin2 := SimulInsert(T_j^d, i).CostMin$
                end if
                end if
            end for
        end for
        if $CostMin2 - CostMin > RegretMax$ **then**
            $RegretMax = CostMin2 - CostMin$
            $i_{best} = i$; $d_{best} = d_i$; $j_{best} = j_i$
            $BestTour = Tour_i$
        end if
    end for
    if $RegretMax \geq 0$ **then**
        $T_{j_{best}}^{d_{best}} := BestTour$
        Supprimer le client i_{best} de L
    else ▷ Il n'y a plus d'insertion possible
        $Stop = true$
    end if
end while
Retourner S

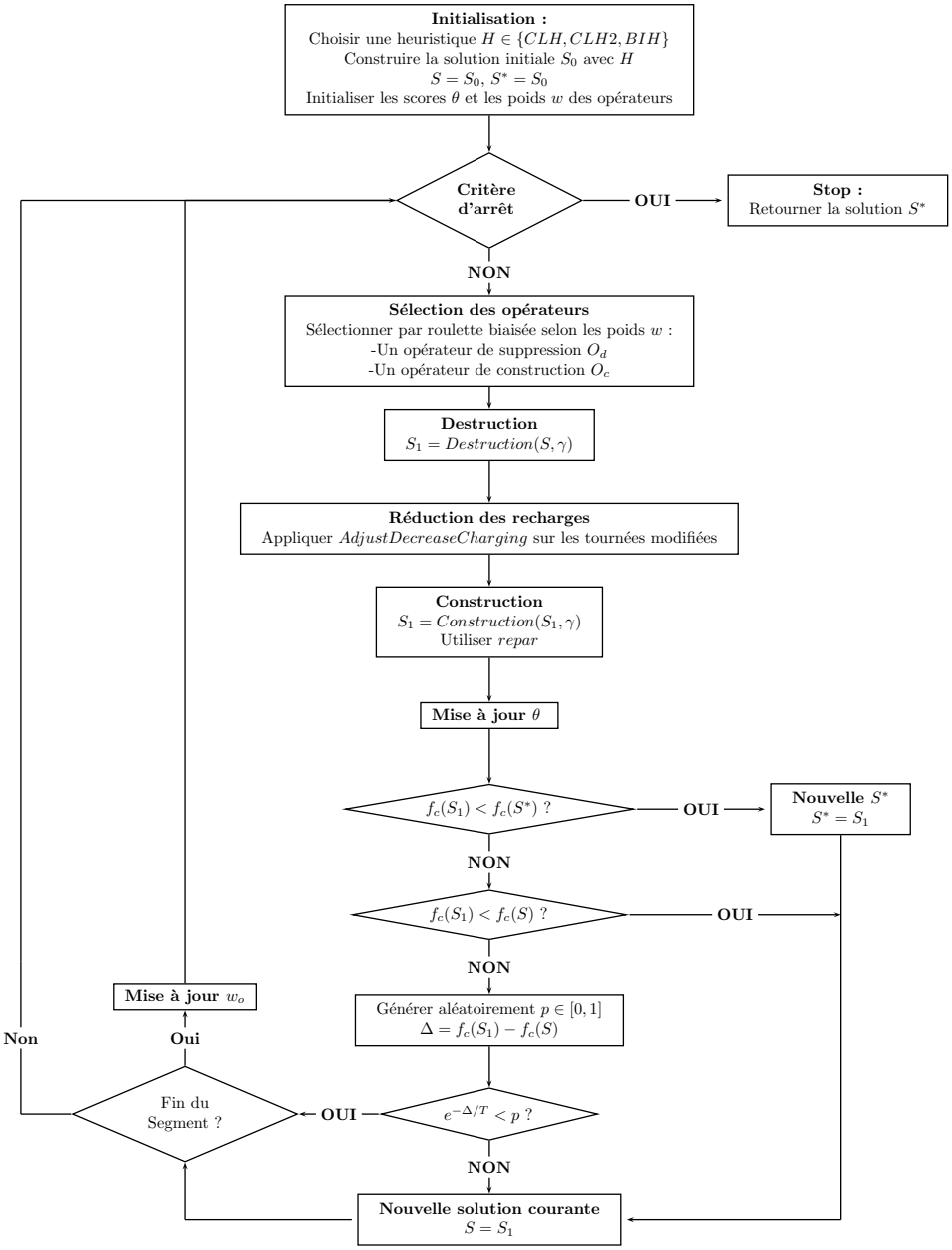


FIGURE 4.11 – Organigramme MALNS

4.6.1 Paramétrage des métaheuristiques

Pour utiliser la métaheuristique LNS, il est nécessaire de choisir une paire unique d'opérateurs de destruction et d'insertion. Pour les deux métaheuristiques LNS et ALNS, il est nécessaire de fixer le nombre p de clients à supprimer à chaque itération. Pour trouver la bonne paire d'opérateurs et p, nous avons lancé une batterie de tests de LNS sur 83 instances. Pour chaque instance, nous avons testé 216 différentes configurations LNS en variant :

— la solution initiale Sol_0 : les trois solutions obtenues par les trois heuristiques BIH,CLH et $CLH2$

Opérateur de destruction	**Opérateur d'insertion**	**Gap Sol_0**	**Gap à best Sol**	**% Best atteint**
$Cluster.removal$	$Best^{All}$	3%	48%	0,00%
$Random.removal$	$Best^{All}$	3%	48%	0,00%
$Worst.removal$	$Best^{All}$	9%	41%	0,00%
$Cluster.removal$	$Best^{2}$	21%	27%	0,00%
$Random.removal$	$Best^{2}$	42%	8%	8,33%
$Worst.removal$	$Best^{2}$	23%	25%	0,00%
$Cluster.removal$	$First^{All}$	7%	43%	0,00%
$Random.removal$	$First^{All}$	7%	43%	0,00%
$Worst.removal$	$First^{All}$	13%	35%	0,00%
$Cluster.removal$	$First^{2}$	23%	25%	0,00%
$Random.removal$	$First^{2}$	44%	6%	13,89%
$Worst.removal$	$First^{2}$	25%	24%	0,00%
$Cluster.removal$	$Regret^{All}$	4%	47%	0,00%
$Random.removal$	$Regret^{All}$	4%	47%	0,00%
$Worst.removal$	$Regret^{All}$	11%	38%	0,00%
$Cluster.removal$	$Regret^{2}$	21%	27%	0,00%
$Random.removal$	$Regret^{2}$	**46%**	**5%**	**72,22%**
$Worst.removal$	$Regret^{2}$	28%	20%	5,56%

TABLE 4.1 – Évaluation des paires d'opérateurs

— l'opérateur de destruction : $Dest.Op \in \{Cluster.removal, Random.removal, Worst.removal\}$
— l'opérateur d'insertion : $Const.Op \in \{Best^{All}, Best^{2}, First^{All}, First^{2}, Regret^{All}, Regret^{2}\}$
— $p \in \{5, 10, 15, 20\}$

Nous avons fixé comme critère d'arrêt une durée de 15 minutes pour chaque test.

Soit S_i^{config} la solution obtenue par une configuration $config = (Sol_0, Dest.Op, Const.Op, p)$ du LNS et une instance i. Et soit S_i^* la meilleure solution obtenue pour l'instance i sur toutes les configuration. Pour évaluer une configuration $config$, il est donc nécessaire d'évaluer S_i^{config}. Nous avons considéré trois métriques pour évaluer S_i^{config} :

— Gap à Sol_0 : $\frac{f_c(Sol_0)-f_c(S_i^{config})}{f_c(Sol_0)}$
— Gap à S_i^* : $\frac{f_c(S_i^{config})-f_c(S_i^*)}{f_c(S_i^*)}$
— Best atteint : $= 1$ Si $f_c(S_i^{config}) = f_c(S_i^*)$

Dans un premier temps, nous allons comparer les différents paires opérateurs (Tableau 4.1) pour sélectionner le couple d'opérateurs le plus performant. Pour évaluer et comparer les différentes paires, nous avons calculé la moyenne du Gap à Sol_0 sur toutes les instances, la moyenne du Gap à S_i^*, et le pourcentage de best atteint sur les 83 instances(996 tests). Les résultats montrent clairement que la paire ($Random.removal$, $Regret^2$) donne les meilleurs résultats.

Le Tableau 4.2 évalue la variation de p sur la performance du LNS. Il est difficile d'identifier la meilleure valeur du p selon les résultats du tableau. D'un côté, la valeur $p = 10$ donne légèrement de meilleurs résultats sur les deux premières métriques, mais $p = 20$ est la meilleure configuration pour atteindre la meilleure solution. Il est donc difficile de décider quelle valeur fixée pour p.

Puisque nous avons choisi la paire d'opérateurs à utiliser dans la LNS, il est intéressant d'évaluer la variation de p sur les configurations avec la paire ($Random.removal$, $Regret^2$) comme opérateurs (Tableau 4.3). Sur ce tableau, la valeur $p = 20$ est plus clairement performante sur les trois métriques.

p	**Gap Sol_0**	**Gap à best Sol**	**Nbr Best atteint**
5	18,17%	31,37%	19,44%
10	**18,77%**	**30,67%**	22,22%
15	18,70%	30,77%	22,22%
20	18,56%	31,04%	**36,11%**

TABLE 4.2 – Évaluation du nombre de clients à supprimer

p	**Gap Sol_0**	**Gap à best Sol**	**Nbr Best atteint**
5	44,93%	5,74%	19,23%
10	45,80%	5,15%	15,38%
15	46,12%	4,84%	26,92%
20	**47,39%**	**4,00%**	**38,46%**

TABLE 4.3 – Évaluation du nombre de clients à supprimer en fixant la paire d'opérateurs

4.6.2 Amélioration des solutions par la métaheuristique ALNS

Nous allons commencer par analyser la performance de la métaheuristique ALNS, puis nous analyserons l'impact de la solution initiale sur la performance de l'ALNS. Nous avons vu dans le chapitre précédent, que les groupes d'instances permettent de tester les heuristiques dans différents environnements, c'est pour cela que nous présenterons les résultats par groupe d'instances.

Pour chaque instance, la métaheuristique ALNS est lancée avec chacune des solutions obtenues par les trois heuristiques. Pour évaluer la performance de l'ALNS, il est nécessaire de comparer les solutions initiales (BIH, CLH,CLH2) aux solutions obtenues à partir de ces dernières (BIH+ALNS, CLH+ALNS,CLH2+ALNS).

Le Tableau 4.4 compare pour chaque heuristique de départ et pour chaque groupe d'instances, la moyenne du coût complet des solutions initiales à la moyenne du coût complet des solutions finales de la métaheuristique ALNS. Pour mieux analyser l'amélioration obtenue par la métaheuristiques ALNS, nous avons calculé l'amélioration du coût complet pour chaque solution donnée par la métaheuristique ALNS, les moyennes de ces améliorations par heuristiques/groupe d'instances sont affichées dans le Tableau 4.5.

Les résultats dans 4.5 montrent une amélioration globale de 24.5% des solutions initiales toutes instances et heuristiques confondus. La meilleure amélioration par heuristiques/groupes d'instances est de 52.1% sur les instances (3,3) avec l'heuristique BIH. Inversement, on enregistre la plus petite amélioration 7% sur les instances (5,3) avec l'heuristique CLH2, suivi par une amélioration de 7.4% sur les même instances mais avec l'heuristique CLH. La solution de l'heuristique BIH a, en moyenne, une meilleure amélioration que les deux autres heuristiques ; cela reste à relativiser par la qualité moindre de la solution de l'heuristique BIH sur certains groupes d'instances.

On voit dans le Tableau 4.4 que la meilleure moyenne est enregistrée sur les résolutions ayant comme solution initiale une des deux heuristiques de clustering (CLH, CLH2) et cela même pour les instances ((2,6),(3,5),(5,5)) où la solution initiale BIH est meilleure. Entre les deux heuristiques de clustering, l'heuristique CLH2 donnent de meilleurs résultats puisqu'elle obtient une meilleure moyenne sur 6 groupes parmi les 9 groupes d'instances. Il est donc en moyenne plus intéressant, en termes de coût complet, de lancer la métaheuristique ALNS avec une solution

Coût Complet Moyen						
Inst	**BIH**	**BIH+ALNS**	**CLH**	**CLH+ALNS**	**CLH2**	**CLH2+ALNS**
(2,4)	5364,37	4026,06	4873	3981,39	**4742,48**	**3895,33**
(2,5)	3068,92	1953,85	**2506,45**	**1947,38**	2550,1	1959,51
(2,6)	**2507,5**	1968,78	2564,5	**1933,06**	2606,36	1943,85
(3,3)	5081,78	2286,35	**3690,59**	**2198,32**	3921,59	2209,98
(3,4)	2871,42	2025,26	2875,43	2017,72	**2797,21**	**2016,18**
(3,5)	**2676,67**	2070,36	3017,86	2053,35	2878,17	**2042,07**
(5,3)	11093,85	9586,04	10347,39	9505,07	**10213,45**	**9344,46**
(5,4)	6659,39	4003,42	5038,14	3801,6	**4998,04**	**3793,75**
(5,5)	**5038,45**	3870,06	5293,91	3824,55	5084,09	**3805,22**
Moy	4929,15	3532,24	4467,48	3473,6	**4421,28**	**3445,59**

TABLE 4.4 – Amélioration moyenne du coût complet par groupes d'instances par ALNS

Inst	**BIH + ALNS**	**CLH + ALNS**	**CLH2 + ALNS**	**Moy**
(2,4)	20,40%	15,19%	15,04%	16,88%
(2,5)	31,56%	21,05%	21,37%	24,66%
(2,6)	20,15%	23,98%	24,23%	22,79%
(3,3)	52,12%	35,65%	36,09%	41,29%
(3,4)	26,78%	29,20%	26,69%	27,55%
(3,5)	21,60%	31,87%	28,71%	27,39%
(5,3)	11,48%	7,38%	7,03%	8,63%
(5,4)	36,69%	23,65%	22,48%	27,61%
(5,5)	20,85%	27,33%	24,64%	24,27%
Moy	26,85%	23,92%	22,92%	24,56%

TABLE 4.5 – Moyenne d'amélioration du coût complet par groupes d'instances et par heuristiques "ALNS"

initiale CLH2.

le Tableau 4.6 compare, pour chaque heuristique de départ et pour chaque groupe d'instances, la moyenne du nombre de clients non visités entre solutions initiales et finales de la métaheuristique ALNS. La métaheuristique ALNS réussit à baisser le nombre de clients non visités dans la majorités des groupes d'instances. Cette baisse est plus conséquente sur les solutions initiales BIH, ce qui s'explique par la difficulté constatée dans le chapitre précédent de cette heuristique à visiter un maximum de clients dans les instances les plus difficiles.

Le tableau 4.7 présente le pourcentage de solutions incomplètes par groupe d'instances, chaque groupe englobant 100 instances distinctes. Sur les instances ((2,4), (5,3)), la métaheuristique ALNS n'obtient aucune solution complète, ce qui renforce l'hypothèse avancée dans le chapitre précédent sur la difficulté de ces deux groupes d'instances. Sur le groupe d'instances (3,3), la métaheuristique ALNS a permis d'obtenir un nombre conséquent de solutions complètes en trouvant 94 nouvelles solutions complètes à partir de la solution initiale de BIH, 51 nouvelles solutions complètes pour CLH, et 54 nouvelles solutions pour CLH2. Sur le reste des groupes d'instances, la métaheuristique obtient toujours des solutions complètes.

Nombre Clients non visités moyen

Inst	BIH	BIH+ALNS	CLH	CLH+ALNS	CLH2	CLH2+ALNS
(2,4)	**10,43**	9,45	10,54	9,37	10,63	**9,27**
(2,5)	1,09	**0**	**0,03**	**0**	0,04	**0**
(2,6)	0,05	**0**	**0**	**0**	0,01	**0**
(3,3)	5,42	0,15	**1,52**	0,11	1,77	**0,06**
(3,4)	0,3	**0**	**0,01**	**0**	0,03	**0**
(3,5)	**0**	**0**	**0**	**0**	**0**	**0**
(5,3)	**29,13**	**29,13**	31,18	29,37	30,94	29,44
(5,4)	3,65	**0**	**0,07**	**0**	0,19	**0**
(5,5)	0,65	**0**	**0,01**	**0**	**0,01**	**0**
Moy	5,64	4,34	**4,82**	4,32	4,85	**4,31**

TABLE 4.6 – Nombre de clients non visités par groupes d'instances par ALNS

% solution incomplète

Inst	BIH	BIH+ALNS	CLH	CLH+ALNS	CLH2	CLH2+ALNS
(2,4)	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**
(2,5)	47%	**0 %**	**3 %**	**0 %**	4%	**0 %**
(2,6)	2%	**0 %**	**0 %**	**0 %**	1%	**0 %**
(3,3)	99%	5%	**56 %**	5%	57%	**3 %**
(3,4)	20%	**0 %**	**1 %**	**0 %**	3%	**0 %**
(3,5)	**0 %**	**0 %**	**0 %**	**0 %**	**0 %**	**0 %**
(5,3)	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**
(5,4)	98%	**0 %**	**6 %**	**0 %**	14%	**0 %**
(5,5)	45%	**0 %**	**1 %**	**0 %**	**1 %**	**0 %**
Moy	56,78%	25%	**29,67 %**	**22,89 %**	31,11%	**22,89 %**

TABLE 4.7 – Nombre de solutions incomplètes par groupes d'instances par ALNS

Coût Complet moyen						
Inst	**BIH**	**BIH+LNS**	**CLH**	**CLH+LNS**	**CLH2**	**CLH2+LNS**
(2,4)	5364,37	3690,14	4873	**3667,1**	**4742,48**	3694,86
(2,5)	3068,92	1831,56	**2506,45**	1813,24	2550,1	**1805,22**
(2,6)	**2507,5**	1840,72	2564,5	**1821,62**	2606,36	1824,63
(3,3)	5081,78	1962,93	**3690,59**	**1952,65**	3921,59	1955,24
(3,4)	2871,42	1873,79	2875,43	**1847,17**	**2797,21**	1851,61
(3,5)	**2676,67**	1909,47	3017,86	1885,36	2878,17	**1882,43**
(5,3)	11093,85	**8998,97**	10347,39	9077,51	**10213,45**	9015,93
(5,4)	6659,39	3504,31	5038,14	3372,29	**4998,04**	**3356,48**
(5,5)	**5038,45**	3574,56	5293,91	3373,57	5084,09	**3369,94**
Moy	4929,15	3242,94	4467,48	3201,17	**4421,28**	**3195,15**

TABLE 4.8 – Amélioration moyenne du coût complet par groupes d'instances par LNS

Inst	**BIH + LNS**	**CLH + LNS**	**CLH2 + LNS**	**Moy**
(2,4)	27,03%	21,73%	19,49%	22,75%
(2,5)	35,51%	26,28%	27,48%	29,76%
(2,6)	25,29%	28,29%	28,74%	27,44%
(3,3)	58,59%	41,88%	42,88%	47,78%
(3,4)	32,19%	35,10%	32,57%	33,28%
(3,5)	27,70%	37,37%	34,24%	33,10%
(5,3)	16,63%	11,42%	10,19%	12,75%
(5,4)	44,20%	31,98%	30,91%	35,70%
(5,5)	26,69%	35,72%	33,06%	31,82%
Moy	32,65%	29,98%	28,84%	30,49%

TABLE 4.9 – Moyenne d'amélioration du coût complet par groupes d'instances et par heuristiques "LNS"

4.6.3 Amélioration des solutions par la métaheuristique LNS

Après la métaheuristique ALNS, nous allons ici analyser les performances de la métaheuristique LNS et l'impact du choix de la solution initiale à la performance de la métaheuristique.

Le Tableau 4.8 présente une comparaison du coût complet moyen de la solution initiale à la solution finale obtenue par la métaheuristique LNS, les résultats sont affichés par groupes d'instances. Le Tableau 4.9 met en évidence l'amélioration moyenne par heuristique et par groupes d'instances.

La métaheuristique LNS permet une amélioration moyenne de 30% toutes heuristiques et instances confondus. La meilleure amélioration 58.6% s'enregistre sur les instances (3,3) avec l'heuristique BIH, et la plus basse amélioration 10.19% est de $10,2\%$ sur les instances (5,3) avec l'heuristique CLH2. En moyenne, la métaheuristique LNS permet une plus grande amélioration (32.7%) sur les solutions initiales BIH. Malgré cette plus grande amélioration, l'heuristique BIH n'obtient une meilleure moyenne de coût complet (Tableau 4.8) que sur un groupe d'instances (5,3). Sur le reste des groupes d'instances, les deux heuristiques de clustering (CLH et CLH2) se partagent les moyennes de coût les plus basses.

Le Tableau 4.10 présente la moyenne du nombre de clients non visités pour les solutions

Nombre Clients non visités moyen

Inst	BIH	BIH+LNS	CLH	CLH+LNS	CLH2	CLH2+LNS
(2,4)	**10,43**	9,12	10,54	**8,96**	10,63	9,24
(2,5)	1,09	**0**	**0,03**	**0**	0,04	**0**
(2,6)	0,05	**0**	**0**	**0**	0,01	**0**
(3,3)	5,42	**0**	**1,52**	**0**	1,77	**0**
(3,4)	0,3	**0**	**0,01**	**0**	0,03	**0**
(3,5)	**0**	**0**	**0**	**0**	**0**	**0**
(5,3)	**29,13**	**29,28**	31,18	29,57	30,94	29,3
(5,4)	3,65	**0**	**0,07**	**0**	0,19	**0**
(5,5)	0,65	**0**	**0,01**	**0**	**0,01**	**0**
Moy	5,64	4,3	**4,82**	**4,28**	4,85	4,29

TABLE 4.10 – Amélioration moyenne du nombre de clients non visités par groupes d'instances par LNS

initiales et finales de la métaheuristique LNS, les résultats sont regroupés par heuristique et par groupe d'instances. Le Tableau 4.11 présente le pourcentage de solutions incomplètes par groupe d'instances et par heuristique, chaque pourcentage est calculé sur une base de 100 instances.

La métaheuristique LNS réussit à baisser le nombre de clients non visités dans la majorité des groupes d'instances, à l'exception du groupe (5,3) où la métaheuristique donne des solutions ayant en moyenne plus de clients non visités. Ce comportement s'explique premièrement par la grande difficulté, si ce n'est impossibilité, de trouver des solutions complètes sur ce groupe d'instances. Aussi, la métaheuristique a comme critère de minimiser la fonction de coût complet, fonction qui pénalise une solution incomplète selon le nombre de véhicules supplémentaires nécessaires pour visiter les clients non visités. Il est donc possible qu'une solution ayant plus de clients non visités soit "meilleure" en termes de coût complet qu'une solution avec moins de clients non visités si la première nécessite moins de véhicules supplémentaires que la deuxième pour visiter tous les clients.

De même que la métaheuristique ALNS, sur les instances (2,4) et (5,3), la métaheuristique LNS n'obtient aucune solution complète, ce qui renforce encore une fois l'hypothèse avancée sur la difficulté de ces deux groupes instances. Sur tous les autres groupes d'instances, la métaheuristique LNS obtient des solutions complètes. La meilleure performance étant de passer de 2% de solutions complètes obtenus par l'heuristique BIH sur les instances (5,4) a 100% de solutions complètes.

4.6.4 Étude comparative des métaheuristiques

Dans cette section nous comparerons les deux métaheuristiques sans prendre en considération la solution initiale. Le premier Tableau 4.12 reprend le coût complet moyen par groupe d'instances. Sachant qu'une métaheuristique est lancée trois fois par instance, une fois pour chaque solution initiale différente, et qu'un groupe d'instances regroupe 100 instances. On remarque sur ce premier tableau que la métaheuristique LNS a une meilleure moyenne sur tous les groupes d'instances avec un écart de 8.4% en moyenne, écart qui descend à 5% pour les instances (5,3) et qui atteint 14% sur les instances (3,3). Ce premier tableau montre donc un avantage de la métaheuristique LNS.

Le deuxième Tableau 4.13 compare les deux métaheuristiques par rapport au nombre moyen

	% solution incomplète					
Inst	**BIH**	**BIH+LNS**	**CLH**	**CLH+LNS**	**CLH2**	**CLH2+LNS**
(2,4)	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**
(2,5)	47%	**0 %**	**3 %**	**0 %**	4%	**0 %**
(2,6)	2%	**0 %**	**0 %**	**0 %**	1%	**0 %**
(3,3)	99%	**0 %**	**56 %**	**0 %**	57%	**0 %**
(3,4)	20%	**0 %**	**1 %**	**0 %**	3%	**0 %**
(3,5)	**0 %**	**0 %**	**0 %**	**0 %**	**0 %**	**0 %**
(5,3)	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**	**100 %**
(5,4)	98%	**0 %**	**6 %**	**0 %**	14%	**0 %**
(5,5)	45%	**0 %**	**1 %**	**0 %**	**1 %**	**0 %**
Moy	56,78%	24,44%	**29,67 %**	**22,33 %**	31,11%	22,56%

TABLE 4.11 – Amélioration du nombre de solutions incomplètes par groupes d'instances par LNS

Inst	**ALNS**	**LNS**
(2,4)	3 967,59	3 684,04
(2,5)	1 953,58	1 816,68
(2,6)	1 948,57	1 828,99
(3,3)	2 231,55	1 956,94
(3,4)	2 019,72	1 857,53
(3,5)	2 055,26	1 892,42
(5,3)	9 478,52	9 030,80
(5,4)	3 866,26	3 411,03
(5,5)	3 833,28	3 439,36
Moy	3 483,81	3 213,09

TABLE 4.12 – Comparaison du coût complet moyen par métaheuristiques sur les groupes d'instances

Inst	ALNS	LNS
(2,4)	9,36	9,11
(2,5)	0,0	0,0
(2,6)	0,0	0,0
(3,3)	0,11	0,00
(3,4)	0,00	0,00
(3,5)	0,00	0,00
(5,3)	29,31	29,38
(5,4)	0,00	0,00
(5,5)	0,00	0,00

TABLE 4.13 – Comparaison du nombre de clients non visités moyen par métaheuristiques sur les groupes d'instances

de clients non visités. Sur les instances (5,3), la métaheuristique ALNS enregistre moins de clients non visités que la métaheuristique LNS. Inversement, la métaheuristique LNS enregistre moins de clients non visités sur les instances (2,4) et (3,3). Comme expliqué précédemment, les instances (2,4) et (5,3) sont des instances où il est impossible de visiter tous les clients et donc d'avoir des solutions complètes. Entre deux solutions incomplètes, la fonction de coût complet n'avantage pas systématiquement la solution ayant moins de clients non visités. Il est donc difficile de juger l'efficacité des métaheuristiques sur ces instances "difficiles" avec la métrique du nombre de clients non visités. La même logique pourrait s'appliquer sur les instances (3,3) où la métaheuristique ALNS a produit 13 solutions incomplètes (voir Tableau 4.7) sur 10 instances différentes, mais le fait que la métaheuristique LNS n'ait aucune solution incomplète sur ce groupe d'instances permet de rejeter cette logique et de consolider l'avantage de la métaheuristique LNS sur la métaheuristique ALNS.

Pour une instance, nous avons possiblement 9 solutions différentes, 3 solutions obtenues par les heuristiques, et 3 solutions obtenues par chacune des métaheuristiques. les métaheuristiques retournant la solution initiale si aucune amélioration n'est trouvée, la meilleure solution trouvée d'une instance fait partie des 6 solutions obtenues par les deux métaheuristiques. Pour évaluer et comparer les deux métaheuristiques, nous avons compté le nombre d'instances où chacune des métaheuristiques a trouvé la meilleure solution. Le Tableau 4.14 donne donc le nombre d'instances où une métaheuristique a trouvé une solution meilleure que les trois solutions trouvées par l'autre métaheuristique. Par exemple sur les instances (2,4), la métaheuristique ALNS a trouvé la meilleure solution pour 50 instances parmi 100 et la métaheuristique LNS a trouvé la meilleure solution pour 52 instances ; la somme n'étant pas égale à 100 puisque les deux métaheuristiques ont trouvé exactement la même solution sur deux instances. Le tableau montrent un équilibre entre les deux métaheuristiques sur les instances (2,4) mais sur le reste des instances, la métaheuristique LNS affiche de meilleurs résultats. Globalement, la métaheuristique LNS obtient la meilleure solution dans 87% des cas.

La métaheuristique LNS est plus performante pour atteindre la meilleure solution, même si elle n'atteint pas la meilleure solution sur toutes les instances. Nous allons donc évalué les deux métaheuristiques dans la situation où elles ne trouvent pas la meilleure solution. Soit $S_{ALNS} = \{S^1_{ALNS}, S^2_{ALNS}, S^3_{ALNS}\}$ et $S_{LNS} = \{S^1_{LNS}, S^2_{LNS}, S^3_{LNS}\}$ les solutions trouvées par les deux métaheuristiques et $S*_{ALNS}$ et $S*_{LNS}$ la meilleure solution pour chacune des métaheuristiques. Sur toutes les instances où l'ALNS trouve la meilleure solution($f_c(S*_{ALNS}) < f_c(S*_{LNS})$), nous calculons le gap $gap_{LNS} = \frac{f_c(S*_{LNS}) - f_c(S*_{ALNS})}{f_c(S*_{ALNS})}$, puis nous faisons la moyenne. Sur les instances où

Inst	ALNS	LNS
(2,4)	50,00	52,00
(2,5)	6,00	94,00
(2,6)	6,00	94,00
(3,3)	20,00	80,00
(3,4)	1,00	99,00
(3,5)	1,00	99,00
(5,3)	36,00	72,00
(5,4)	1,00	99,00
(5,5)	0,00	100,00
Moy	13,44	87,67

TABLE 4.14 – Cas où la métaheuristique a trouvé la meilleure solution

Inst	ALNS	LNS
(2,4)	8,4%	5,5%
(2,5)	4,9%	1,3%
(2,6)	4,3%	1,1%
(3,3)	7,2%	3,1%
(3,4)	6,2%	0,1%
(3,5)	6,1%	0,3%
(5,3)	5,2%	3,3%
(5,4)	10,2%	0,1%
(5,5)	10,2%	/
Moy	7%	2%

TABLE 4.15 – Gap à la meilleure solution

la LNS trouve la meilleure solution, nous faisons la moyenne sur $gap_{ALNS} = \frac{f_c(S*_{ALNS}) - f_c(S*_{LNS})}{f_c(S*_{LNS})}$. La moyenne par groupe d'instances est donnée dans le Tableau 4.15.

Les résultats dans le Tableau 4.15 montrent que sur les instances où la métaheuristique ALNS trouve une meilleure solution, la métaheuristique LNS reste à moins de 2% en moyenne. Inversement, l'ALNS est à 7% en moyenne de la meilleure solution obtenue par la LNS.

4.7 Conclusion

Dans ce chapitre, nous avons développé deux métaheuristiques pour la résolution du PEVRP. La première métaheuristique et une métaheuristique de recherche à large voisinage (LNS) et la deuxième de recherche adaptative à large voisinage (ALNS). Nous avons aussi présenté la fonction $AdjustDecreaseCharging$, fonction qui permet de réduire les recharges non nécessaires dans une tournée.

Les premières expérimentations nous ont permis de trouver les meilleurs paramètres pour les deux métaheuristiques. Puis, nous avons analyser l'amélioration obtenue de la solution initiale obtenue par les deux méthodes de résolution. Finalement, nous avons comparé les résultats des deux métaheuristiques sur une batterie d'instances.

Nous avons pu conclure que la LNS est plus performante sur l'ensemble des instances. La méthode LNS réussit à trouver la meilleure solution pour 87% des instances. Sur les instances où

la métaheuristique LNS ne trouve pas la meilleure solution, elle obtient en moyenne une solution à 2% de la meilleure obtenue. Une forte intensification est donc plus adaptée et utile pour le PEVRP.

Chapitre 5

Problème de planification des tournées de véhicules électriques avec multi-visites sur l'horizon

Sommaire

5.1 Introduction

Dans ce chapitre nous étudions une extension du problème traité dans le chapitre précédent. Cette fois-ci, la fréquence de visites des clients est multiples, nous généralisons alors les approches de résolutions proposées dans le quatrième chapitre.

5.2 Définition de la problématique et modélisation

Le m-GPEVRP est défini sur le graphe orienté complet $G = (V, A)$. $V = C \cup B \cup \cup\{0\}$, l'ensemble C de n sommets représentent les clients, l'ensemble B de ns stations externes, qui peuvent être visitées chaque jour de l'horizon et le sommet 0 le dépôt qui comporte les bornes permettant une recharge pendant le jour et pendant le soir. A est l'ensemble des arcs, où chaque arc (i, j) a un coût de déplacement c_{ij}, une distance d_{ij} et un temps de parcours t_{ij}. Lorsqu'un arc (i, j) est parcouru par un véhicule électrique (EV), il consomme une quantité d'énergie $e_{i,j} = taux \times d_{i,j}$, où $taux$ indique un taux de consommation d'énergie constant.

Nous considérons un horizon de temps H de np périodes typiquement des "jours", dans lequel chaque client i a une fréquence $fr(i)$ de visites. Cela signifie que le client i doit être servi $fr(i)$ fois sur l'horizon temporel, où $1 \leq fr(i) \leq np$, mais chaque client ne peut être visité qu'au plus une fois par jour. Alors le nombre total de visites nv à effectuer sur H est $nv = \sum_{\forall i \in C} fr(i)$. Chaque client i a un ensemble $comb(i)$ de scénarios de visites ; chaque scénario a $fr(i)$ jours de visites, par exemple, $(1, 3)$ signifie que le client sera visité le premier et le troisième jour ("lundi" et "mercredi") si ce scénario est choisi. À chaque visite, un client i a besoin d'une quantité q_i de marchandises. Un coût de recharge fixe de Cc est considéré, qui ne dépend ni de la quantité d'énergie fournie par les stations de recharge ni du temps nécessaire pour charger le véhicule. La puissance d'énergie délivrée à chaque véhicule k au cours d'une journée h est une variable de décision $P_{h,0,k}$ qui fixe l'état initial de charge du véhicule k avant de commencer les trajets du jour $h + 1$, $h \in \{1...np\}$.

Le m-GPEVRP consiste à assigner à chaque client un scénario dans $comb(i)$ qui minimise le coût total des tournées et de la recharge sur H. Une solution réalisable de m-GPEVRP doit satisfaire l'ensemble des contraintes suivantes : (i) chaque tournée doit débuter et se terminer au dépôt, (ii) chaque client i doit être visité $fr(i)$ fois pendant l'horizon de planification selon un scénario de $comb(i)$, (iii) la demande du client q_i doit être entièrement satisfaite pendant chaque visite, (iv) pas plus de m véhicules électriques sont utilisés, (v) la durée totale de la tournée, calculée comme la somme du temps requis pour atteindre les clients, du temps requis pour charger le véhicule pendant la journée ; (vi) la quantité totale de marchandises livrée le long de la tournée, donnée par la somme des demandes q_i des clients visités, ne doit pas dépasser la capacité du véhicule Q.

La fonction objectif à minimiser est $f(S) = \sum_{tr \in S} TourCost(tr)$, où les tr sont les tournées composant la solution S et $TourCost$ la fonction calculant le coût d'une tournée. $TourCost(tr) = \alpha \times TourDist(tr) + Cc \times nbs(tr)$ où $TourDist(tr)$ est la distance totale de la tournée tr, et $nbs(tr)$ est le nombre total de visites aux stations de recharge dans la tournée tr. α est un poids donné représentant le coût d'une unité de distance.

5.2.1 Formulation mathématique

Pour le modèle mathématique nous allons partir du même graphe G décrit plus haut à l'exception du dépôt qui sera dupliqué en deux sommets, un dépôt de départ $\bar{0}$ et un d'arrivée $\underline{0}$. Cette modification est nécessaire pour calculer l'heure de fin d'une tournée et vérifier qu'elle ne dépasse pas T_{max}, $0 = \{\bar{0}, \underline{0}\}$ devient donc l'ensemble des deux, et on posera aussi $V_{/0} = C \cup B$ l'ensemble des sommets à l'exception des dépôts. Aussi, l'ensemble B ne représentera plus l'ensemble des bornes mais un ensemble contenant plusieurs copies de chaque borne. Cela permet de garder un schéma ou tous les sommets, à l'exception du dépôt, ne peuvent être visité qu'une fois tout en permettant de visiter une borne plusieurs fois dans la solution. Cette unicité permet d'un coté d'exprimer les contraintes d'élimination des sous-tours et de l'autre de ne pas

Paramètres	Définitions
$\bar{0}$ ($\underline{0}$)	Copie du dépôt d'où partent (arrivent) les véhicules
O	$= \{\bar{0}, \underline{0}\}$
C	Ensemble des clients
B	Ensemble des copies des bornes
V	Ensemble de tous les sommets $= B \cup C \cup O$
$V_{/O}$	Ensemble des sommets à l'exception du dépôt $= B \cup C$
V_e	Ensemble des m véhicules
H	L'ensemble des jours de l'horizon
t_{ij}	Temps de parcours de l'arc (i, j)
e_{ij}	Énergie nécessaire pour parcourir
c_{ij}	Coût de parcours de l'arc (i, j)
Cc	Coût forfaitaire d'une visite à une borne
T_{max}	Durée maximale d'une tournée
s_i	Temps de service du client i
q_i	Demande du client i
$comb(i)$	Ensemble de scénarios du client i
$taux$	Taux de recharge
Q	Capacité de charge des marchandises d'un véhicule
E	Capacité maximale de la batterie d'un VE

Table 5.1 – Notations

confondre deux tournées qui visitent la même borne. Ci-dessous nous allons recenser les notations et variables nécessaires pour le modèle puis nous présenterons le modèle.

Le paramètre sc^d_{il} permet de connaître les jours faisant partie des différents scénarios :

$$sc^d_{il} = \begin{cases} 1 \text{ Si le jour } d \text{ fait partie du scénario } l \text{ du client } i \\ 0 \text{ Sinon} \end{cases}$$

Variables La variable principale permet de décider si un arc est emprunté par un véhicule ou non et cela pour chaque véhicule et chaque jour de l'horizon :

$$x^d_{ijk} = \begin{cases} 1 \text{ Si l'arc}(i, j) \text{ est emprunté par le véhicule } k \text{ le jour } d \\ 0 \text{ Sinon} \end{cases}$$

Une deuxième variable permet de choisir le scénario pour chaque client :

$$w_{il} = \begin{cases} 1 \text{ Si le scénario } l \text{ est choisi pour le client } i \\ 0 \text{ Sinon} \end{cases}$$

D'autre variables intermédiaires sont nécessaires pour vérifier les contraintes de temps et d'énergie :

- y_i Quantité d'énergie encore disponible dans la batterie du véhicule en arrivant au sommet (client, borne ou dépôt) i
- z_i Quantité d'énergie rechargée à la borne i, cette variable est exclusive aux bornes
- π_i Heure d'arrivée du véhicule visitant le sommet i. Sachant qu'un client n'est visité qu'une fois par jour, il n'est pas nécessaire de connaître le véhicule faisant cette visite.

Modèle mathématique

$$\min \quad \sum_{d\in H}\sum_{k\in V_e}\sum_{\{i,j\}\in A} c_{ij}x^d_{ijk} + Cc\sum_{d\in H}\sum_{k\in V_e}\sum_{i\in B}\sum_{j\in V} x^d_{ijk} \tag{5.1a}$$

$$\text{s.t.} \quad \sum_{l\in comb(i)} w_{il} = 1 \qquad \forall i \in C \tag{5.1b}$$

$$\sum_{k\in V_e}\sum_{i\in V}^{i\neq j} x^d_{ijk} = w_{jl} \times sc^d_{jl} \qquad \forall d \in D, \forall j \in C \tag{5.1c}$$

$$\sum_{d\in H}\sum_{k\in V_e}\sum_{i\in V}^{i\neq j} x^d_{ijk} \leq 1 \qquad \forall j \in B \tag{5.1d}$$

$$\sum_{i\in V} x^d_{ipk} = \sum_{j\in V} x^d_{pjk} \qquad \forall d \in H, \forall p \in V_0, \forall k \in V_e \tag{5.1e}$$

$$\sum_{j\in V}^{j\neq \bar{0}} x^d_{\bar{0}jk} = 1 \qquad \forall d \in H, \forall k \in V_e \tag{5.1f}$$

$$\sum_{i\in V}^{j\neq \underline{0}} x_{i\underline{0}k} = 1 \qquad \forall d \in D, \forall k \in V_e \tag{5.1g}$$

$$\pi_j - \pi_i + T_{max}(1 - x^d_{ijk}) \geq (t_{ij} + s_i)x^d_{ijk} \qquad \forall i \in C, j \in V, d \in H, \forall k \in V_e \tag{5.1h}$$

$$\pi_j - \pi_i + (M)(1 - x^d_{ijk}) \geq t_{ij}x^d_{ijk} + z_i * taux \qquad \forall i \in B, j \in V, \forall k \in V_e \tag{5.1i}$$

$$t_{0j} \leq \pi_j \leq T_{max} - (t_{j0} + s_j) \qquad \forall j \in C \tag{5.1j}$$

$$t_{0j} \leq \pi_j \leq T_{max} - (t_{j0} + taux * z_j) \qquad \forall j \in B \tag{5.1k}$$

$$\pi_{\bar{0}} = 0, \quad \pi_{\underline{0}} \leq T_{max} \tag{5.1l}$$

$$y_j - y_i + e_{ij}x^d_{ij} \leq E(1 - x^d_{ij}) \qquad \forall i \in C, j \in V, d \in D \tag{5.1m}$$

$$y_j - y_i + e_{ij}x^d_{ij} \leq E(1 - x^d_{ij}) + z_i \qquad \forall i \in B, j \in V, d \in D \tag{5.1n}$$

$$y_{\bar{0}} = E, \quad y_{\underline{0}} \geq 0 \tag{5.1o}$$

$$0 \leq z_j \leq E - y_j \qquad \forall j \in B \tag{5.1p}$$

$$\sum_{i\in C} q_i \times \sum_{j\in C} x^d_{ijk} \leq Q \qquad \forall k \in V \tag{5.1q}$$

La fonction objectif (3.1a) est la somme des coûts des arcs empruntés dans la solution et des coûts des stations visités. Les contraintes (5.1b) permettent de vérifier qu'un seul scénario est choisi pour chaque client. Les contraintes (5.1c),(5.1d) traduisent le fait qu'un client doit être visité une fois par jour si ce jour fait partie du scénario choisi, et que la copie d'une borne ne peut être visité plus qu'une fois. Les contraintes de conservation de flot (5.1e) imposent que le même véhicule qui a visité un sommet (client ou borne) un jour donné, "reparte" de chez ce client le même jour. Les contraintes (5.1f),(5.1g) assurent que chaque véhicule fait une tournée par jour. Les contraintes (5.1h) (5.1i) servent à calculer l'heure d'arrivée chez chaque client (borne). Pour la contrainte (5.1i), nous avons utilisé M, qui est une borne supérieure de $(\pi_i + z_i * taux)$, et qui peut être exprimé par $T_{max} + taux * E$. Les contraintes (5.1j) et (5.1k) vérifient que les tournées ne dépassent pas la durée maximale. Pour calculer l'énergie des véhicules une fois arrivé chez un client ou une borne, nous avons établi les contraintes (5.1m) et (5.1n). La contrainte (5.1p) permet

Clients	Fréquence	Scénarios
1	1	{1}, {3}
2	1	{2}, {3}
3	1	{1}, {2}
4	2	{1,2}, {1,3}
5	2	{1,3}, {2,3}
6	2	{1,2}, {2,3}
7	2	{1,3}, {2,3}
8	3	{1,2,3}
9	3	{1,2,3}
10	3	{1,2,3}

TABLE 5.2 – Fréquences et scénarios des clients de l'instance *Inst*

0	1	3	4	11	63%	8	0
0	5	7	9	10	0		
0	2	4	6	0			
0	9	10	8	11	25%	0	
0	5	6	12	86%	7	9	0
0	8	10	0				

TABLE 5.3 – Représentation d'une solution

de vérifier que la recharge ne dépasse pas la capacité de la batterie d'un véhicule. Les contraintes (5.1l) et (5.1o) initialisent et bornent les variables de temps et de recharge pour les dépôts (initial et final). La contrainte (5.1q) vérifie que les marchandises nécessaires pour satisfaire l'ensemble des demandes des clients d'une tournée ne dépasse pas la capacité du véhicule.

5.3 Représentation et évaluation d'une solution

5.3.1 Représentation d'une solution

Nous avons choisi de garder la même représentation que celle présentée pour la problématique PEVRP puisqu'elle suffit à décrire la solution. En revanche, pour vérifier la réalisabilité de la solution, il est nécessaire d'avoir les données de l'instance, données incluant la fréquence et les différents scénarios de chaque client.

Soit une instance *Inst* avec trois jours, deux véhicules, dix clients et deux bornes. les indices des clients sont donc dans $\{1, .., 10\}$ et ceux des bornes dans $\{11, 12\}$. les fréquences et scénarios des clients sont explicité dans le tableau 5.2

Soit *Sol* une solution pour cette instance. Cette solution sera représentée par la matrice de la Table 5.3.

Les deux premières lignes de la matrice 5.3 correspondent aux deux tournées du premier jour, les deux suivantes aux tournées du deuxième, et les deux dernières aux tournées du troisième jour. Lors de la première tournée du premier jour, le véhicule visite les clients 1, 3 et 4 puis visite la station 11 où le véhicule recharge 63% de sa batterie, puis visite le client 8 et revient au dépôt.

Le client 1 et 3 sont donc visités selon leurs premier scénario ({1}). Les deux scénarios du client 4 contenant le premier jour, il est donc nécessaire de le visiter ce jour là. Le client 8 doit être visité pendant tous les jours de l'horizon.

5.3.2 Évaluation d'une solution

Le coût (simple) défini dans la fonction objectif ne permet pas d'évaluer une solution incomplète puisqu'il ne considère pas les clients non visités. Pour la problématique PEVRP de la Section 3.3.2, nous avons présenté une fonction de coût complet f_c qui permet d'évaluer les solutions complètes et incomplètes en rajoutant des coûts supplémentaires selon les clients non visités.

5.4 Heuristiques de construction

Pour résoudre notre problématique nous avons proposé des heuristiques à deux phases, nous choisissons un scénario pour chaque client, et puis nous construisons les tournées. Ce schéma de résolution est inspiré par les heuristiques CLH et renforcé par leurs efficacités sur la problématique PEVRP.

Pour la première phase nous avons établi une procédure, *choix.scenario* qui fixe les scénarios des clients en utilisant la logique de clusters.

5.4.1 Procédure de choix des scénarios

Soit le cluster Cl_h, ensemble des clients à visiter le jour h. Avant de fixer les scénarios des clients, ces clusters sont vides. Un client est affecté à un cluster Cl_h si un de ses scénarios de visite a été choisi et que ce scénario choisi contient le jour h.

Nous avons vu dans les chapitres précédents que la capacité des véhicules en termes de marchandise représente une contrainte dure pour la construction d'une solution. Nous poserons donc comme contrainte que la somme des demandes des clients d'un cluster, ne dépasse pas la capacité totale disponible de ce jour ($m \times Q$).

La procédure commence par trier les clients dans une liste L par ordre lexicographique décroissant sur leurs fréquences et demandes. Le premier élément de L est donc le client ayant la plus haute fréquence (généralement $|H|$) et ayant la plus grande demande parmi tous les clients de la même fréquence. Le dernier client aura la fréquence minimale et la plus petite demande parmi tous les clients de la même fréquence. Ce tri permet de gérer les clients les plus difficiles en premier.

Une fois la liste L obtenue, la procédure choisit itérativement un scénario pour minimiser la somme des distances dans tous les clusters concernés tout en vérifiant la contrainte de capacité.

5.4.2 Heuristique d'insertion par liste

L'heuristique commence par lancer la procédure de choix des scénarios (Section 5.4.1). Une fois les scénarios fixés, l'heuristique d'insertion ($Hconst1$) reprend la liste L définie précédemment et insère itérativement les clients dans la solution. L'heuristique teste donc l'insertion d'un client dans les tournées des jours composant son scénario, puis l'insère dans la meilleure tournée de chaque jour.

Algorithm 16 *choix.scenario*

Input : C l'ensemble des clients
Output : Sc vecteur donnant l'indice du scénario choisi pour chaque client
$Sc = \{-1 | i \in C\}$
Initialiser Cl
$L := C$
Trier L ordre lexicographique décroissant (fréquence, demande)
for all client $i \in L$ **do**
    $Min := \infty$
    for all $sc \in comb(i)$ **do**
        Coût d'insertion $\Delta := 0$,
        for all $h \in sc$ **do**
            if $q_i + \sum_{j \in C_h} q_{j,h} \leq m \times Q$ **then**
                $\Delta := \Delta + \sum_{j \in C_h} d_{i,j}$
            else
                $\Delta := \infty$
            end if
        end for
        if $\Delta < Min$ **then**
            $Min = \Delta$
            $Sc(i) = sc$
        end if
    end for
    Choisir le scénario $sc*$ pour le client i
end for

Algorithm 17 *Hconst*1

Input : C l'ensemble des clients
Output : Solution S
T_j^d : tournée du véhicule j le jour d (vide initialement).
Initialiser $S = \cup T_j^d$
$Sc = choix.scenario(C)$
$L := C$
Trier L par ordre lexicographique décroissant (fréquence, demande)
for all $i \in L$ **do**
    for all $h \in comb(i, Sc(i))$ **do**
        $BestTour$ le meilleur tour, j_{best} indice du meilleur véhicule
        for all $j \in \{1, .., m\}$) **do**
            if $(SimulInsert(T_j^d, i).CostMin < CostMin)$ **then**
                $CostMin := SimulInsert(T_j^d, i).CostMin$
                $BestTour := SimulInsert(T_j^d, i).Tour$
                $j_{best} := j$
            end if
        end for
        $T_{j_{best}}^h = BestTour$
    end for
end for

5.4.3 Heuristiques de meilleure insertion

Une fois les scénarios des clients fixés par la procédure de choix des scénarios (Section 5.4.1), nous proposons d'adapter l'heuristique BIH sur notre problématique. Nous devons donc calculer et comparer les coûts d'insertion des clients et insérer le client à moindre coût à chaque itération. Pour cela nous aurons besoin d'une fonction permettant d'évaluer l'insertion d'un client dans les différents jours de son scénario. Cette fonction devra aussi permettre de comparer l'insertion de deux clients, deux clients pouvant avoir des fréquences différentes.

Évaluation de l'insertion d'un client dans une solution

Dans les chapitres précédents, nous avons introduit la fonction $SimulInsert(i, Tour)$ qui permet d'évaluer le coût d'insertion d'un client i dans une tournée $Tour$. Cette fonction suffisait à évaluer le coût d'insertion global d'un client dans une solution en testant son insertion dans toutes les tournées de la solution. Puisque les fréquences étaient égales à 1, il suffisait de choisir l'insertion à moindre coût entre toutes les insertions réalisables. Dans cette nouvelle problématique, un client i nécessite $fr(i)$ visites sur l'horizon, son coût d'insertion doit donc englober ses différents coûts d'insertion dans tous les jours du scénario choisie. Nous avons donc besoin d'une fonction qui évalue l'insertion d'un client dans une solution à partir de $fr(i)$ coûts d'insertion dans des tournées différentes. Cette fonction doit permettre de comparer le coût d'insertion de deux clients n'ayant pas la même fréquence. Soit un client i, sc un de ses scénarios, et $CostDay(i, h)$ la fonction donnant le meilleur coût d'insertion du client i dans la journée h donnée par la formule (5.2).

$$CostDay(i, h) = \min_{Tour \in h} SimulInsert(i, Tour) \tag{5.2}$$

Nous avons proposé trois métriques différentes pour évaluer le coût d'insertion d'un client dans une solution.

$$Metrique1(i, cb) = \sum_{h \in cb} \frac{CostDay(i, h)}{fr(i)} \tag{5.3}$$

$$Metrique2(i, cb) = \sum_{h \in cb} \frac{CostDay(i, h)}{fr(i)^2} \tag{5.4}$$

$$Metrique3(i, cb) = \sum_{h \in cb} \frac{CostDay(i, h)}{fr(i) * \sqrt{fr(i)}} \tag{5.5}$$

La première fonction (5.3) est intuitive, puisqu'elle divise la somme des différents coûts par la fréquence d'un client. Nous savons qu'un client ayant une plus grande fréquence est plus difficile à insérer et donc il est plus efficace de l'insérer en avance. Pour cela, nous avons proposé deux autre métriques (5.4) et (5.5) qui avantagent les clients ayant une plus grande fréquence.

Algorithme des heuristiques de meilleure insertion

Le principe de l'heuristique est de tester toutes les insertions des clients, évaluer le coût d'insertion de chaque client dans la solution et choisir à chaque itération le client à moindre coût à insérer. Dans l'algorithme qui suit, nous expliciterons l'algorithme avec la fonction $Metrique1$, qui nous donne l'heuristique $Hconst2$. L'heuristique $Hconst3$ et $Hconst4$ ont le même fonctionnement mais avec les métriques $Metrique2$ et $Metrique3$ respectivement.

Algorithm 18 Heuristique de meilleure insertion $Hconst2$

Input : C l'ensemble des clients
Output : Solution S
T_j^d : tournée du véhicule j le jour d (vide initialement).
Initialiser S
$Sc = choix.scenario(C)$
$C' := C$ Les clients à visiter,
$Stop := false$,
i_{best} indice du client à insérer à chaque itération
while $C' \neq \emptyset$ & $Stop = false$ **do**
    $CostMin := \infty$
    for all $(i \in C')$ **do**
        for all $d \in Sc(i)$ **do**
            Calculer $CostDay(i, d)$
        end for
        Calculer $Metrique1(i, Sc(i))$
        if $Metrique1(i, Sc(i)) < CostMin$ **then**
            $i_{best} = i$
        end if
    end for
    if $CostMin < \infty$ **then**
        Insérer le client i_{best} dans les jour $Sc(i_{best})$
        Supprimer le client i_{best} de C'
    else ▷ Il n'y a plus d'insertion possible
        $Stop = true$
    end if
end while
Retourner S

L'Algorithme 18 initialise une solution S vide. L'ensemble C' contient les clients non insérés. Une fois un client inséré, il est supprimé de C', puis les tests d'insertion des clients restants dans C' sont relancés. Dans l'Algorithme 18 nous avons relancé toutes les insertions par simplification, mais il est possible et plus efficace de ne relancer les tests que pour les tournées modifiées. A chaque itération et pour chaque client i, les valeurs de $CostDay(i, d)$ sont actualisées, puis agrégées en une seule valeur de coût global avec $Metrique1$. Le client i_{best} ayant le moindre coût global sera inséré dans les tournées donnant son meilleur coût $CostDay(i_{best}, h)$ sur les jours composant son scénario $\forall h \in Sc(i_{best})$. L'algorithme s'arrête si tous les clients sont insérés où si aucune insertion n'est possible.

5.5 Métaheuristiques en deux phases : Multi-Start Guided LNS

La méthode Multi-Start Guided LNS (GLNS) est une métaheuristique hybride à deux phases. L'objectif de la première phase est d'améliorer les tournées des VE sans modifier les scénarios et la deuxième phase vise à modifier le niveau de planification en modifiant les scénarios pour certains clients. L'objectif est d'alterner entre une phase d'amélioration des tournées (Recherche locale) et une phase de modification guidées des scénarios de certains clients(perturbations).

L'algorithme 19 fournit le pseudo-code de l'approche proposée. Le détail des étapes de l'algorithme est présenté dans les sous-sections suivantes. Le GLNS commence par une solution initiale S_0. La boucle principale est exécutée jusqu'à ce qu'un critère d'arrêt soit rempli (délai maximum ou nombre maximum d'itérations). Chaque itération de la boucle répète deux phases dont l'objectif a été donné ci-dessus.

La première phase utilise une recherche locale basée sur LNS, $LNS(S0, Adaptability, S1)$, appliquée pour améliorer de façon intensive le routage et la recharge sans modifier la planification. Au cours de cette phase du LNS, l'algorithme enregistre certaines valeurs pour calculer un indicateur $adaptability(i, h)$ qui évalue les scénarios choisis. La deuxième phase est une phase de perturbation permettant de changer les scénarios de certains clients. L'idée est d'utiliser l'indicateur $adaptability(i, h)$ pour décider quels clients doivent changer leurs scénarios. La solution modifiée sera utilisée comme nouvelle solution pour redémarrer le LNS.

Algorithm 19 Multi-Start Guided LNS

1: **Input :** une instance m-GPEVRP
2: **Output :** la meilleure solution trouvée S^*
3: Générer une solution initiale S_0
4: **while** Critère d'arrêt non atteint **do**
5: Executer LNS($S_0, Adaptability, S_1$)
6: Mettre à jour la meilleure solution S^*
7: Perturbation ($S_1, Adaptability, S_2$)
8: $S_0 := S_2$
9: **end while**

5.5.1 Phase 1 - LNS :

Notre recherche locale est basée sur le LNS. Le LNS part d'une solution initiale donnée et l'améliore en utilisant le processus Destroy-Repair. En effet, LNS retire un nombre relativement important de clients de la solution actuelle et tente de les réinsérer dans des positions différentes.

Cela conduit à une solution complètement différente, qui aide l'heuristique à échapper à l'optimum local. Dans l'approche proposée, nous utilisons le LNS comme une recherche locale de sorte que le nombre de clients éjectés doit rester relativement faible [16]. [93]. Le LNS ne doit pas modifier les patterns des clients. Nous avons choisi d'appliquer le LNS sur l'ensemble de l'horizon au lieu de l'appliquer au quotidien pour avoir une vision plus globale de l'impact lié à l'insertion d'un client en fonction d'un scénario donné. Le LNS reçoit en entrée la solution S_0 qui peut être une solution partielle (cas où $NS \neq \emptyset$). La liste NS doit alors être intégrée dans la liste des clients à éjecter. Ensuite, nous utilisons l'opérateur "Random Removal" pour éjecter le nombre nécessaire de clients, et "Regret insertion" comme opérateur de réparation. Ce choix est guidé par notre étude précédente qui a analysé plusieurs opérateurs de destruction et de réparation sur le PEVRP et a confirmé l'efficacité du Random removal et de l'insertion avec regret. Ces opérateurs utilisent les procédures spécifiques $AdjustIncreaseCharging$ et $repar$ pour gérer la contrainte énergétique.

Une nouvelle solution trouvée au cours du processus de destruction/réparation est acceptée si et seulement si elle est meilleure que la précédente. Dans ce cas, le processus Destroy-Repair se poursuit avec cette nouvelle solution.

A la fin de chaque itération LNS it, nous mesurons l'adaptation de chaque client i dans chaque jour h de son scénario $adapt(i,h)$ en calculant la distance à son voisinage proche (précédent et successeur de i dans la tournée du jour h). Nous calculons $adaptability(i,h)$ la moyenne de la distance calculée $adapt(i,h)$ sur toutes les itérations du LNS. Les clients les moins adaptés dans la solution actuelle sont ceux qui ont une valeur élevée de $adaptability(i,h)$. D'où l'utilité de changer le scénario pour ces clients dans la phase de perturbation afin de mieux guider le LNS lors du prochain redémarrage.

$$adapt(i,h) = d_{k,i} + d_{i,l} - d_{k,l} \tag{5.6}$$

où $k(l)$ est le successeur (prédécesseur) de i dans la tournée qui inclut i dans le jour h.

5.5.2 Phase 2 - perturbation

L'objectif de cette phase ($Perturbation(S_1, Adaptability(.))$) est de sélectionner les plus mauvais clients et de changer leurs scénarios en se basant sur la matrice $Adaptability$. Dans un premier temps, nous allons détecter les nc clients les plus "mauvais", pour modifier leurs scénarios (fonction $SelectLessAdapted(Adaptability, nc)$. Puis nous modifions les scénarios (fonction $ChangeAffectation(S_1, L)$).

$SelectLessAdapted(Adaptability, nc)$ retourne une liste L des paires (client i,jour h) les plus mauvaises qui devraient être exclues de la solution courante (voir Algorithme 19). Avoir le plus mauvais jour d'un client permet de guider le changement de scénario lors de la prochaine étape de perturbation.

La procédure $ChangeAffectation(S_1, L)$ permet de perturber la solution S_1 en supprimant tous les scénarios fixes impliqués dans L (sortie de la procédure $SelectLessAdapted()$) et les remplacer par un autre de leurs scénarios.

La fonction $SelectLessAdapted(Adaptability, L', nc)$ commence par créer une liste L' de toutes les paires (client, jour). Cette liste sera triée dans l'ordre décroissant de leurs scores dans la matrice $Adaptability$. Les premières paires de cette liste sont donc les plus mauvaises, mais il est nécessaire de filtrer cette liste pour obtenir la liste L. Les clients ayant un unique scénario sont ignorés puisqu'il n'est pas possible de leurs changer de scénario. Les clients ayant déjà été sélectionnés dans L ne le seront pas avec un deuxième jour, pour ne pas trop contraindre

Algorithm 20 SelectLessAdapted(Adaptability, L',nc)

Input : la matrice Adaptability,nc
Output : la liste L
Liste $L' := \{(i,h), \forall i \in C, \forall h \in Sc(i)\}$
Trier L' dans l'ordre décroissant de $Adaptibility(i,h)$
$L := \emptyset$, $Nbselect := 0$, $SelectedClient := \emptyset$
while $Nbselect < nc$ **do**
    Sélectionner $l = (i,h)$ le premier élement de L', ▷ i : client, h : jour
    supprimer l de L'
    if $(|comb(i)| > 1)$ & $(i \notin SelectedClient)$ **then**
        $Nbselect++$
        ajouter l à L, ajouter i à $SelectedClient$
    end if
end while
retourner L

le changement de scénario dans la prochaine étape et aussi pour avoir nc clients dont le scénario sera modifié.

Algorithm 21 $ChangeAffectation(S_1, L)$

Input : solution S_1', liste L ▷ $|L| = nc$
Output : solution S_2
$S_2 := S_1$
for all $(i,h) \in L$ **do** ▷ i : un client, h : un jour
    $List_{cb} := \emptyset$ ▷ $List_{cb}$: une liste des scénarios potentiels
    for all $cb \in comb(i)$ **do**
        if $h \notin cb$ **then**
            $List_{cb} = List_{cb} \cup cb$
        end if
    end for
    if $List_{cb} \neq \emptyset$ **then**
        choisir le nouveau scénario $cb^*(i)$ aléatoirement de $list_{cb}$
        supprimer le client i de S_2
        Réinsérer le client supprimé en respectant le scénario cb^* in S_2
    end if
end for
retourner S_2

Pour chaque paire (i,h) de L, la fonction $ChangeAffectation$ parcourt tous les scénarios de i et stocke ceux n'incluant pas h dans $list_{cb}$. Parmi tous les scénarios dans $list_{cb}$, la fonction choisit aléatoirement un scénario et réinsère le client i en respectant ce nouveau scénario.

5.6 Métaheuristiques à voisinage

Dans cette section nous présentons deux métaheuristiques à voisinage (LNS et ALNS). Nous avons utilisé ces deux métaheuristiques pour le problème PEVRP.

Nous avons repris le même fonctionnement du LNS utilisé pour la problématique à fréquence unique. Nous avons donc choisi ($Random.removal$, $Regret^2$) comme paire d'opérateurs. Pour la métaheuristique $ALNS$ nous reprenons les mêmes opérateurs et le même système de scores et de poids pour sélectionner la paire d'opérateurs à chaque itération. Pour les deux métaheuristiques, nous reprenons comme critère d'acceptation la règle de Metropolis et comme fonction d'évaluation des solutions la fonction de coût complet. Les opérateurs nécessitent une adaptation puisque un client ne se résume plus à une seule visite sur l'horizon mais à autant de visites que sa fréquence.Dans ce qui suit, nous expliquons les opérateurs tels que modifiés pour cette nouvelle problématique.

5.6.1 Opérateurs de destruction

L'objectif de ces opérateurs est de supprimer un nombre fixe γ de clients. Nous reprenons ici les trois opérateurs de destruction : le random removal, le worst removal, et le cluster removal. Le choix des clients à supprimer reste le même, mais la procédure de suppression change puisqu'il est nécessaire de supprimer le client dans tous les jours composant son scénario.

Random removal $(S, S', f(S'))$

L'opérateur choisit aléatoirement γ clients de la solution puis supprime chacun des clients dans tous les jours composant son scénario. Toutes les tournées modifiées par ces suppressions seront optimisées en termes de recharge avec la procédure $AdjustDecreaseCharging(Tr)$.

Cluster removal $(S, S', f(S'))$

L'opérateur commence par choisir aléatoirement un client i, puis les $\gamma - 1$ clients les plus proches de i sont sélectionnés. Ces clients sont choisis sur la carte géographique des clients sans prendre en compte les tournées où ils sont visités, leurs fréquences ou scénarios. La procédure $AdjustDecreaseCharging(Tr)$ est appliquée à chaque tournée impactée par l'opérateur.

Worst removal $(S, S', f(S'))$

Pour chaque client $i \in S$, l'algorithme simule la suppression de i dans toutes les tournées où il est visité. Le gain de coût global est calculé en utilisant une des trois métriques définies. Le client i^* dont la suppression produit le plus grand gain est choisi. L'opérateur répète les mêmes étapes jusqu'à supprimer γ clients.

5.6.2 Opérateurs de construction

L'objectif est de réinsérer les clients supprimés par l'opérateur de destruction. Si l'insertion d'un client cause la violation de la contrainte d'énergie, la procédure *repar* est utilisée pour tenter la réparation de la tournée.

Nous considérons toujours deux stratégies d'insertion d'un client i dans une tournée Tr :

— $InsertC^{All}$ qui teste l'insertion du client i à toutes les positions de la tournée $Tour$, nous avons donc $|Tr| - 1$ tests d'insertion.

— $InsertC^2$ qui choisit les deux clients visités les plus proche de i dans la tournée Tr, et teste l'insertion du client i avant et après chacun d'eux.

First Improvement insertion

L'opérateur sélectionne un sommet dans la liste L des clients éjectés et l'insère dans chaque jour de son scénario. Pour chaque jour, la tournée qui génère l'augmentation minimale des coûts est choisie pour l'insertion du client. Pour identifier la tournée minimisant le coût d'insertion dans un jour donné, l'opérateur teste l'insertion dans toutes les tournées en utilisant la procédure *repar* si besoin.

Best Improvement insertion

Soit L la liste des clients éjectés. L'opérateur répète les trois étapes ci-dessous jusqu'à insérer tous les clients ou si aucune insertion n'est possible. L'étape 1 trouve l'insertion de coût minimum de chaque client $i \in L$ dans chaque jour de son scénario. L'étape 2 calcule en utilisant une métrique, le coût global d'insertion de chaque client. L'étape 3 choisit le client générant le moindre coût global et l'insère dans chaque jour de son scénario dans les tournées générant le moindre coût.

Regret insertion

L'opérateur teste l'insertion de tous les clients dans chacun des jours de leurs scénarios puis mémorise les deux meilleures insertion par jour. Le coût global des meilleures insertions, et le coût global des secondes meilleures insertions sont calculés. L'opérateur choisit le client ayant le plus grand écart entre les deux valeurs, puis l'insère dans les tournées de chaque jour de son scénario.

5.7 Expérimentations

Nos méthodes sont mises en œuvre en utilisant C++. Tous les calculs sont effectués sur un processeur Intel Core (TM) i7- 5600U, à 2,60 GHz, avec 8 Go de mémoire RAM.

5.7.1 Génération des instances

Pour tester nos méthodes, nous généralisons les instances à 100 clients proposées pour le PEVRP .Nous générons des fréquences de visites dans l'ensemble $\{1 \dots 5\}$ et nous fixons les scénarios de chaque client en respectant la distribution fournie dans le tableau 5.4. Chaque scénario est ensuite généré de manière aléatoire par rapport à la fréquence du client.

Après différents tests, nous avons fixé le nombre de véhicules à 5. Afin d'évaluer toutes les méthodes dans les mêmes conditions, nous avons considéré comme critère d'arrêt un délai de deux heures de calcul pour les métaheuristiques.

Fréquence	**% de clients**	**nombre de scénarios**
1	30	2
2	25	3
3	20	3
4	15	2
5	10	1

TABLE 5.4 – Génération des fréquences

	C_f	CPU	CNV
$Hconst1$	6 634,26	30,18	4,00%
$Hconst2$	6 641,58	435,54	6,00%
$Hconst3$	6 641,58	435,42	6,00%
$Hconst4$	6 641,58	435,69	6,00%

TABLE 5.5 – Évaluation des heuristiques

	$\gamma = 5$	$\gamma = 7$	$\gamma = 10$
Best Solution Occurrence	21	**31**	0
Average Gap	0,82%	**0,63%**	11,91%
Average Gap Without Best	**1,38%**	1,55%	11,91%
Average Number of Restarts	48	35	27

TABLE 5.6 – Multi-start GLNS results

5.7.2 Évaluation et comparaison des heuristiques de construction

Pour évaluer et comparer les quatre heuristiques de construction, nous les avons lancées sur l'ensemble des instances décrites précédemment. Nous avons choisi trois indicateurs :

— f_c : le coût complet moyen des solutions obtenus par heuristiques
— CPU : le temps d'exécution moyen des heuristiques sur les 100 instances
— CNV : Pourcentage moyen de clients non visités par heuristique

On voit dans la Table 5.5 que les trois heuristiques de meilleure insertion ($Hconst2$, $Hconst3$, $Hconst4$) trouvent les mêmes solutions. Ceci permet de dire que les heuristiques de construction ne sont pas sensibles au changement de métrique. En revanche, les résultats de $Hconst1$ (heuristique d'insertion par liste) sont meilleurs sur les trois indicateurs. Nous considérerons dans ce qui suit, cette solution comme solution de départ des différentes métaheuristiques.

5.7.3 Paramétrage de la métaheuristique GLNS

Nous voulons définir le paramètre *gamma* de la phase de recherche locale dans le GLNS. Ce paramètre γ étant le nombre de clients à supprimer-réinsérer à chaque itération. Nous avons proposé et testé trois valeurs de γ, à savoir, 5, 7 et 10. Pour chaque valeur, nous avons obtenu les paramètres suivants : (i) "Best Solution Occurrence" indique le nombre de fois où la méthode fournit une meilleure solution, (ii) "Average Gap" indique l'écart moyen par rapport à la meilleure solution obtenue sur tous les tests, (iii) "Average Gap Without Best" renvoie l'écart moyen par rapport à la meilleure solution obtenue dans les cas où la meilleure solution n'est pas obtenue, et (iv) "Average Number of Restarts" le nombre moyen de lancements du LNS. On peut voir dans le tableau 5.6 que $\gamma = 7$ donne de meilleurs résultats pour le nombre de meilleures solutions trouvées et pour l'écart moyen.

5.7.4 Évaluation et comparaison des métaheuristiques à voisinage

Dans cette section, nous comparons les résultats de l'ALNS et du LNS. Dans le tableau (5.7), nous avons indiqué quatre mesures, les trois premières étant les mêmes que dans l'expérience des

	ALNS	LNS
Best Solution Occurrence	11	**41**
Average Gap	1,91%	**0,41%**
Average Gap without Best	2,42%	**1,96%**
Average Best Iteration	194	**166**

TABLE 5.7 – ALNS/LNS comparaison

	GLNS7	GLNS7.random	LNS	ALNS
Best Solution Occurrence	14	4	62	20
Average Gap	3,3%	5,1%	0,8%	2,5%
Average Gap Without Best	3,8%	5,3%	2,0%	3,1%

TABLE 5.8 – Evaluation finale des métaheuristiques

GLNS et la quatrième étant le nombre moyen d'itérations où la meilleure solution est atteinte. Nous pouvons voir que le LNS donne la meilleure solution pour 80 % des cas, et qu'elle est au maximum de 1,96 % de la meilleure solution trouvée par l'ALNS dans les autres cas. Ces résultats montrent qu'une approche simple comme celle du LNS réussit à être plus compétitive que celle de l'ALNS, qui utilise un mécanisme adaptatif pour choisir ses opérateurs. Cela peut s'expliquer par le fait que le problème est très difficile en raison des contraintes d'énergie, de routage et de planification, et que le nombre de solutions irréalisables peut être très important. Cela peut limiter l'espace de recherche et n'a pas permis à ALNS de dépasser les résultats de LNS.

5.7.5 Évaluation finale

Nous avons vu que GLNS7 (GLNS avec $\gamma = 7$) donne les meilleurs résultats sur les instances testés. Pour évaluer la partie perturbation de cette métaheuristique, nous avons implémenté une version du GLNS où le changement de scénario se fait aléatoirement. Nous appellerons cette version GLNS7.random

D'autres part nous avons les deux métaheuristiques à voisinage qui donnent de bons résultats. Nous avons donc lancé les quatre métaheuristiques (GLNS7,GLNS7.random, LNS, ALNS) sur les 100 instances. Nous avons considérés trois paramètres (Best Solution Occurrence, Average Gap, Average Gap Without Best). Les résultats dans la Table 5.8 montrent que la méthode LNS est performante pour trouver la meilleure solution parmi les autres méthodes ($\simeq 60\%$) et aussi pour avoir le être la plus proche lorsque les autres méthodes trouvent une meilleure solution. Comme mentionné précédemment, une forte intensification est plus adaptée et utile pour ce problème. En outre, les trois classes de contraintes étant fortement dépendantes, la gestion séparée de chacune d'entre elles, comme dans le GLNS, limite les solutions à explorer et ne donne pas de bons résultats. En résumé, le changement de modèles dans la première méthode et le changement d'opérateurs dans le GLNS réduit le temps d'exploration du voisinage.

Sur l'évaluation de notre phase de perturbation, nous pouvons voir que GLNS7 donne de meilleurs résultats que GLNS7.random ce qui nous permet de dire que notre méthode de perturbation basée sur l'historique de performance est efficace.

5.8 Conclusion

Dans ce chapitre nous avons présenté le m-GPEVRP, une extension du PEVRP où la fréquence de visites des clients est supérieure ou égale à 1. Dans un premier temps, nous avons défini et modélisé cette problématique. Puis, nous avons développé quatre heuristiques de construction, et trois métaheuristiques.

Les heuristiques proposées fonctionnent en deux temps : les scénarios des clients sont choisit puis les tournées sont construites selon ses scénarios. La méthode d'insertion différencie les quatre heuristiques. La première heuristique insère les clients par liste alors que les trois autres heuristiques choisissent la meilleure insertion à chaque itération. L'évaluation d'une insertion se base sur une des trois métriques présentées.

La première métaheuristique est la "Multi-Start Guided LNS". Cette métaheuristique fonctionne en deux phases : modification/fixation des scénarios, puis optimisation des tournées par une méthode LNS. L'objectif est d'alterner entre une phase d'amélioration des tournées par une LNS et une phase de perturbations guidées des scénarios de certains clients.

La deuxième et troisième métaheuristiques développées sont des métaheuristiques à voisinage. Une métaheuristique LNS et une métaheuristique ALNS adaptées au m-GPEVRP.

Pour les expérimentations, nous avons généré des instances pour le m-GPEVRP à partir des instances pour le PEVRP. Ces instances nous ont permis d'évaluer les différentes méthodes de résolution.

Entre les heuristiques, c'est l'heuristique d'insertion par liste qui a donné les meilleurs résultats devant les trois autres heuristiques.

Dans un premier temps, nous avons fixé les différents paramètres des trois métaheuristiques puis nous avons comparé leurs résultats. La méthode LNS trouve une meilleure solution que les autres métaheuristiques sur 62% des instances et trouve une solution seulement 2% plus mauvaise sur le reste des instances. De même que pour le PEVRP, la méthode LNS est efficace par une intensification plus forte que l'ALNS. Par rapport à la méthode GLNS, on peut conclure que la considération séparée des contraintes ne donne pas de bons résultats tant les contraintes sont inter-dépendantes.

Chapitre 6

Problème de planification des tournées de véhicules électriques avec multi-visites sur l'horizon et flotte illimitée

Sommaire

6.1 Introduction

Ce chapitre propose une version plus générale du problème de planification des tournées de véhicules électriques, nommé GPEVRP (Generalized Periodic electric vehicle routing problem). Le GPEVRP considère le m-GPEVRP étudié dans le chapitre précédent en relaxant la contrainte de limitation du nombre de véhicules. Pour la définition de la problématique, voir la Section 5.2. Nous nous plaçons ici dans un contexte où le décideur a besoin d'identifier le nombre de tournées nécessaires pour effectuer la planification de ces tâches de livraison ou de collecte, pour ensuite décider du nombre de véhicules à réserver à ses tâches. Nous avons choisi dans ce chapitre d'utiliser une autre classe de métaheuristique, qui sont les algorithmes génétiques, car les

algorithmes génétiques ont été très peu utilisés pour le EVRP et ses variantes. Les seuls travaux que nous avons identifiés sont ceux de [77], qui ont proposé un algorithme génétique hybridé avec une recherche locale adaptative pour le GVRP, la version de base des problèmes de tournées de véhicules électriques. Dans ce chapitre, nous présentons le problème étudié ; et notre algorithme génétique qui utilise plusieurs opérateurs de croisement et de mutation, dont une mutation à base de recherche locale à grand voisinage (LNS). L'hybridation de l'algorithme génétique avec la méthode LNS a été motivée par les résultats encourageants de LNS sur les autres variantes présentées dans les chapitres précédents.

6.2 Schéma général de l'algorithme génétique pour le GPEVRP

Les algorithmes génétiques (Genetic Algorithm ou GA) ont été introduits initialement par John Holland en 1960 sur la base du concept de la théorie de l'évolution, puis développés par David Goldberg en 1989 [46]. Les GA ont été depuis utilisés avec un succès croissant en optimisation combinatoire. Les algorithmes génétiques tentent de simuler le processus d'évolution des êtres vivants. Ils utilisent un vocabulaire similaire à celui de la génétique naturelle, bien que les processus naturels auxquels ils font référence soient beaucoup plus complexes. On parlera ainsi d'individus dans une population. Pour un problème donné, un individu représente une solution, à laquelle on associe une mesure d'adaptation (fitness). Un individu est composé d'un ou plusieurs chromosomes, eux-mêmes constitués de gènes qui contiennent les caractères héréditaires de l'individu. Pour un problème d'optimisation combinatoire, un individu est simplement une solution et son fitness est le coût de cette solution (critère à optimiser). Les principes de sélection, croisement et mutation s'appuient sur les processus naturels de même nom. La population est codée en général comme un tableau d'individus. La sélection a pour but de choisir deux individus-parents dans la population, en favorisant les plus adaptés. Le croisement ou crossover combine les individus choisis pour former un ou deux nouveaux individus appelés enfants. C'est lui qui assure l'exploration de l'espace des solutions. La mutation modifie aléatoirement et avec une faible probabilité les enfants. Elle apporte une certaine diversification visant à éviter une convergence prématurée.

Nous développons ici un GA capable de combiner des solutions en modifiant simultanément des décisions tactiques (affectation des scénarios de jours aux clients) et des décisions opérationnelles (composition des tournées dans chaque période). Nous proposons une modélisation de la solution sous forme de tours géants (un tour géant par jour de l'horizon). Une procédure de type Split [82], adaptée à notre problème, nommée $SPLITGPEVRP$ sera utilisée pour découper le tour géant en tournées et en réparant les tournées si c'est nécessaire.

Le principe général du GA proposé est présenté dans l'Algorithme 22. Il adapte les composants du GA aux contraintes particulières de GPEVRP.

Algorithm 22 GA pour le GPEVRP

```
1: Generer() : Générer la population initiale Popul
2: for all (solution S ∈ Popul) do
3:     SplitGPEVRP(S)
4: end for
5: while (critère d'arrêt non atteint) do
6:     for i=1 to NbSelect do
7:         Select() : Choisir deux parents P1 et P2 avec tournoi binaire
8:         Croisement() : Générer les enfants C1 et C2
9:         Mutation() : Appliquer la mutation aux enfants générés
10:    end for
11:    Évaluer les nouvelles solutions
12:    Remplace() : Générer la population suivante
13: end while
14: Retourner la meilleure solution obtenue
```

L'algorithme commence par générer la population initiale de manière aléatoire. Ensuite, le GA est exécuté un certain nombre d'itérations. Chaque itération du GA répète les étapes suivantes qui seront détaillées ci-dessous. Tout d'abord, une heuristique de construction aléatoire est appliquée pour générer la population initiale. Ensuite, chaque solution générée est évaluée et réparée si les contraintes d'énergie sont violées grâce à une adaptation de la procédure *Split* [82]. Le tournoi binaire est utilisé pour sélectionner *Nbselect* couple(s) de solutions candidates à la reproduction. Un opérateur de croisement est appliqué aux parents sélectionnés pour produire deux nouvelles solutions, appelées enfants. Les enfants sont mutés selon une probabilité *pmut* en utilisant un opérateur de mutation, puis les enfants sont évalués et réparés si nécessaire. Nous avons proposé quatre opérateurs de croisement et quatre opérateurs de mutation pour le GPEVRP. Finalement, une approche élitiste est appliquée pour permettre le passage d'une génération à une autre. La meilleure solution est restituée lorsque le critère d'arrêt du GA est atteint.

6.3 Les composants du GA proposés pour le GPEVRP

6.3.1 Chromosome

Une solution du GPEVRP est codée sous forme d'une longue séquence de clients à servir. Cette séquence contient $ns = \sum f(i)$ éléments (nombre total de services des clients sur l'horizon) et est formée de np sous-listes successives (une par jour), séparées par des délimiteurs de jour. Chaque client i apparaît $f(i)$ fois et au plus une fois dans la sous-liste d'un jour donnée. Les $f(i)$ jours où i apparaît doivent former un scénario faisable pour i. La Figure 6.1 donne un exemple.

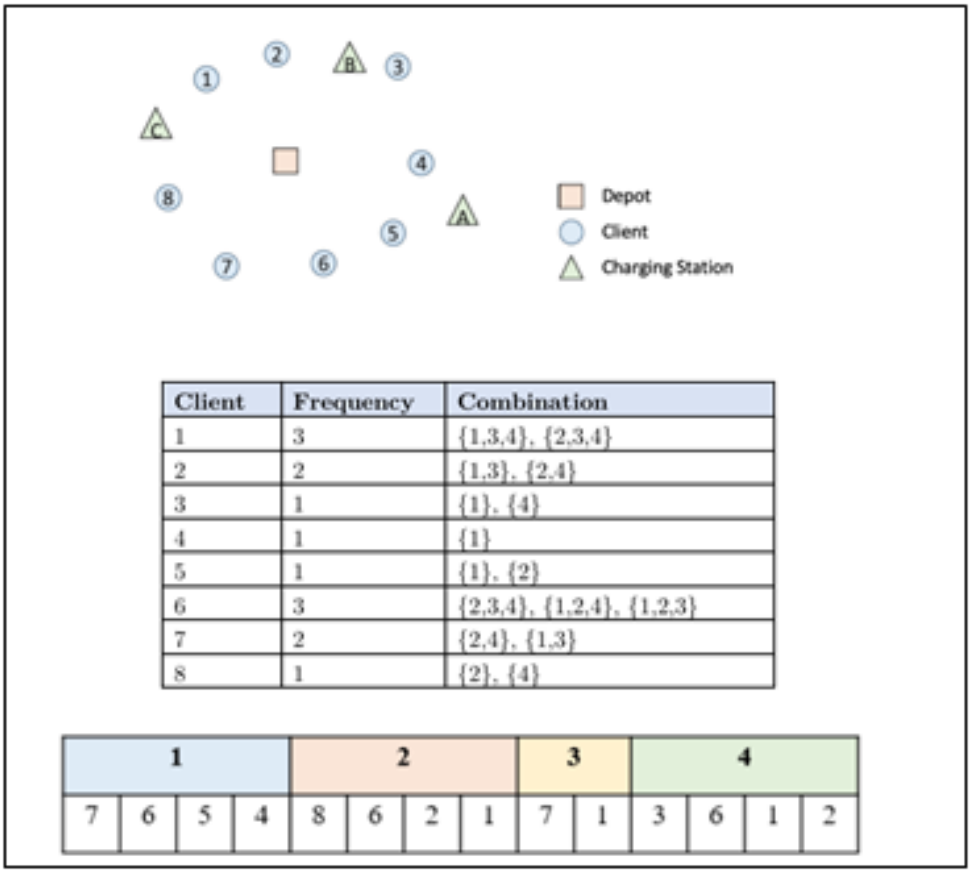
Un chromosome peut être interprété physiquement comme un ordre de priorité pour un véhicule traitant tous les clients sur l'horizon, jour après jour. Une sous-liste pour un jour donné correspond à une suite de clients sans délimieurs de tournées. Il s'agit d'un tour géant qu'un véhicule pourrait parcourir en ignorant la capacité des véhicules.

6.3.2 Évaluation du chromosome : *Split* adapté au GPEVRP

Pour évaluer un chromosome, nous utilisons la méthode *SplitGPEVRP* qui adapte la procédure *split* [82], connue pour son efficacité sur les problèmes de tournées de véhicules, à notre étude. Cette méthode est détaillée dans l'algorithme 23.

Algorithm 23 *SplitGPEVRP(Ch)*

Input : chromosome Ch
Output : solution Sol
for all jour d in horizon H **do**
    Extraire Ch_d vecteur des clients du jour d
    for all client $cl \in Ch_d$ **do**
        $Label[cl] = \infty$
        $Pred[cl] = 0$
    end for
    $i = 1$; $n = CH_d.size()$
    while $i \leq n$ **do**
        $j = i$, $continue = true$;
        while $j \leq n$ and $continue$ **do**
            Créer la tournée $Tr = \{\bar{0}, Ch_d[i], .., Ch_d[j], \underline{0}\}$
            if $Temps(Tr) < T_{max}$ et $Demande(Tr) < Q$ **then**
                if $GetMinEnergy(Tr) < 0$ **then**
                $Tr = Repar2(Tr, \kappa, Option)$
                end if
                if $GetMinEnergy(Tr) > 0$ And $Label[Ch_d[j] < Cost(Tr)$ **then**
                $Label[Ch_d[j]] = Cost(Tr)$;
                $Pred[Ch_d[j]] = Ch_d[i]$
                end if
            else
                $continue = false$;
            end if
        end while
    end while
    Construire Sol_d la solution partielle du jour d à partir des vecteurs $Label$ et $Pred$
end for
$Sol = \cup_{d \in H} Sol_d$
Retourne Sol



Client	Frequency	Combination
1	3	{1,3,4}, {2,3,4}
2	2	{1,3}, {2,4}
3	1	{1}, {4}
4	1	{1}
5	1	{1}, {2}
6	3	{2,3,4}, {1,2,4}, {1,2,3}
7	2	{2,4}, {1,3}
8	1	{2}, {4}

1				2				3		4			
7	6	5	4	8	6	2	1	7	1	3	6	1	2

FIGURE 6.1 – Codage du chromosome pour le GPEVRP

L'algorithme $SplitGPEVRP$ minimise le coût total de la séquence des clients définis par le chromosome, et découpe chaque tour géant en tournées faisables. Chaque tournée faisable doit respecter les contraintes de durée totale, de capacité et d'énergie. Si la tournée en cours d'évaluation ne respecte pas la capacité ou la durée totale, la tournée est éjectée. Si la tournée en cours d'évaluation est faisable en termes de contraintes de capacité et de temps, mais ne respecte pas la contrainte d'énergie, une procédure de réparation nommée $Repar2$ est appliquée pour réparer la tournée en insérant des stations de recharge et en mettant à jour l'évaluation de la tournée. La Figure 6.2 représente le schéma général de fonctionnement de la méthode Split.

L'algorithme $SplitGPEVRP(ch)$ reçoit en entrée un chromosome ch et retourne la meilleure solution possible Sol. Pour chaque jour de l'horizon, l'algorithme génère toutes les tournées réalisables. La labellisation des clients (vecteur $Label$) permet de calculer le plus court chemin sans générer explicitement le graphe auxiliaire G_a. Pour chaque tournée potentielle examinée dans $SplitGPEVRP$, la fonction $GetMinEnergy$ (cette fonction a été présentée dans la Section 3.4), permet de vérifier si la tournée est faisable en termes d'énergie ; et la fonction $Repar2$ (qui sera définie plus loin) permet de réparer, dans la mesure du possible, cette tournée. Une fois toutes les tournées réalisables énumérées, l'algorithme choisit le scénario de tournées visitant tous les clients et générant le moindre coût en calculant le plus court chemin sur le graphe auxiliaire G_a. La Figure 6.3 montre un exemple de graphe auxiliaire H.

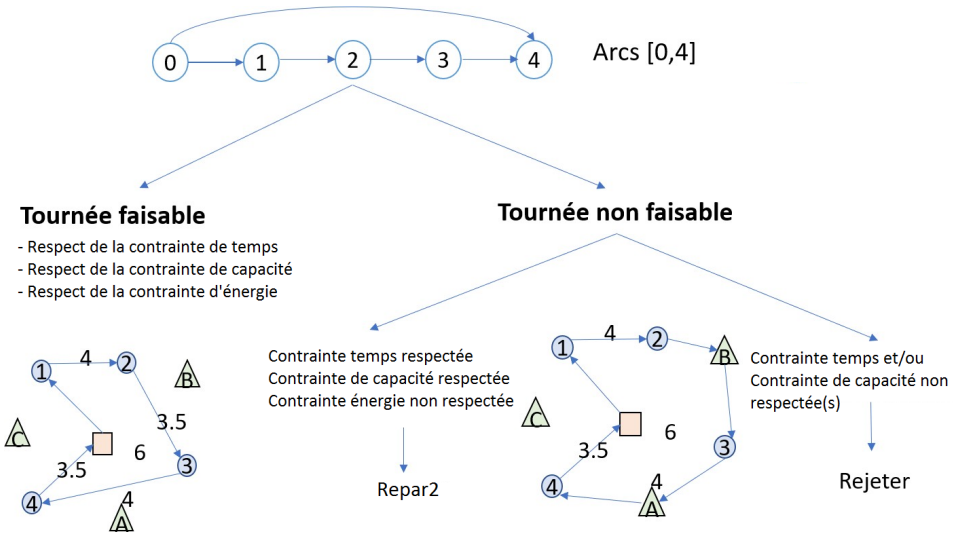


FIGURE 6.2 – Principe générale de la méthode *SplitGPEVRP*

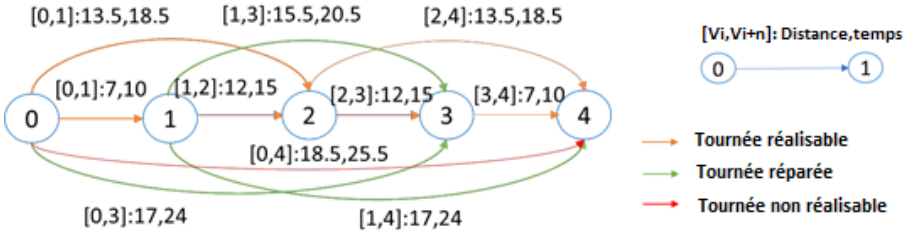


FIGURE 6.3 – *SplitGPEVRP* - exemple d'un graphe auxiliaire

La réparation des tournées dans Split

Les approches à base de LNS, que nous avons introduites dans les chapitres précédents, utilisent une fonction *Repar* (voir Section 3.4) qui tente de réparer une tournée en insérant au maximum une station de recharge dans la tournée. Cette fonction n'était appliquée que sur des tournées ou la contrainte d'énergie est violée par l'insertion d'un seul client. Il est donc plus rentable de tester l'insertion de ce client dans une autre tournée que de chercher à insérer plus d'une station dans la même tournée.

Dans le cas qui se présente ici, la fonction *Repar*2 que nous proposons sera appliquée sur une tournée (une séquence de clients), il est donc nécessaire de considérer la possibilité d'insérer plusieurs stations de recharge, mais le nombre de stations à insérer doit rester raisonnable pour ne pas engendrer un coût de recharge trop élevé et/ou un dépassement de la durée maximale d'une tournée. Quelques tests préliminaires ainsi que l'examen des solutions obtenues dans les chapitres précédents nous ont permis de fixer ce nombre de stations à deux stations, au maximum, par tournée. Ce choix nous permet aussi d'avoir une fonction *Repar*2 raisonnable en termes de complexité.

La fonction *Repar*2 commence par déterminer les stations de recharge atteignables à partir de chaque client, ce qui permet de sélectionner un sous-ensemble de stations atteignables au lieu de tester toutes les stations disponibles. Nous définissons un seuil d'énergie, c'est-à-dire que les stations de recharge qui seront sélectionnées sont limitées à des cercles dont le rayon est

la distance maximale qui peut être atteinte en dessous du seuil d'énergie. Ensuite, si plusieurs réparations sont possibles, la fonction nécessite un critère lui permettant de choisir parmi celles-ci. Nous avons considéré trois critères de coût. Soit une station b à insérer entre deux clients i et j, sachant que i est le prédécesseur de j dans la séquence, et d_{ij} la distance entre deux nœuds i et j, alors le premier critère $C1 = d_{ib} + d_{bj} - d_{ij}$, est le critère le plus classique, l'idée est de minimiser la distance en amont et en aval de la station. Le deuxième critère $C2 = d_{bj}$ ne considère que la distance en aval de la station b, ce qui permet de maximiser l'énergie injectée dans la tournée. Le troisième critère $C3 = 0,5 \times C1 + 0,5 \times C2$ utilise une pondération entre les deux précédents critères.

Afin d'optimiser les temps de calcul, nous avons aussi proposé d'évaluer 5 stratégies d'insertion (nommée $Option_1$ à $Option_5$). Ces stratégies permettent de limiter le nombre de positions d'insertion à tester pour réparer la tournée. Nous avons illustré ces options à l'aide de l'exemple de la Figure 6.3. Dans cet exemple, la tournée à réparer est donnée par l'arc infaisable $(0, 4)$ qui représente la séquence de tournée (dépôt ,1, 2, 3, 4, dépôt).

— $Option_1$: cette stratégie considère toutes les positions d'insertion possibles pour toutes les stations de recharge sélectionnées. Par exemple, dans la Figure 6.3, il faut tester toutes les positions entre deux nœuds successifs de la séquence (dépôt,1,2,3,4, dépôt)

— $Option_2$: lors de l'évaluation d'une tournée non faisable représentée par l'arc (i, j) dans $SplitGPEVRP$, au lieu de procéder comme dans $Option_1$, nous proposons d'utiliser la solution réparée de $(i, j-1)$ (si elle existe), c'est-à-dire de considérer les stations de recharge déjà insérées dans la tournée représentée par l'arc $(i, j-1)$ et la compléter si besoin par de nouvelles stations, dont l'insertion sera testée uniquement entre les nœuds $\{j-1, j\}$ et entre $\{j, depot\}$. Ceci permet d'utiliser l'historique sur les arcs réparés et de réduire le nombre de positions d'insertion à explorer. Si la solution de réparation retenue pour l'arc $(i, j-1)$ permet de réparer la tournée de l'arc (i, j), la réparation de l'arc (i, j) utilisera la même réparation que l'arc $(i, j-1)$, donc aucune nouvelle station ne sera insérée. Si la solution de réparation retenue pour l'arc $(i, j-1)$ ne permet pas de réparer la tournée de l'arc (i, j), la réparation de l'arc (i, j) utilisera la réparation de l'arc $(i, j-1)$, et on cherchera de nouvelles stations à insérer entre les noeuds $\{j-1, j\}$ et $\{j, depot\}$.Par exemple, pour réparer l'arc $(0, 4)$ de la Figure 6.3, nous considérons la solution de l'arc réparé $(0, 3)$ et si l'ajout du client 4 ne permet pas de garder la faisabilité, nous chercherons à insérer une autre station de recharge dans les dernières positions soit entre les nœuds $\{3, 4\}$ et $\{4, 0\}$.

— $Option_3$ et $Option_4$: l'idée ici est de réduire le nombre de positions à tester afin d'optimiser le temps de calcul. Au lieu de passer en revue toutes les positions d'insertion de la tournée en cours d'évaluation, dans $Option_3$, nous conservons les 2 dernières positions possibles dans la tournée, et les 3 dernières positions possibles pour $Option_4$.
Dans l'exemple de la Figure 6.3, cela revient à tester les positions entre les nœuds $\{3.4\}$ et les noeuds $\{4.0\}$ pour $Option_3$, et à tester les positions entre les nœuds $\{2.3\}$, $\{3.4\}$ et $\{4.0\}$ pour $Option_4$.

— $Option_5$: nous envisageons la possibilité d'insérer des stations de recharge à partir du moment où le véhicule atteint un seuil donné de niveau de batterie (par exemple 50% de l'énergie).

6.3.3 Population initiale et méthode de sélection

La population est un tableau de $MaxPopul$ chromosomes. Elle contient des chromosomes aléatoires générés par l'algorithme 24. La génération d'un individu commence par le tri aléatoire

des clients dans un vecteur. Pour chaque client, un de ces scénarios est choisi aléatoirement. Dans l'ordre du vecteur, les clients sont insérés dans chaque jour de l'horizon du scénario choisi. La sélection des individus de la population se fait par tournoi (Tournament selection ou TS). Cette méthode a été utilisée largement sur les problèmes de tournées avec succès [38].

Algorithm 24 *Generer*()

```
1: Input : MaxPopul : taille de la population
2: Output : Popul : tableau de population générée
3: for i=1 to MaxPopul do
4:     Générer les séparateurs de périodes ;
5:     Trier les clients aléatoirement dans le vecteur T
6:     for i = 1 to n do
7:         Prendre le premier client T[i]
8:         Choisir aléatoirement un scénario sc dans Comb(T[i]) ;
9:         for all jour j ∈ sc do
10:            Insérer client T[i] à la fin de la journée j.
11:        end for
12:    end for
13: end for
```

La méthode de tournoi la plus commune est le tournoi binaire, où on choisit deux individus aléatoirement ($S = 2$) puis on sélectionne le meilleur (qui a la valeur de fitness la plus élevée). Les individus sélectionnés sont ensuite recombinés pour générer de nouveaux individus (enfants), selon divers processus décrits dans la prochaine section. L'Algorithme 25 présente le pseudo-code de cette méthode. La procédure permet de sélectionner un nombre d'enfant *Nbselect* sur *Pop* la population de solution et retourne *TabSelect* le vecteur des solutions sélectionnées. Si *Nbselect* est supérieur à 2, on exécute plusieurs fois le tournoi binaire.

Algorithm 25 *Selection*

```
1: Input : NbSelect,Population Pop
2: Output : Les solutions sélectionnées TabSelect
3: for i = 1 To NbSelect do
4:     Choisir 2 solution p1 et p2 aléatoire parmi Pop
5:     if p1.fitness ≥ p2.fitness then
6:         Insérer p1 dans TabSelect
7:     else
8:         Insérer p2 dans TabSelect
9:     end if
10: end for
```

6.3.4 Croisement

Le croisement a pour but de découvrir de nouvelles solutions en combinant des chromosomes. Classiquement, les croisements sont conçus pour deux parents $P1$ et $P2$ et génèrent deux enfants $C1$ et $C2$. Nous proposons pour le GPEVRP quatre croisements, ces croisements ont pour but de préserver la propriété d'affectation aux jours (les scénarios) et/ou la propriété d'ordre dans un jour donné.

Croisement $Affect$

Ce croisement préserve l'affectation des clients aux jours sans tenir compte de l'ordre dans la tournée. Étant donné deux parents $P1$ et $P2$ avec ns éléments, le croisement $Affect$ génère deux points de coupures a et b avec $1 \leq a \leq b \leq ns$. Pour obtenir le premier enfant $Child1$, ce croisement copie $P1(a)$ à $P1(b)$ dans $Child1$ en respectant les scénarios de chaque client dans $P1$. $P1(i)$ sera inséré dans chaque jour de son scénario à la dernière position dans la séquence du jour sélectionné. $P2$ est ensuite balayé de gauche à droite et les tâches manquantes dans $Child1$ sont insérées selon leur scénario dans $P2$. $Child2$ est généré de la même manière en inversant les rôles de $P1$ et $P2$.

La Figure 6.4 montre un exemple de croisement $Affect$ avec comme points de croisement $a = 5$ et $b = 8$, les clients du segment $(5, 6, 7, 1)$ sont insérés suivant cet ordre dans le chromosome $Child$ dans chaque jour de leur scénario. Par exemple, les clients 1 et 6 étant visités le jour 3 dans $P1$, ils seront visités dans $Child$ le jour 3 dans l'ordre $(6, 1)$ défini par le segment. Le parcours de $P2$ de gauche à droite donne le segment $(4, 8, 2, 3)$ de clients non visités à insérer dans $Child$. Ces clients seront insérés dans chaque jour selon leur scénario dans $P2$. Par exemple, le client 4 sera inséré en premier le jour 1 après les clients de $P1$.

client	Fréquence(client)	Combinaison(client)
1	3	{1,3,4}, {2,3,4}
2	2	{1,3}, {2,4}
3	1	{1}, {4}
4	1	{1}
5	1	{1}, {2}
6	3	{2,3,4}, {1,2,4}, {1,2,3}
7	2	{2,4}, {1,3}
8	1	{2}, {4}

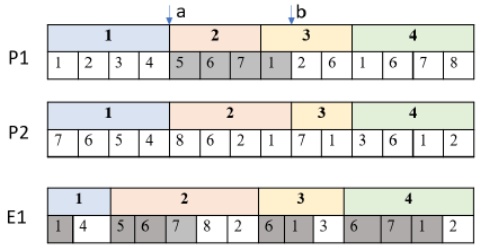


Figure 3.6 Exemple de croisement

FIGURE 6.4 – Exemple de croisement $Affect$

Croisement $AffectInsert$

Le croisement $AffectInsert$ vise à conserver les propriétés d'affectation et d'ordre des deux parents. Il suit les mêmes étapes que le croisement précédent, sauf qu'au lieu d'insérer $P2(i)$ à la dernière position dans la séquence de la période sélectionnée de $Child1$, ce croisement recherche la meilleure position de $P2(i)$ dans $Child1$ tout en respectant l'ordre de $P2(i)$ dans $P2$. L'insertion de $P2(i)$ doit alors maximiser le nombre de prédécesseurs et de successeurs communs entre $Child1$ et $P2$.

Pour mieux expliquer ce croisement, nous reprenons le même exemple que dans la Figure 6.4 et nous appliquons le croisement $AffectInsert$ dans la Figure 6.5. Comme dans le croisement $Affect$, les clients $(5, 6, 7, 1)$ sont insérés dans $Child$ suivant leur scénario dans $P1$. Les clients restants seront insérés selon leur scénario dans $P2$ dans une position qui permet d'avoir les mêmes voisins que dans $P2$ (maximiser le nombre de successeurs et prédécesseurs commun du client i dans $Child$ et dans $P2$). Par exemple, le client 8 a comme successeur les clients 6, 2 , et 1 dans le jour 2 du parent $P2$, il est donc inséré avant le client 6 dans $Child$.

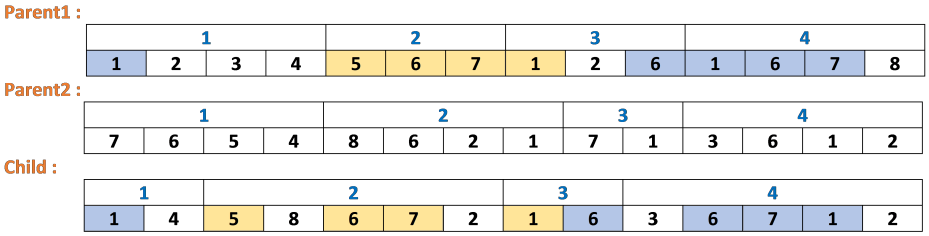


FIGURE 6.5 – Exemple de croisement $Affect$

Croisement *RandomInsert*

Ce croisement est une variante des deux croisements précédents, dans lequel l'insertion des clients doit respecter les scénarios des parents, mais au lieu d'insérer les clients issus de $P2$, dans $Child$ en fin de liste, ils seront insérés de manière aléatoire dans la séquence du jour sélectionné.

Comme dans $Affect$, tous les clients entre a et b sont insérés dans l'ordre de P1 dans le chromosome $Child$ dans chaque jour de leur scénario. Ensuite, les clients restants sont insérés selon leur scénario dans $P2$. La différence avec $Affect$ réside dans le fait que les clients hérités de $P2$ seront insérés dans des positions aléatoires pour chaque jour de leur scénario.

Dans l'exemple de la Figure 6.6, les clients de $P1$ entre les deux points de croisement sont insérés dans $Child$. Puis le reste des clients sont insérés selon le scénario dans $P2$ à une position aléatoire. Par exemple, le client 8 est inséré à une position aléatoire du deuxième jour contrairement à $Affect$ où il était visité après les clients de $P1$.

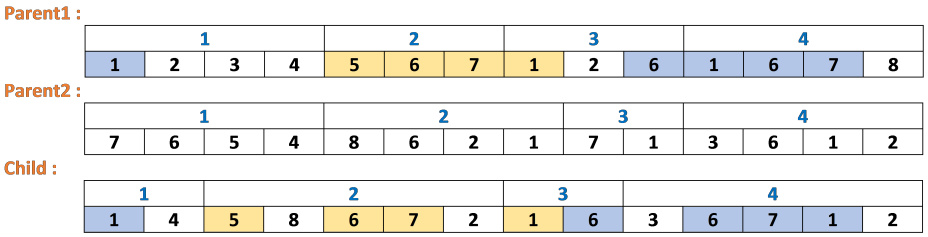


FIGURE 6.6 – Exemple de croisement $AffectRandom$

Croisement *ELOX*

Le croisement $ELOX$ (Extended LOX), inspiré des travaux dans [57] est une extension de LOX à des chromosomes multi-périodes et dans lesquelles les tâches apparaissent plus d'une fois. Ce croisement respecte au mieux l'ordre et l'affectation des deux parents.

Les clients entre les deux points de croisement sont insérés dans le chromosome $Child$ à la même position. Ces clients nécessitent potentiellement d'autres visites pour respecter leurs fréquences. Le chromosome $P2$ est alors scanné de gauche à droite et pour chaque client trois possibilités se présentent :

1. Le client est déjà visité autant de fois que nécessaire selon son scénario dans $P1$, dans ce cas ce client ne sera pas copier de nouveau dans $Child$
2. Le client n'est pas visité autant de fois que sa fréquence dans $Child$ et peut être visité selon son scénario dans $P2$. Il est alors inséré selon son scénario dans $P2$

3. Le client n'est pas visité autant de fois que sa fréquence dans $Child$ et ne peut pas être visité selon son scénario dans $P2$. Parmi les scénarios possibles, un scénario est choisi aléatoirement et le client est inséré selon celui-ci.

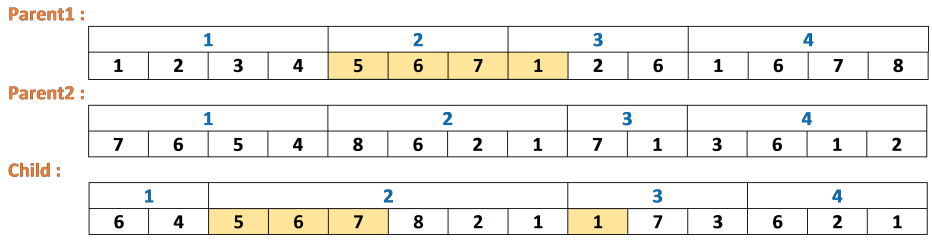


FIGURE 6.7 – Exemple de croisement $ELOX$

L'exemple dans la Figure 6.7 explicite le croisement $ELOX$. En premier, nous insérons dans $Child$ les clients $(5, 6, 7)$ au deuxième jour, et le client 1 au troisième jour. Ensuite, $P2$ est parcouru de gauche à droite, chaque client est examiné comme suit :

— Le parcours de $P2$ retourne le client 7 en premier. Ce client étant déjà visité le deuxième jour, son scénario dans $P2$ ($\{1, 3\}$) ne pourra pas être respecté. Il lui reste un scénario ($\{2, 4\}$) faisable, il sera donc visité le jour 4.
— Le deuxième client est le client 6. Le scénario du client 6 dans $P2$ est ($\{1, 2, 4\}$), il reste donc à le visiter le premier et quatrième jour.
— Le client 5 est déjà visité autant de fois que sa fréquence, il ne sera pas copié de nouveau.
— Le client 4 n'est pas encore visité dans $Child$, il sera donc visité suivant son scénario dans $P2$.
— etc.

6.3.5 Mutation

Après le croisement, un opérateur de mutation est appliqué aux enfants avec une probabilité très faible, nommé taux de mutation ou Pm. Chaque enfant de notre GA est converti en une solution GPEVRP afin de subir une mutation. Nous proposons deux mouvements simples ($Move$ et $MoveNclient$) et une mutation de recherche locale basée sur une recherche à grand voisinage (LNS).

client	Fréquence(client)	Combinaison(client)
1	3	{1,3,4}, {2,3,4}
2	2	{1,3}, {2,4}
3	1	{1}, {4}
4	1	{1}
5	1	{1}, {2}
6	3	{2,3,4}, {1,2,4}, {1,2,3}
7	2	{2,4}, {1,3}
8	1	{2}, {4}

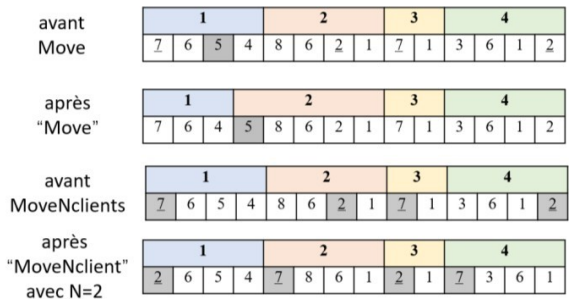


FIGURE 6.8 – Exemple des deux méthodes de mutation

— $Move$: La première méthode est $Move$. Nous sélectionnons un client i, et choisissons aléatoirement un de ses scénarios au hasard $sc(i)$. Ensuite, le client i est supprimé dans chaque jour de son scénario. Puis, il est réinséré suivant le scénario $sc(i)$.

— $MoveNclients$: répéter le mouvement $Move$ N fois (N généré aléatoirement). La Figure 6.8 présente un exemple des deux mutations ci-dessus.
— Mutation de recherche locale basée sur LNS (Large neighbourhood search) : pour être efficace, le GA doit être hybridé avec une recherche locale, donnant un GA hybride ou un algorithme mémétique (MA). Nous avons choisi d'utiliser une recherche locale à base de LNS car cette méthode a prouvé son efficacité sur notre problème (voir les chapitres 4 et 5). Cette procédure utilise un nombre limité de clients à supprimer afin de concentrer l'intensification dans un voisinage proche de la solution. Une étape de tests préliminaire nous a permis de fixer cette valeur à 3. Le LNS est appliqué à la solution GPEVRP. D'abord trois clients sont supprimés en utilisant un opérateur de destruction aléatoire, puis une procédure d'insertion est utilisée pour insérer les clients supprimés à la meilleure position possible. L'Algorithme 26 résume les étapes de cette mutation.

Algorithm 26 *LNS*

Input : une solution S
Output : S modifié
$S_{best} = S$ Meilleure solution trouvée
while Stoping Criteria **do**
    $S = S_{best}$;
    Destruction : Supprimer trois clients aléatoirement dans S
    Réinsérer les clients supprimés dans S selon principe $BestImprovement$
    if $Fitness(S) < Fitness(S_{best})$ **then**
        $S_{best} = S$;
    end if
end while

6.3.6 Remplacement

La "stratégie de l'élitisme" est utilisée pour déterminer les solutions à éliminer de la population. La population des nouveaux enfants est ajoutée à la population des parents. Les deux populations sont regroupées, triées et seul un nombre fixe des meilleures solutions sera retenu pour la prochaine génération. L'Algorithme 27 résume la stratégie de remplacement utilisée.

Algorithm 27 *Remplacement*

Input : Pop,Population de l'itération précédente
        Enf Ensemble des enfants
Output : $NewPop$ Population finale
$n := Pop.size()$
$NewPop$ vecteur de taille n
$Pop := Pop \cup Enf$
Trier Pop dans l'ordre décroissant du fitness
for $i = 1$ To n **do**
    $NewPop[i] = Pop[i]$
end for

	$Option_1$	$Option_2$	$Option_3$	$Option_4$	$Option_5$
C_1	11538,30	12953,26	13062,39	12266,04	11523,80
C_2	13226,51	13910,94	14115,39	13746,72	13398,76
C_3	12645,57	13518,98	13691,45	13074,66	12671,02

TABLE 6.1 – Comparaison du $Cost$ sur les 15 configurations de $SplitGPEVRP$

	$Option_1$	$Option_2$	$Option_3$	$Option_4$	$Option_5$
C_1	16,62	23,53	23,38	19,83	16,92
C_2	23,63	27,13	27,23	25,33	24,33
C_3	21,85	30,80	26,54	23,86	22,17

TABLE 6.2 – Comparaison de $NbTr$ sur les 15 configurations de $SplitGPEVRP$

6.4 Expérimentation numérique

Le GA a été implémenté en C++ sous Clion et a été exécuté avec un Intel Core i7, 2.50GHz avec 8Go de RAM.

Un sous-ensemble d'instances du $PEVRP$, et du $m-GPEVRP$, que nous avons présentées dans les chapitres précédents, a été utilisé pour évaluer notre GA.

Nous avons développé des composants du GA spécifiques à notre problème $GPEVRP$; ce qui ne permet pas la comparaison avec les métaheuristiques à voisinages proposées pour le $PEVRP$ et le $m-PEVRP$, qui elles gèrent des solutions partielles (à cause de la limitation de la flotte de véhicules) de manière plus intuitive grâce au principe même des méthodes à base de LNS.

Dans ce qui suit, nous présenterons l'évaluation de la méthode $SplitGPEVRP$, le paramétrage de notre GA, puis l'évaluation des opérateurs de croisement et de mutation sur les instances de $PEVRP$, et enfin nous finirons par une évaluation numérique du GA sur un groupe d'instances du $m-GPEVRP$.

6.4.1 Évaluation de méthode $SplitGPEVRP$

Dans cette section, nous évaluons 15 variantes de $SplitGPEVRP$, chaque variante utilise une $Option_i$ donnée ($i \in \{1,..,5\}$) pour une stratégie d'insertion de chargeur, et un coût C_j ($j \in \{1,2,3\}$), pour un critère d'insertion. Pour comparer les résultats des 15 variantes, nous utilisons les trois indicateurs suivants : $NbTr$ pour le nombre de tournées maximum sur l'horizon, Cpu pour le temps de calcul, et $Cost$ pour le coût de solution.

Nous avons effectué les tests sur 50 instances choisis aléatoirement parmi les instances du problème.

Les résultats des Tableaux 6.1, 6.2, et 6.3 , et les Figures 6.9, 6.10, et 6.11, montrent que l'utilisation de l'$Option_1$ permet d'avoir la réparation avec le moindre coût. En effet, l'$Option_1$ teste toutes les positions d'insertion. Ces résultats sont assez proches des résultats de l'$Option_5$

	$Option_1$	$Option_2$	$Option_3$	$Option_4$	$Option_5$
C_1	0,57	0,05	0,11	0,22	0,29
C_2	0,56	0,06	0,11	0,21	0,28
C_3	0,53	0,05	0,09	0,17	0,31

TABLE 6.3 – Comparaison du Cpu en secondes sur les 15 configurations de $SplitGPEVRP$

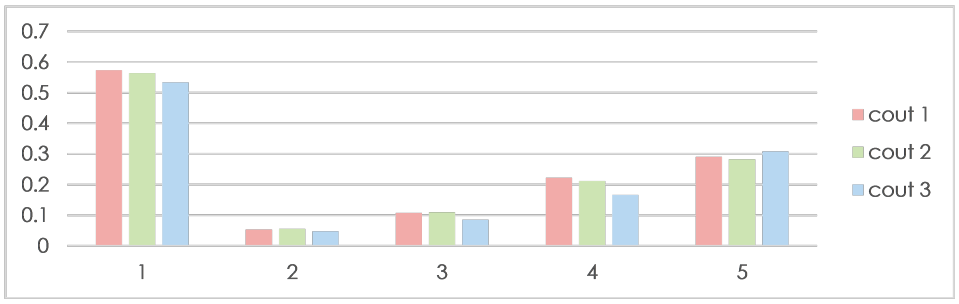


FIGURE 6.9 – Évaluation du *Cpu* sur les 15 configuration du SplitGPEVRP

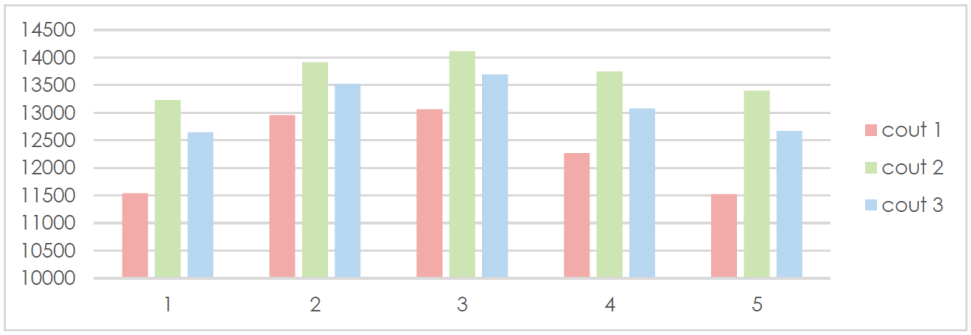


FIGURE 6.10 – Évaluation du *Cost* sur les 15 configuration du SplitGPEVRP

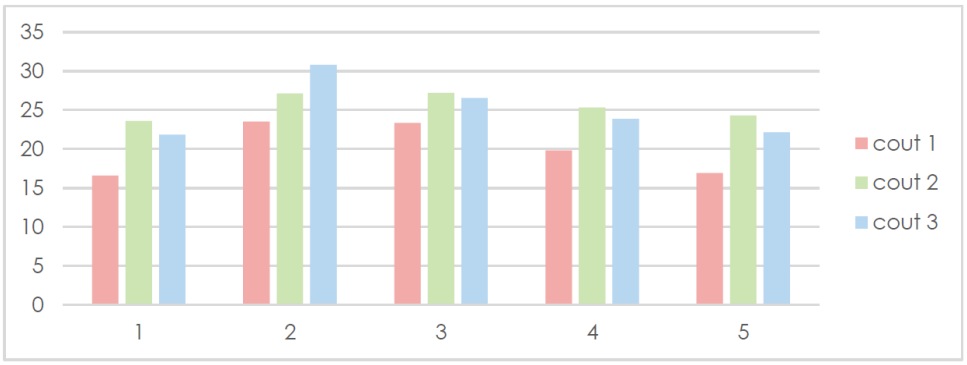


FIGURE 6.11 – Évaluation du $NbTr$ sur les 15 configuration du SplitGPEVRP

qui ne teste l'insertion qu'à partir d'un seuil de 50%. Par contre, l'$Option_5$ est beaucoup plus rapide. Pour la suite des expérimentations, nous avons alors retenu l' l'$Option_5$ et le coût l' C_1.

6.4.2 Paramétrage de l'algorithme génétique

Dans cette section, nous nous intéressons au paramétrage de l'algorithme génétique. Nous avons réalisé plusieurs simulations de configurations afin de déterminer la meilleure.

L'objectif est de fixer les paramètres suivants : itération It, taille de population $PopSize$, le nombre de parents à sélectionner $NbSel$, la méthode de mutation Mut et la probabilité de mutation $PrMut$. Pour cela, nous avons testé l'algorithme génétique sur les instances à 100 clients, 5 jours, $f(i) = 1$, et flotte illimitée. On notera (It, $PopSize$, $NbSel$, Mut, $PrMut$) une configuration donnée de l'algorithme, le tableau 6.4 donne l'ensemble des simulations réalisées.

Itération(It)	[1000,10000]
Taille de population ($PopSize$)	20,30,40,50,60,70
Nombre de sélection($NbSel$)	2, 8, 15
Nombre d'enfant (Mut)	$NbSel - 1$
Méthode de mutation (Mut)	$Move$, $MoveNclient$, $MoveRandom$
Probabilité de mutation ($PrMut$)	0.02,0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7

TABLE 6.4 – Paramétrage de l'algorithme génétique

Sur ces tests exhaustifs, nous avons fixé le croisement à « ELOX ».

Notons, que pour la mutation LNS, nous nous sommes inspirés d'autres travaux hybridant la méthode LNS avec des métaheuristiques ([94], [92]) pour fixer le nombre de clients à éjecter/réinsérer à 3, le nombre d'itération $ItLNS$ à 30, et $PrMut$ à 0,02.

Nombre d'itération It

La Figure 6.12 présente les résultats en fonction du nombre d'itérations. Nous constatons qu'il n'y a pas beaucoup d'amélioration entre l'itération 5000 et l'itération 10000. Pour avoir un compromis entre le temps de calcul et le coût de la solution, nous avons retenu ($It = 5000$)

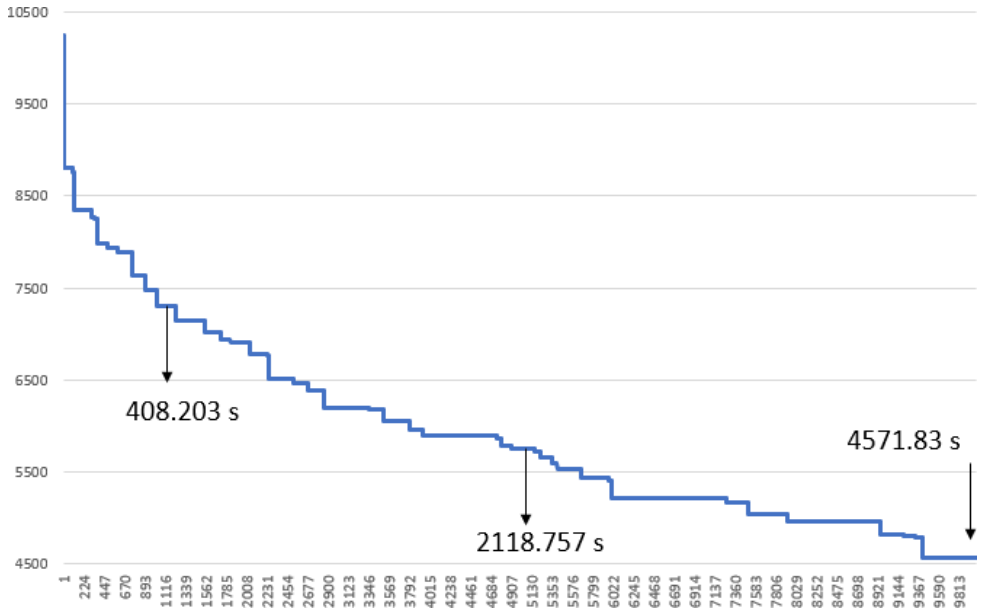


FIGURE 6.12 – Convergence des solutions du GA par It

Taille de la population $PopSize$

Pour ces tests, nous avons fixé $NbSel = 2$, $Mut = Move$, et $PrMut = 0,1$. La variation du coût moyen des solutions et des temps de calcul sont présentés dans la Figure 6.13. La valeur $PopSize = 60$ semble permettre un bon compromis entre efficacité et temps de calcul.

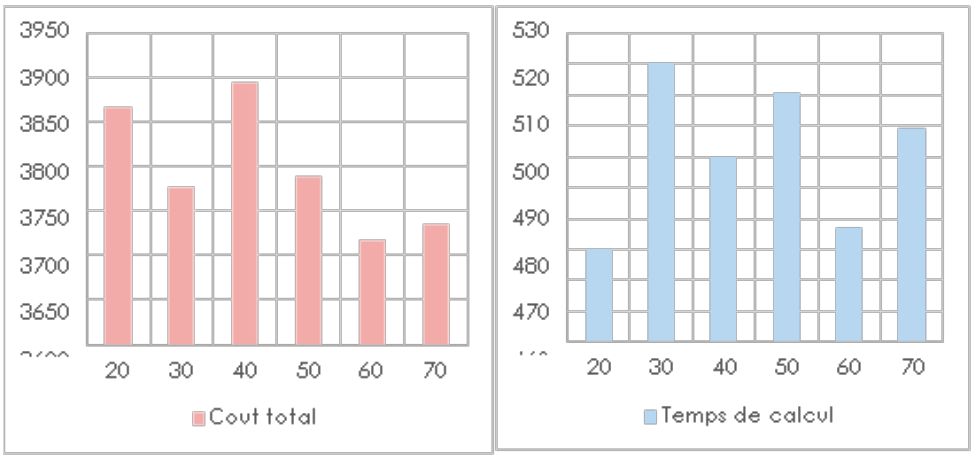


FIGURE 6.13 – Coût moyen et temps de calcul par taille de population

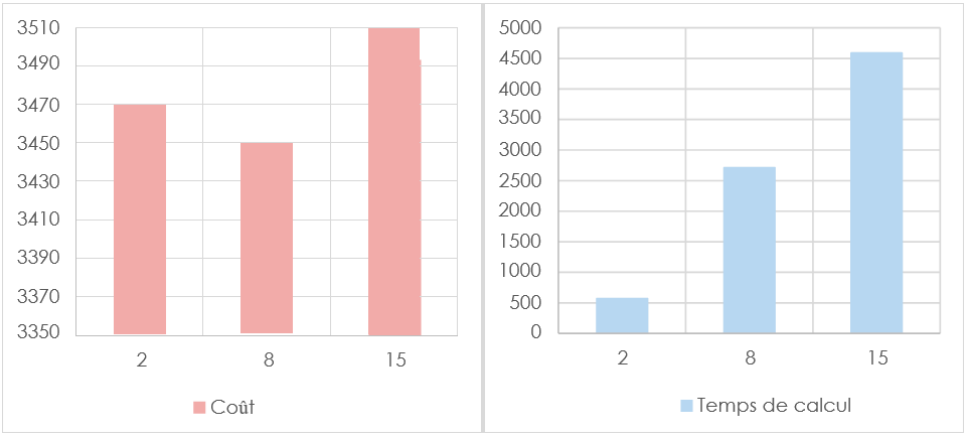


FIGURE 6.14 – Coût moyen et temps d'exécution par $NbSel$

Nombre de parents à sélectionner $NbSel$

Le paramètre $NbSel$ permet de définir le nombre d'individus à sélectionner dans la population, qui seront concernés par le croisement.

La Figure 6.14 présente la variation du coût moyen des solutions et du temps de calcul pour différentes valeurs de $NbSel$. En moyenne, les meilleures solutions sont obtenues quand $NbSel = 8$ mais cela au détriment d'un temps d'exécution élevé. Les solutions obtenues quand $NbSel = 2$ présentent un bon compromis entre temps d'exécution et coût de la solution. Nous avons donc fixé $NbSel$ à 2.

Probabilité $PrMut$ **et méthode de mutation** Mut

Notre objectif ici est d'évaluer $PrMut$ pour les trois mutations $Move$, $MoveNclients$ et $MoveRandom$. Les deux premières mutations ont été présentées précédemment. $MoveRandom$ choisit aléatoirement à chaque itération une des deux méthodes $Move$ et $MoveNclients$.

Les résultats d'étude de la variation du coût moyen en fonction de $PrMut$ pour la mutation $Move$ présentés dans la Figure Figure 6.15, nous ont permis de fixer $PrMut = 0,3$. Concernant la mutation $MoveNclients$, nous avons varié $PrMut$ et le nombre de clients à perturber N dans

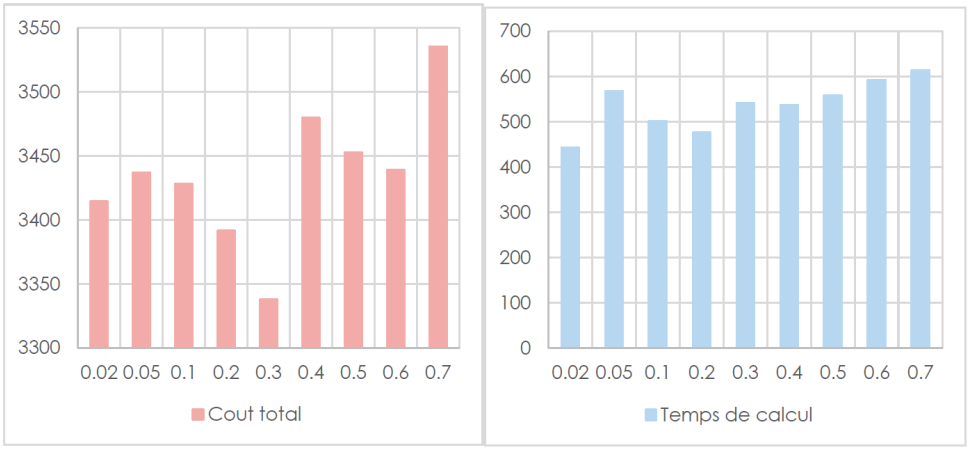


FIGURE 6.15 – Évaluation de la mutation *Move* : Coût total moyen, et temps de calcul en fonction de *Pmut*

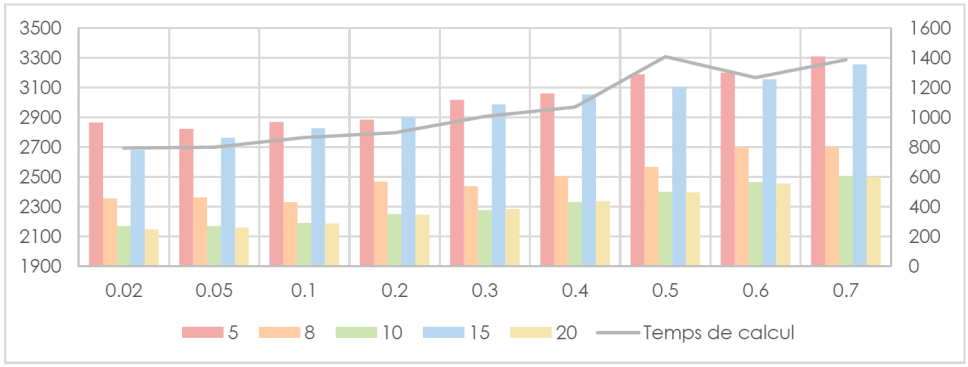


FIGURE 6.16 – Coût total moyen et temps total moyen en fonction de *Pmut* et *N*

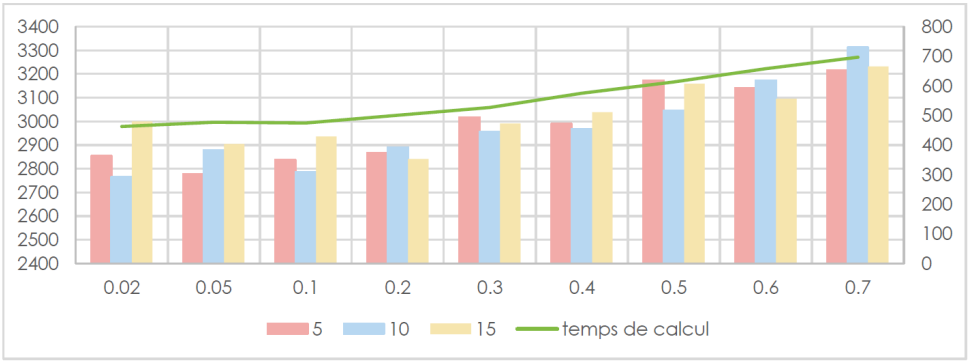


FIGURE 6.17 – Évaluation de *MovRandom* : Coût total moyen et temps total moyen en fonction de *Pmut* et *N*

$\{5, 8, 10, 15, 20\}$. La Figure 6.16 synthétise les résultats moyens obtenus et montre que $N = 10$ et $Pmut = 10$ constitue un bon compromis entre efficacité et temps de calcul. Figure 6.17 regroupe les résultats d'évaluation de la mutation $MoveRandom$ en fonction de N et $Pmut$ et permet de confirmer le même paramétrage fixé pour $MoveNclients$, c'est-dire $N = 10$ et $Pmut = 0,02$.

Évaluation des paires de mutation et de croisement

Nous avons effectué des tests pour évaluer l'influence du choix de la paire (mutation, croisement) sur les résultats de l'algorithme génétique. Pour chaque mutation fixée, nous avons varié les croisements, et nous avons exécuté chaque paire (mutation, croisement) 3 fois. Pour évaluer chaque paire, nous avons examiné 5 métriques :

- $Gap\_initial = (cost_i - cost_f)/cost_i$, avec $cost_i$ coût de la meilleure solution de la population initiale, et $cost_f$ coût de la meilleure solution à la fin de l'algorithme.
- $Gap\_best = (cost_f - cost_{best})/cost_f$, avec $cost_{best}$ coût de la meilleure solution trouvée sur l'ensemble des tests réalisés
- $Iteration\_best$: l'itération où la meilleure solution a été trouvée
- $Time$: temps d'exécution
- Nbr_best : nombre de fois où la paire a trouvé la meilleure solution connue

Mutation $Move$ Ici, nous avons fixé la mutation à $Move$ et nous varions la méthode de croisement.

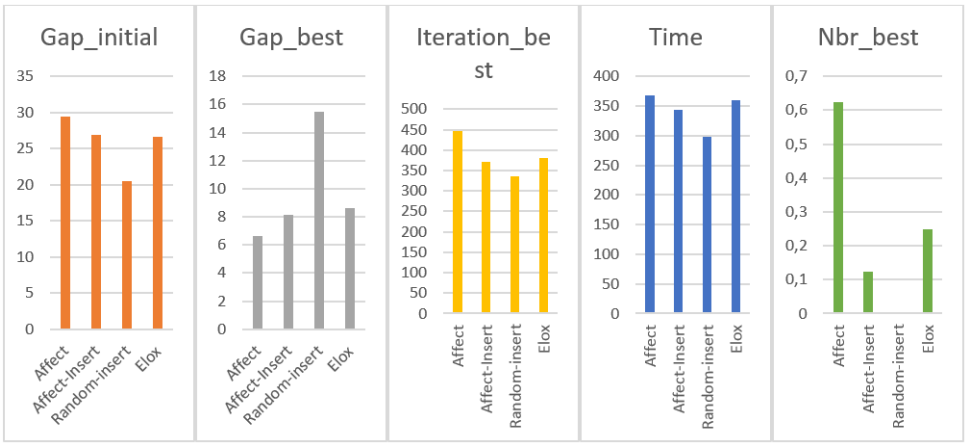


FIGURE 6.18 – Variation des croisements avec $Move$

On remarque dans la Figure 6.18, que la méthode $Affect$ est celle qui a le meilleur $gap\_initial$ et le meilleur Nbr_best, ce qui signifie qu'elle fournit la meilleure solution le plus souvent. Le $gap\_initial$ approche 30%. La meilleure solution avec ce croisement est obtenue à un nombre d'itérations plus élevé que les autres méthodes, ce qui montre que la convergence est un peu plus lente que les autres paires. Le croisement $RandomInsert$ obtient les mauvaises performances, cela est certainement dû à son caractère aléatoire, qui combiné avec une mutation de type «perturbation aléatoire» ne permet pas à ce croisement d'être compétitif. Le temps de calcul reste relativement raisonnable, 350 secondes en moyenne.

Mutation $MoveNclient$ **:** Le graphique de la Figure 6.19 montre que la meilleure méthode de croisement à coupler avec $MoveNclient$ est $AffectInsert$. Ce couple trouve la meilleure solution

sur la moitié des instances. La convergence est relativement rapide pour cette paire $MoveNclient$ et $AffectInsert$ puisque la meilleure solution est atteinte à l'itération 450. Le temps de calcul reste aussi raisonnable. Comme dans les résultats avec la mutation $Move$, le croisement $Affect$ obtient ici des résultats tout à fait intéressants et comparables à $AffectInsert$.

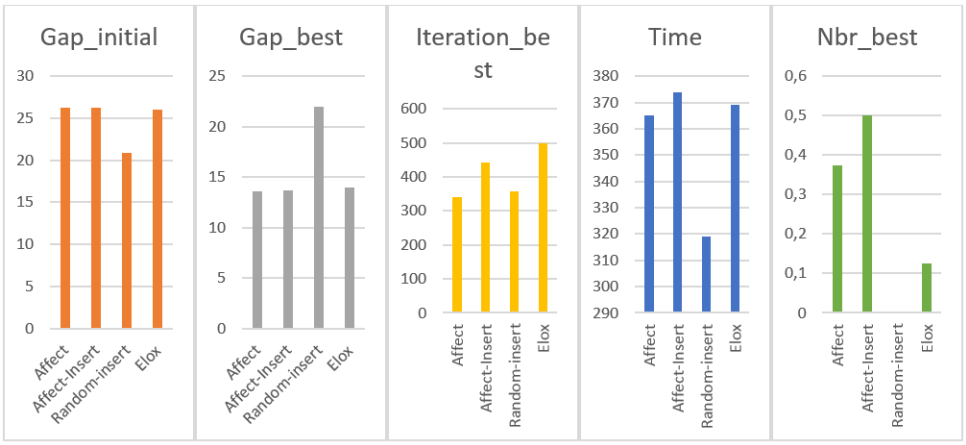


FIGURE 6.19 – Variation des croisements avec $MoveNclient$

Mutation LNS_3 : Nous avons fixé It à 300 parce que la mutation LNS_3 est de type recherche locale et consommatrice en termes de temps de calcul.

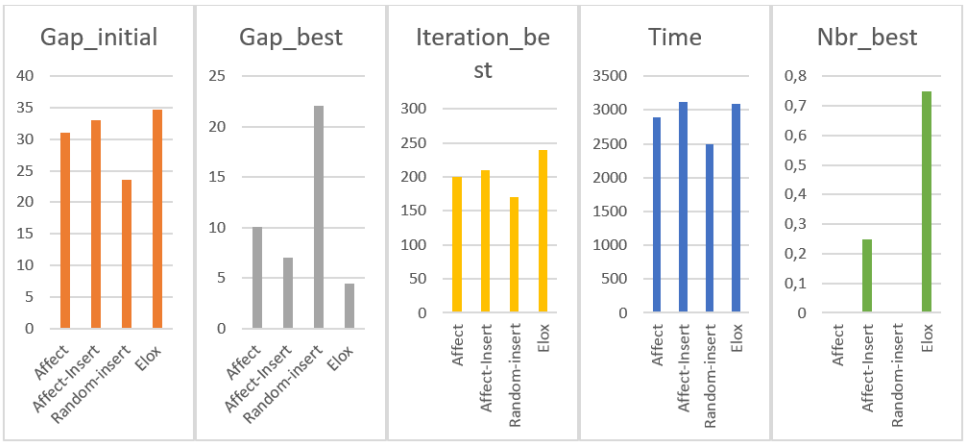


FIGURE 6.20 – Variation des croisement avec LNS_3

La Figure 6.20 montre que le croisement $ELOX$ a atteint la meilleure solution dans plus de 70% des cas, il a également le $gap\_initial$ le plus élevé et le Gap_best le plus bas. Le $gap\_initial$ approche 35%, et il est meilleur comparé aux résultats obtenus avec les autres paires. La même remarque peut être formulée pour le gap_best qui atteint les meilleures performances avec le couple ($ELOX$, LNS_3). Le temps de calcul est assez élevé, il atteint 3000 secondes en moyenne, cependant la convergence est atteinte à $iter = 200$ en moyenne.

Mutation $Moverandom$: La mutation $Moverandom$ choisit entre les mutations Mov, $MovNclient$, et LNS_3.

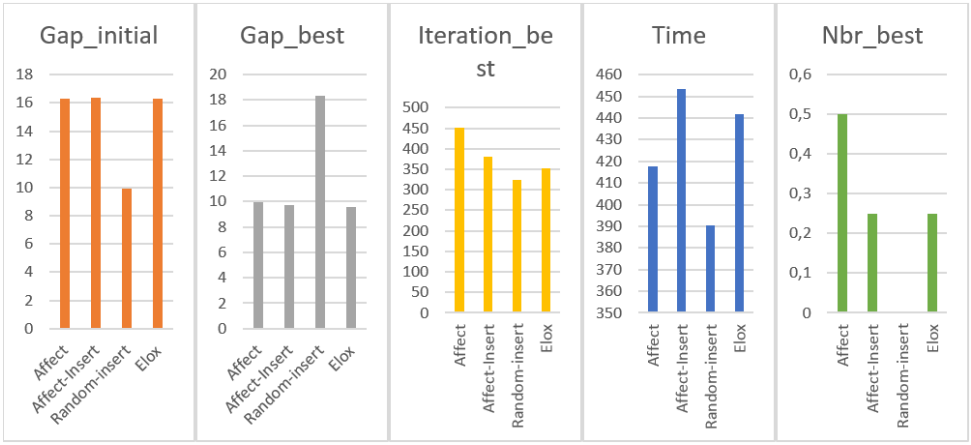


FIGURE 6.21 – Variation des croisement avec *Moverandom*

	$Iter_best(s)$	Gap_i
Ins0	2678,71	30,96
Ins1	3150,30	29,94
Ins2	2546,16	31,05
Ins3	2689,59	29,39
Ins4	2881,43	25,73
Ins5	2382,31	26,86
Ins6	1549,47	12,84
Ins7	3369,72	24,95
Ins8	2561,03	22,41
Ins9	3065,22	26,23
Moyenne	**2687,40**	**26,04**

TABLE 6.5 – Évaluation du GA sur un ensemble d'instances de m-GPEVRP avec flotte illimité

La Figure 6.21, montre que les performances globales de cette mutation restent assez faibles. En effet le *gap_initial* ne dépasse pas 18% sur tous les croisements.

En conclusion, la méthode hybride combinant le GA avec un croisement $ELOX$ et une recherche locale de type LNS permet d'obtenir les meilleures performances. Nous avons alors testé dans ce qui suit cette version sur les instances de notre étude $m-GPEVRP$

6.4.3 Résultats du GA sur les instances du $m-GPEVRP$

Dans cette section, nous évaluons le GA avec la paire $(ELOX, LNS)$ et avec les paramétrages fixés ci-dessus sur 10 instances du m-GPEVRP à 100 clients, 5 jours, et flotte illimitée (voir Tableau 6.5). Le critère d'arrêt est fixé à un temps de calcul de 1h et chaque instance est testée 3 fois. Les résultats montrent que le Gap_i moyen est de $26,04\%$ et le temps moyen de calcul est de 2687,40 secondes. Ces résultats confirment que notre GA est assez générique et pourrait s'appliquer aux cas mono-visite et multi-visites sur l'horizon.

6.5 Conclusion

Dans ce chapitre, nous avons présenté une nouvelle extension temporelle du PEVRP, nommée GPEVRP qui est une généralisation du m-GPEVRP étudié dans le chapitre précédent. Nous avons présenté un algorithme génétique GA qui permet de traiter simultanément des décisions du niveau opérationnel (définition des tournées journalières) et du niveau tactique (affectation des clients aux jours) grâce à un codage du chromosome en un tour géant par jour, et sans délimiteurs de tournées dans chaque jour. Nous avons proposé 4 opérateurs de croisement et 4 opérateurs de mutation, que nous avons évalués sur les instances proposées dans les chapitres précédents en relaxant la limitation du nombre de véhicules. Les tests réalisés montrent que le GA permet d'obtenir les meilleurs résultats avec le croisement $ELOX$ et la mutation de type recherche locale LNS. Ces résultats confirment l'intérêt d'hybrider le GA avec LNS et montrent la validité du GA sur les cas mono-visite et multi-visites. Pour la suite, nous envisageons de tester notre GA dans le cas particulier $f(i) = 1$ et $np = 1$, pour pouvoir comparer les résultats sur les instances EVRP de la littérature.

Chapitre 7

Conclusion et perspectives

7.1 Conclusion

Dans cette thèse, nous avons étudié le problème de planification des tournées de véhicules électriques. En plus de la construction des tournées, notre problématique concerne l'aspect planification des visites des clients sur un horizon et la prise en compte de l'écosystème du véhicule électrique. Nous avons commencé par dresser un état de l'art sur les problématiques proches de la nôtre. Cet état de l'art nous a permis de constater que les travaux sur les problèmes de tournées de véhicules électriques connaissent une croissance importante ces dernières années, mais que le problème de planification de ces tournées n'a jamais été étudié auparavant alors que l'intérêt pratique est certain.

Dans le troisième chapitre, nous avons présenté un nouveau modèle, nommé $PEVRP$, traitant de la planification des tournées de véhicules électriques dans le cadre d'une visite unique à chaque client sur un horizon de temps de plusieurs jours, et utilisant une flotte de véhicules limitée. Nous avons défini puis modélisé cette problématique. Ensuite, nous avons proposé trois heuristiques de résolution. Les expérimentations nous ont permis d'évaluer la qualité de nos heuristiques sur des instances de complexités différentes. L'heuristique BIH, basée sur l'insertion est la plus efficace sur les instances peu contraintes, alors que les heuristiques $CLH1$ et $CLH2$, basées sur la clusterisation, sont plus efficaces sur les instances plus difficiles.

Dans le quatrième chapitre, nous avons proposé deux métaheuristiques à base de voisinage large pour résoudre le $PEVRP$. Une métaheuristique de recherche à grand voisinage (LNS) et la deuxième est une métaheuristique de recherche adaptative à voisinage large ($ALNS$). Nous avons pu constater, grâce aux expérimentations, que la métaheuristique LNS est plus efficace pour résoudre le $PEVRP$.

Dans le cinquième chapitre, nous avons considéré une généralisation du $PEVRP$, nommée $m-GPEVRP$ où les clients nécessitent plusieurs visites sur l'horizon, et où la flotte est limitée. Nous avons commencé par définir et modéliser cette problématique. Quatre heuristiques de construction et trois métaheuristiques sont proposées et détaillées.

Les quatre heuristiques proposées commencent par fixer le scénario de chacun des clients puis construisent les tournées. La première heuristique ($Hconst1$) construit les tournées en insérant les clients dans l'ordre donné par une liste. Les trois autres heuristiques ($Hconst2$, $Hconst3$, $Hconst4$), comparent l'insertion des clients suivant une métrique ($Metrique1$, $Metrique2$, $Metrique3$) et insère le client à moindre coût à chaque itération. Parmi les heuristiques, c'est l'heuristique $Hconst1$ qui trouve les meilleures solutions.

Les métaheuristiques proposées sont aussi basées sur une résolution par voisinage large, il

s'agit de $GLNS$, LNS et $ALNS$. La métaheuristique Multi-Start Guided LNS ($GLNS$) alterne entre une phase d'intensification et une phase de perturbation. Initialement, les scénarios des clients sont fixés puis une phase d'intensification basé sur un LNS est lancée pour trouver la meilleure solution avec ces scénarios. La phase de perturbation permet de changer les scénarios d'un sous ensemble de clients jugés difficiles à insérer dans la phase d'intensification précédente. De même que sur la précédente problématique, la métaheuristique LNS présente les meilleurs résultats en termes des meilleures solutions trouvées.

Dans le sixième chapitre, nous avons étudié une variante de notre problématique où le nombre de véhicules n'est plus limité. Cette variante, nommée $GPEVRP$, généralise le $m-GPEVRP$ en relaxant la contrainte sur la limitation de la flotte. Nous avons proposé un algorithme génétique permettant de traiter simultanément le niveau tactique (affectation des clients aux jours) et le niveau opérationnel (séquence des clients par jour). Le codage du chromosome utilise un tour géant par jour sans délimiteurs de tournées, et permet l'évaluation de la solution grâce à $SplitGPEVRP$, une adaptation de la méthode $Split$. Notons que $Split$ a été largement utilisée pour évaluer des solutions de tournées de véhicules codées sous forme de tours géants. Quatre opérateurs de croisement et quatre opérateurs de mutation, dont une mutation à base de recherche locale de type LNS, ont été proposés et évalués. Les résultats de ce chapitre montrent encore l'efficacité du LNS car c'est l'algorithme génétique hybride qui utilise une recherche locale de type LNS qui donne les meilleures performances. Nous pourrons aussi confirmer la puissance de notre GA puisqu'il est adapté au cas mono-visite et multi-visites.

Toutes les expérimentations et les évaluations réalisées dans les chapitres ci-dessus ont été réalisées sur de nouvelles instances que nous avons générées en généralisant les instances connues sur le $EVRP$.

7.2 Perspectives

Les problématiques traitées dans cette thèse sont assez originales et d'actualité. De nombreuses perspectives sont donc envisageables :

- Considération d'une flotte hétérogène de véhicules électriques. En effet, il existe différents types de véhicules en termes d'autonomie, de capacité d'emport et de vitesse de recharge. La prise en compte de cette contrainte permettrait de s'approcher plus des problèmes réels.
- Prise en compte d'une fonction de consommation d'énergie plus proche de la réalité et donc non linéaire. La consommation d'énergie des véhicules électriques est un sujet qui a inspiré plusieurs travaux dans la littérature, mais elle a été étudiée le plus souvent indépendamment des problèmes de tournées de véhicules. Il existe donc peu de travaux couplant une consommation d'énergie non linéaire à une problématique de tournée de véhicules électriques. Même s'il est difficile de s'approcher de la consommation réelle des véhicules électriques en exploitant des fonctions non linéaires, l'utilisation de ces fonctions est un bon compromis entre la complexité de l'étude des problèmes d'optimisation et l'intérêt pratique des solutions produites par ces problèmes et leurs impacts sur les problèmes réels.
- Amélioration des heuristiques de clustering ($CLH1$ et $CLH2$). En effet, nous avons noté l'efficacité de ces heuristiques pour trouver des solutions complètes sur des instances "difficiles", mais leur performance reste à améliorer pour obtenir des solutions complètes moins coûteuses.
- Comparaison des approches à base de LNS avec les algorithmes génétiques sur les ins-

tances de $PEVRP$ et $GPEVRP$ avec flotte limitée. En effet, cela nécessiterait l'adaptation du codage, des croisements,des mutations pour gérer des solutions incomplètes au cours de la recherche, mais aussi adapter $SplitGPEVRP$ pour obtenir un nombre fixé de tournées. L'adaptation du $SplitGPEVRP$ reviendrait à résoudre un problème du plus court chemin contraint, qui est un problème difficile.

— Comparaison du GA avec les approches de la littérature sur le cas particulier $EVRP$. La prise en compte d'une flotte illimitée permettrait de réaliser ces comparaisons d'une manière plus simple que pour les approches à base de LNS qui intègrent la spécificité d'une flotte limitée dans la recherche des solutions.

— Amélioration du GA en intégrant des techniques adaptatives, permettant à l'algorithme de sélectionner de manière dynamique ses opérateurs de croisement et de mutation.

Ces différentes perspectives ci-dessus permettraient d'avancer les recherches sur les modèles de planification des tournées de véhicules électriques et de proposer des solutions aux entreprises désirant utiliser des véhicules électriques.

Bibliographie

[1] Yogesh Agarwal, Kamlesh Mathur et Harvey M. Salkin. « A set-partitioning-based exact algorithm for the vehicle routing problem ». In : *Networks* 19.7 (1989), p. 731-749. doi : 10.1002/net.3230190702. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230190702. url : https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230190702.

[2] Juan J. Alcaraz, Luis Caballero-Arnaldos et Javier Vales-Alonso. « Rich vehicle routing problem with last-mile outsourcing decisions ». In : *Transportation Research Part E : Logistics and Transportation Review* 129 (2019), p. 263-286. issn : 1366-5545. doi : https://doi.org/10.1016/j.tre.2019.08.004. url : http://www.sciencedirect.com/science/article/pii/S1366554519302339.

[3] Ammar OULAMARA. *Ordonnancement et routage des véhicules électriques pour la logistique urbaine : défis et challenges – EVERS*. https://anr.fr/Projet-ANR-15-CE22-0017.

[4] Marwa Amous, Said Toumi, Bassem Jarboui et Mansour Eddaly. « A variable neighborhood search algorithm for the capacitated vehicle routing problem ». In : *Electronic Notes in Discrete Mathematics* 58 (2017). 4th International Conference on Variable Neighborhood Search, p. 231-238. issn : 1571-0653. doi : https://doi.org/10.1016/j.endm.2017.03.030. url : http://www.sciencedirect.com/science/article/pii/S1571065317300665.

[5] G. Andreatta, M. Casula, C. De Francesco et L. De Giovanni. « A branch-and-price based heuristic for the stochastic vehicle routing problem with hard time windows ». In : *Electronic Notes in Discrete Mathematics* 52 (2016). INOC 2015 – 7th International Network Optimization Conference, p. 325-332. issn : 1571-0653. doi : https://doi.org/10.1016/j.endm.2016.03.043. url : http://www.sciencedirect.com/science/article/pii/S1571065316300488.

[6] Yulia Anoshkina et Frank Meisel. « Technician teaming and routing with service-, cost- and fairness-objectives ». In : *Computers& Industrial Engineering* 135 (2019), p. 868-880. issn : 0360-8352. doi : https://doi.org/10.1016/j.cie.2019.05.016. url : http://www.sciencedirect.com/science/article/pii/S0360835219302839.

[7] ASSEMBLÉE NATIONALE. *Projet de loi d'orientation des mobilités*. http://www.assemblee-nationale.fr/15/textes/1974.aspD_Article_26_A.

[8] Roberto Baldacci, Enrico Bartolini, Aristide Mingozzi et Andrea Valletta. « An Exact Algorithm for the Period Routing Problem ». In : *Operations Research* 59.1 (2011), p. 228-241. doi : 10.1287/opre.1100.0875. eprint : https://doi.org/10.1287/opre.1100.0875. url : https://doi.org/10.1287/opre.1100.0875.

[9] Roberto Baldacci, Nicos Christofides et Aristide Mingozzi. « An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts ». In : *Mathematical Programming* 115.2 (oct. 2008), p. 351-385. issn : 1436-4646. doi : 10.1007/s10107-007-0178-5. url : https://doi.org/10.1007/s10107-007-0178-5.

[10] E. J. Beltrami et L. D. Bodin. « Networks and vehicle routing for municipal waste collection ». In : *Networks* 4.1 (1974), p. 65-94. doi : 10.1002/net.3230040106. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230040106. url : https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230040106.

[11] Sinaide Nunes Bezerra, Sérgio Ricardo de Souza et Marcone Jamilson Freitas Souza. « A GVNS Algorithm for Solving the Multi-Depot Vehicle Routing Problem ». In : *Electronic Notes in Discrete Mathematics* 66 (2018). 5th International Conference on Variable Neighborhood Search, p. 167-174. issn : 1571-0653. doi : https://doi.org/10.1016/j.endm.2018.03.022. url : http://www.sciencedirect.com/science/article/pii/S1571065318300684.

[12] U. Breunig, R. Baldacci, R.F. Hartl et T. Vidal. « The electric two-echelon vehicle routing problem ». In : *Computers& Operations Research* 103 (2019), p. 198-210. issn : 0305-0548. doi : https://doi.org/10.1016/j.cor.2018.11.005. url : http://www.sciencedirect.com/science/article/pii/S0305054818302909.

[13] Herminia I. Calvete, Carmen Galé, María-José Oliveros et Belén Sánchez-Valverde. « A goal programming approach to vehicle routing problems with soft time windows ». In : *European Journal of Operational Research* 177.3 (2007), p. 1720-1733. issn : 0377-2217. doi : https://doi.org/10.1016/j.ejor.2005.10.010. url : http://www.sciencedirect.com/science/article/pii/S0377221705006508.

[14] Diego Cattaruzza, Nabil Absi et Dominique Feillet. « Vehicle routing problems with multiple trips ». In : *4OR* 14.3 (sept. 2016), p. 223-259. issn : 1614-2411. doi : 10.1007/s10288-016-0306-2. url : https://doi.org/10.1007/s10288-016-0306-2.

[15] Xi Chen, Barrett W. Thomas et Mike Hewitt. « The technician routing problem with experience-based service times ». In : *Omega* 61 (2016), p. 49-61. issn : 0305-0483. doi : https://doi.org/10.1016/j.omega.2015.07.006. url : http://www.sciencedirect.com/science/article/pii/S0305048315001486.

[16] Hayet Chentli, Chentli Ouafi et Wahiba Ramdane Cherif-Khettaf. « Impact of Iterated Local Search Heuristic Hybridization on Vehicle Routing Problems : Application to the Capacitated Profitable Tour Problem ». In : *Operations Research and Enterprise Systems*. Sous la dir. de Greg H. Parlier, Federico Liberatore et Marc Demange. Cham : Springer International Publishing, 2019, p. 80-101. isbn : 978-3-030-16035-7.

[17] N. Christofides et J. E. Beasley. « The period routing problem ». In : *Networks* 14.2 (1984), p. 237-256. doi : 10.1002/net.3230140205. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230140205. url : https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230140205.

[18] N. Christofides et J. E. Beasley. « The period routing problem ». In : *Networks* 14.2 (1984), p. 237-256. doi : 10.1002/net.3230140205. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230140205. url : https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230140205.

[19] G. Clarke et J. W. Wright. « Scheduling of Vehicles from a Central Depot to a Number of Delivery Points ». In : *Oper. Res.* 12.4 (août 1964), p. 568-581. ISSN : 0030-364X. DOI : 10.1287/opre.12.4.568. URL : http://dx.doi.org/10.1287/opre.12.4.568.

[20] Ryan Conrad et Miguel Figliozzi. « The Recharging Vehicle Routing Problem ». In : *Proc. of the 61st Annual IIE Conference* (jan. 2011).

[21] Claudio Contardo et Rafael Martinelli. « A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints ». In : *Discrete Optimization* 12 (2014), p. 129-146. ISSN : 1572-5286. DOI : https://doi.org/10.1016/j.disopt.2014.03.001. URL : http://www.sciencedirect.com/science/article/pii/S1572528614000097.

[22] Jean-François Cordeau, Michel Gendreau et Gilbert Laporte. « A tabu search heuristic for periodic and multi-depot vehicle routing problems ». In : *Networks* 30.2 (1997), p. 105-119. DOI : 10.1002/(SICI)1097-0037(199709)30:2<105::AID-NET5>3.0.CO;2-G. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1002/%28SICI%291097-0037%28199709%2930%3A2%3C105%3A%3AAID-NET5%3E3.0.CO%3B2-G. URL : https://onlinelibrary.wiley.com/doi/abs/10.1002/%28SICI%291097-0037%28199709%2930%3A2%3C105%3A%3AAID-NET5%3E3.0.CO%3B2-G.

[23] Benoit Crevier, Jean-François Cordeau et Gilbert Laporte. « The multi-depot vehicle routing problem with inter-depot routes ». In : *European Journal of Operational Research* 176.2 (2007), p. 756-773. ISSN : 0377-2217. DOI : https://doi.org/10.1016/j.ejor.2005.08.015. URL : http://www.sciencedirect.com/science/article/pii/S0377221705006983.

[24] Z. J. Czech et P. Czarnas. « Parallel simulated annealing for the vehicle routing problem with time windows ». In : *Proceedings 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing.* 2002, p. 376-383. DOI : 10.1109/EMPDP.2002.994313.

[25] G. B. Dantzig et J. H. Ramser. « The Truck Dispatching Problem ». In : *Manage. Sci.* 6.1 (oct. 1959), p. 80-91. ISSN : 0025-1909. DOI : 10.1287/mnsc.6.1.80. URL : http://dx.doi.org/10.1287/mnsc.6.1.80.

[26] Eelco Den Boer, S Aarnink, F Kleiner et J Pagenkopf. « Zero emissions trucks : An overview of state-of-the-art technologies and their potential ». In : *CE Delf* (2013).

[27] Guy Desaulniers, Jacques Desrosiers, Andreas Erdmann, Marius M. Solomon et François Soumis. « 9. VRP with Pickup and Delivery ». In : *The Vehicle Routing Problem*, p. 225-242. DOI : 10.1137/1.9780898718515.ch9. eprint : https://epubs.siam.org/doi/pdf/10.1137/1.9780898718515.ch9. URL : https://epubs.siam.org/doi/abs/10.1137/1.9780898718515.ch9.

[28] Elizabeth D. Dolan et Jorge J. Moré. « Benchmarking optimization software with performance profiles ». In : *Mathematical Programming* 91.2 (jan. 2002), p. 201-213. ISSN : 1436-4646. DOI : 10.1007/s101070100263. URL : https://doi.org/10.1007/s101070100263.

[29] Lúcia M.A. Drummond, Luiz S. Ochi et Dalessandro S. Vianna. « An asynchronous parallel metaheuristic for the period vehicle routing problem ». In : *Future Generation Computer Systems* 17.4 (2001). Workshop on Bio-inspired Solutions to Parallel Computing problems, p. 379-386. ISSN : 0167-739X. DOI : https://doi.org/10.1016/S0167-739X(99)00118-1. URL : http://www.sciencedirect.com/science/article/pii/S0167739X99001181.

[30] Tomislav ERDELIĆ et Tonci CARIC. « A Survey on the Electric Vehicle Routing Problem : Variants and Solution Approaches ». In : *Journal of Advanced Transportation* 2019 (mai 2019), p. 1-48. DOI : 10.1155/2019/5075671.

[31] Sevgi ERDOĞAN et Elise MILLER-HOOKS. « A Green Vehicle Routing Problem ». In : *Transportation Research Part E : Logistics and Transportation Review* 48.1 (2012). Select Papers from the 19th International Symposium on Transportation and Traffic Theory, p. 100-114. ISSN : 1366-5545. DOI : https://doi.org/10.1016/j.tre.2011.08.001. URL : http://www.sciencedirect.com/science/article/pii/S1366554511001062.

[32] Airam EXPÓSITO, Julio BRITO, José A. MORENO et Christopher EXPÓSITO-IZQUIERDO. « Quality of service objectives for vehicle routing problem with time windows ». In : *Applied Soft Computing* 84 (2019), p. 105707. ISSN : 1568-4946. DOI : https://doi.org/10.1016/j.asoc.2019.105707. URL : http://www.sciencedirect.com/science/article/pii/S1568494619304880.

[33] Ángel FELIPE, M. Teresa ORTUÑO, Giovanni RIGHINI et Gregorio TIRADO. « A heuristic approach for the green vehicle routing problem with multiple technologies and partial recharges ». In : *Transportation Research Part E : Logistics and Transportation Review* 71 (2014), p. 111-128. ISSN : 1366-5545. DOI : https://doi.org/10.1016/j.tre.2014.09.003. URL : http://www.sciencedirect.com/science/article/pii/S1366554514001574.

[34] Roberto Álvarez FERNÁNDEZ. « A more realistic approach to electric vehicle contribution to greenhouse gas emissions in the city ». In : *Journal of Cleaner Production* 172 (2018), p. 949-959. ISSN : 0959-6526. DOI : https://doi.org/10.1016/j.jclepro.2017.10.158. URL : http://www.sciencedirect.com/science/article/pii/S0959652617324654.

[35] Matteo FISCHETTI, Paolo TOTH et Daniele VIGO. « A Branch-and-Bound Algorithm for the Capacitated Vehicle Routing Problem on Directed Graphs ». In : *Operations Research* 42 (oct. 1994), p. 846-859. DOI : 10.1287/opre.42.5.846.

[36] Peter FRANCIS et Karen SMILOWITZ. « Modeling techniques for periodic vehicle routing problems ». In : *Transportation Research Part B : Methodological* 40.10 (2006), p. 872-884. ISSN : 0191-2615. DOI : https://doi.org/10.1016/j.trb.2005.12.001. URL : http://www.sciencedirect.com/science/article/pii/S019126150600004X.

[37] Peter M. FRANCIS, Karen R. SMILOWITZ et Michal TZUR. « The Period Vehicle Routing Problem and its Extensions ». In : *The Vehicle Routing Problem : Latest Advances and New Challenges*. Sous la dir. de Bruce GOLDEN, S. RAGHAVAN et Edward WASIL. Boston, MA : Springer US, 2008, p. 73-102. ISBN : 978-0-387-77778-8. DOI : 10.1007/978-0-387-77778-8_4. URL : https://doi.org/10.1007/978-0-387-77778-8_4.

[38] Alex A FREITAS. *Data mining and knowledge discovery with evolutionary algorithms*. Springer Science & Business Media, 2013.

[39] Aurélien FROGER, Jorge E. MENDOZA, Ola JABALI et Gilbert LAPORTE. « Improved formulations and algorithmic components for the electric vehicle routing problem with nonlinear charging functions ». In : *Computers & Operations Research* 104 (2019), p. 256-294. ISSN : 0305-0548. DOI : https://doi.org/10.1016/j.cor.2018.12.013. URL : http://www.sciencedirect.com/science/article/pii/S0305054818303253.

[40] Ricardo Fukasawa, Humberto Longo, Jens Lysgaard, Marcus Poggi de Aragão, Marcelo Reis, Eduardo Uchoa et Renato F. Werneck. « Robust Branch-and-Cut-and-Price for the Capacitated Vehicle Routing Problem ». In : *Mathematical Programming* 106.3 (mai 2006), p. 491-511. issn : 1436-4646. doi : 10.1007/s10107-005-0644-x. url : https://doi.org/10.1007/s10107-005-0644-x.

[41] « Large Neighborhood Search ». English. In : *Handbook of Metaheuristics*. Sous la dir. de Michel Gendreau. 2e éd. Springer, 2010, p. 399-420. isbn : 978-1-4419-1663-1.

[42] Michel Gendreau, Alain Hertz et Gilbert Laporte. « A Tabu Search Heuristic for the Vehicle Routing Problem ». In : *Management Science* 40.10 (1994), p. 1276-1290. issn : 00251909, 15265501. url : http://www.jstor.org/stable/2661622.

[43] Konstantinos N. Genikomsakis et Georgios Mitrentsis. « A computationally efficient simulation model for estimating energy consumption of electric vehicles in the context of route planning applications ». In : *Transportation Research Part D : Transport and Environment* 50 (2017), p. 98-118. issn : 1361-9209. doi : https://doi.org/10.1016/j.trd.2016.10.014. url : http://www.sciencedirect.com/science/article/pii/S1361920915302881.

[44] Billy E. Gillett et Leland R. Miller. « A Heuristic Algorithm for the Vehicle-Dispatch Problem ». In : *Operations Research* 22.2 (1974), p. 340-349. doi : 10.1287/opre.22.2.340. eprint : https://doi.org/10.1287/opre.22.2.340. url : https://doi.org/10.1287/opre.22.2.340.

[45] Dominik Goeke et Michael Schneider. « Routing a mixed fleet of electric and conventional vehicles ». In : *European Journal of Operational Research* 245.1 (2015), p. 81-99. issn : 0377-2217. doi : https://doi.org/10.1016/j.ejor.2015.01.049. url : http://www.sciencedirect.com/science/article/pii/S0377221715000697.

[46] D. E. GOLDBERG. « Genetic Algorithms in Search ». In : *Optimization, and Machine-Learning* (1989). url : https://ci.nii.ac.jp/naid/10000038763/en/.

[47] Bruce Golden, Saahitya Raghavan et Edward Wasil. *The vehicle routing problem : Latest advances and new challenges*. T. 43. Jan. 2008. doi : 10.1007/978-0-387-77778-8.

[48] Frederico Gonçalves, Sónia R Cardoso, Susana Relvas et APFD Barbosa-Póvoa. « Optimization of a distribution network using electric vehicles : A VRP problem ». In : *Proceedings of the IO2011-15 Congresso da associação Portuguesa de Investigação Operacional, Coimbra, Portugal*. 2011, p. 18-20.

[49] E. A. Grunditz et T. Thiringer. « Performance Analysis of Current BEVs Based on a Comprehensive Review of Specifications ». In : *IEEE Transactions on Transportation Electrification* 2.3 (sept. 2016), p. 270-289. issn : 2372-2088. doi : 10.1109/TTE.2016.2571783.

[50] Gerhard Hiermann, Jakob Puchinger, Stefan Ropke et Richard F. Hartl. « The Electric Fleet Size and Mix Vehicle Routing Problem with Time Windows and Recharging Stations ». In : *European Journal of Operational Research* 252.3 (2016), p. 995-1018. issn : 0377-2217. doi : https://doi.org/10.1016/j.ejor.2016.01.038. url : http://www.sciencedirect.com/science/article/pii/S0377221716000837.

[51] Wanchen Jie, Jun Yang, Min Zhang et Yongxi Huang. « The two-echelon capacitated electric vehicle routing problem with battery swapping stations : Formulation and efficient methodology ». In : *European Journal of Operational Research* 272.3 (2019), p. 879-904. issn : 0377-2217. doi : https://doi.org/10.1016/j.ejor.2018.07.002. url : http://www.sciencedirect.com/science/article/pii/S0377221718306076.

[52] Cor de Jong, Goos Kant et A Van Vlient. « On finding minimal route duration in the vehicle routing problem with multiple time windows ». In : *Manuscript, Department of Computer Science, Utrecht University, Holland* (1996).

[53] Angel Alejandro Juan, Carlos Alberto Mendez, Javier Faulin, Jesica De Armas et Scott Erwin Grasman. « Electric Vehicles in Logistics and Transportation : A Survey on Emerging Environmental, Strategic, and Operational Challenges ». In : *Energies* 9.2 (2016). issn : 1996-1073. doi : 10.3390/en9020086. url : https://www.mdpi.com/1996-1073/9/2/86.

[54] Merve Keskin et Bülent Çatay. « A matheuristic method for the electric vehicle routing problem with time windows and fast chargers ». In : *Computers & Operations Research* 100 (2018), p. 172-188. issn : 0305-0548. doi : https://doi.org/10.1016/j.cor.2018.06.019. url : http://www.sciencedirect.com/science/article/pii/S0305054818301849.

[55] Merve Keskin, Gilbert Laporte et Bülent Çatay. « Electric Vehicle Routing Problem with Time-Dependent Waiting Times at Recharging Stations ». In : *Computers & Operations Research* 107 (2019), p. 77-94. issn : 0305-0548. doi : https://doi.org/10.1016/j.cor.2019.02.014. url : http://www.sciencedirect.com/science/article/pii/S030505481930053X.

[56] Çağrı Koç et Gilbert Laporte. « Vehicle routing with backhauls : Review and research perspectives ». In : *Computers& Operations Research* 91 (2018), p. 79-91. issn : 0305-0548. doi : https://doi.org/10.1016/j.cor.2017.11.003. url : http://www.sciencedirect.com/science/article/pii/S0305054817302794.

[57] Philippe Lacomme, Christian Prins et Wahiba Ramdane Cherif. « Evolutionary algorithms for periodic arc routing problems ». In : *European Journal of Operational Research* 165 (sept. 2005), p. 535-553. doi : 10.1016/j.ejor.2004.04.021.

[58] Gilbert Laporte, Hélène Mercure et Yves Nobert. « An exact algorithm for the asymmetrical capacitated vehicle routing problem ». In : *Networks* 16.1 (1986), p. 33-46. doi : 10.1002/net.3230160104. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230160104. url : https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230160104.

[59] Lu Li, Hong K. Lo, Feng Xiao et Xuekai Cen. « Mixed bus fleet management strategy for minimizing overall and emissions external costs ». In : *Transportation Research Part D : Transport and Environment* 60 (2018). Special Issue on Traffic Modeling for Low-Emission Transport, p. 104-118. issn : 1361-9209. doi : https://doi.org/10.1016/j.trd.2016.10.001. url : http://www.sciencedirect.com/science/article/pii/S1361920916306757.

[60] Ran Liu et Zhibin Jiang. « A hybrid large-neighborhood search algorithm for the cumulative capacitated vehicle routing problem with time-window constraints ». In : *Applied Soft Computing* 80 (2019), p. 18-30. issn : 1568-4946. doi : https://doi.org/10.1016/j.asoc.2019.03.008. url : http://www.sciencedirect.com/science/article/pii/S1568494619301267.

[61] Jens Lysgaard, Adam N. Letchford et Richard W. Eglese. « A new branch-and-cut algorithm for the capacitated vehicle routing problem ». In : *Mathematical Programming* 100.2 (juin 2004), p. 423-445. issn : 1436-4646. doi : 10.1007/s10107-003-0481-8. url : https://doi.org/10.1007/s10107-003-0481-8.

[62] Silvano Martello et Paolo Toth. *Knapsack Problems : Algorithms and Computer Implementations*. New York, NY, USA : John Wiley & Sons, Inc., 1990. isbn : 0-471-92420-2.

[63] Mohamed Amine Masmoudi, Manar Hosny, Emrah Demir, Konstantinos N. Genikomsakis et Naoufel Cheikhrouhou. « The dial-a-ride problem with electric vehicles and battery swapping stations ». In : *Transportation Research Part E : Logistics and Transportation Review* 118 (2018), p. 392-420. issn : 1366-5545. doi : https://doi.org/10.1016/j.tre.2018.08.005. url : http://www.sciencedirect.com/science/article/pii/S1366554517310086.

[64] Ministère de la Transition écologique et solidaire. *LOI n 2015-992 du 17 août 2015 relative à la transition énergétique pour la croissance verte*. https://www.legifrance.gouv.fr/affichTexte.do;jsessionid=0C02F70EC4B7D9DDC1D418613190B55D.tplgfr21s_2?cidTexte=JORFTEXT000031044385&idArticle=&dateTexte=20190716.

[65] S. Mirmohammadi, Erfan Babaee Tirkolaee, Alireza Goli et Saeed Dehnavi-Arani. « The periodic green vehicle routing problem with considering of time-dependent urban traffic and time windows ». In : 7 (août 2016), p. 143-156.

[66] Anis Mjirda. « Recherche à voisinage variable pour des problèmes de routage avec ou sans gestion de stock ». 2014VALE0023. Thèse de doct. 2014. url : http://www.theses.fr/2014VALE0023/document.

[67] N. Mladenović et P. Hansen. « Variable neighborhood search ». In : *Computers& Operations Research* 24.11 (1997), p. 1097-1100. issn : 0305-0548. doi : https://doi.org/10.1016/S0305-0548(97)00031-2. url : http://www.sciencedirect.com/science/article/pii/S0305054897000312.

[68] Alejandro Montoya, Christelle Guéret, Jorge E. Mendoza et Juan G. Villegas. « A multi-space sampling heuristic for the green vehicle routing problem ». In : *Transportation Research Part C : Emerging Technologies* 70 (2016), p. 113-128. issn : 0968-090X. doi : https://doi.org/10.1016/j.trc.2015.09.009. url : http://www.sciencedirect.com/science/article/pii/S0968090X15003320.

[69] Alejandro Montoya, Christelle Guéret, Jorge E. Mendoza et Juan G. Villegas. « The electric vehicle routing problem with nonlinear charging function ». In : *Transportation Research Part B : Methodological* 103 (2017). Green Urban Transportation, p. 87-110. issn : 0191-2615. doi : https://doi.org/10.1016/j.trb.2017.02.004. url : http://www.sciencedirect.com/science/article/pii/S0191261516304556.

[70] M. Mourgaya et F. Vanderbeck. « Column generation based heuristic for tactical planning in multi-period vehicle routing ». In : *European Journal of Operational Research* 183.3 (2007), p. 1028-1041. issn : 0377-2217. doi : https://doi.org/10.1016/j.ejor.2006.02.030. url : http://www.sciencedirect.com/science/article/pii/S0377221706002293.

[71] M. Mourgaya et F. Vanderbeck. « The periodic Vehicle routing problem : classification and heuristic ». In : *RAIRO - Operations Research* 40.2 (2006), p. 169-194. doi : 10.1051/ro:2006015.

[72] Sophie N. Parragh, Karl F. Doerner et Richard F. Hartl. « A survey on pickup and delivery problems ». In : *Journal für Betriebswirtschaft* 58.1 (avr. 2008), p. 21-51. issn : 1614-631X. doi : 10.1007/s11301-008-0033-7. url : https://doi.org/10.1007/s11301-008-0033-7.

[73] Sophie N. Parragh, Karl F. Doerner et Richard F. Hartl. « A survey on pickup and delivery problems ». In : *Journal für Betriebswirtschaft* 58.2 (juin 2008), p. 81-117. issn : 1614-631X. doi : 10.1007/s11301-008-0036-4. url : https://doi.org/10.1007/s11301-008-0036-4.

[74] Samuel Pelletier, Ola Jabali et Gilbert Laporte. « The electric vehicle routing problem with energy consumption uncertainty ». In : *Transportation Research Part B : Methodological* 126 (2019), p. 225-255. issn : 0191-2615. doi : https://doi.org/10.1016/j.trb.2019.06.006. url : http://www.sciencedirect.com/science/article/pii/S0191261519300153.

[75] Samuel Pelletier, Ola Jabali, Gilbert Laporte et Marco Veneroni. « Battery degradation and behaviour for electric vehicles : Review and numerical analyses of several models ». In : *Transportation Research Part B : Methodological* 103 (2017). Green Urban Transportation, p. 158-187. issn : 0191-2615. doi : https://doi.org/10.1016/j.trb.2017.01.020. url : http://www.sciencedirect.com/science/article/pii/S0191261516303794.

[76] Samuel Pelletier, Ola Jabali, Jorge E. Mendoza et Gilbert Laporte. « The electric bus fleet transition problem ». In : *Transportation Research Part C : Emerging Technologies* 109 (2019), p. 174-193. issn : 0968-090X. doi : https://doi.org/10.1016/j.trc.2019.10.012. url : http://www.sciencedirect.com/science/article/pii/S0968090X1930868X.

[77] Bo Peng, Yuan Zhang, Yuvraj Gajpal et Xiding Chen. « A Memetic Algorithm for the Green Vehicle Routing Problem ». In : *Sustainability* 11.21 (2019), p. 6055.

[78] Victor Pillac, Christelle Guéret et Andrés Medaglia. « On the Technician Routing and Scheduling Problem ». In : juil. 2011, p. 675-678. isbn : 9788890098437.

[79] David Pisinger et Stefan Ropke. « A general heuristic for vehicle routing problems ». In : *Computers & Operations Research* 34.8 (2007), p. 2403-2435. issn : 0305-0548. doi : https://doi.org/10.1016/j.cor.2005.09.012. url : http://www.sciencedirect.com/science/article/pii/S0305054805003023.

[80] C. Prins, N. Labadi et M. Reghioui. « Tour splitting algorithms for vehicle routing problems ». In : *International Journal of Production Research* 47.2 (2009), p. 507-535. doi : 10.1080/00207540802426599. eprint : https://doi.org/10.1080/00207540802426599. url : https://doi.org/10.1080/00207540802426599.

[81] Christian Prins. « A simple and effective evolutionary algorithm for the vehicle routing problem ». In : *Computers& Operations Research* 31.12 (2004), p. 1985-2002. ISSN : 0305-0548. DOI : https://doi.org/10.1016/S0305-0548(03)00158-8. URL : http://www.sciencedirect.com/science/article/pii/S0305054803001588.

[82] Christian Prins, Philippe Lacomme et Caroline Prodhon. « Order-first split-second methods for vehicle routing problems : A review ». In : *Transportation Research Part C : Emerging Technologies* 40 (2014), p. 179-200. ISSN : 0968-090X. DOI : https://doi.org/10.1016/j.trc.2014.01.011. URL : http://www.sciencedirect.com/science/article/pii/S0968090X14000230.

[83] H. G. M. Pullen et M. H. J. Webb. « A computer application to a transport scheduling problem ». In : *The Computer Journal* 10.1 (jan. 1967), p. 10-13. ISSN : 0010-4620. DOI : 10.1093/comjnl/10.1.10. eprint : http://oup.prod.sis.lan/comjnl/article-pdf/10/1/10/1068241/100010.pdf. URL : https://doi.org/10.1093/comjnl/10.1.10.

[84] T.K. Ralphs, L. Kopman, W.R. Pulleyblank et L.E. Trotter. « On the capacitated vehicle routing problem ». In : *Mathematical Programming* 94.2 (jan. 2003), p. 343-359. ISSN : 1436-4646. DOI : 10.1007/s10107-002-0323-0. URL : https://doi.org/10.1007/s10107-002-0323-0.

[85] Wahiba Ramdane Cherif- Khettaf, Mais Haj Rachid, Christelle Bloch et Pascal Chatonnay. « New Notation and Classification Scheme for Vehicle Routing Problems ». In : *RAIRO - Operations Research* 49.1 (2015), p. 161-194. DOI : 10.1051/ro/2014030. URL : https://hal.archives-ouvertes.fr/hal-01307009.

[86] Marc Reimann, Karl Doerner et Richard F Hartl. « D-Ants : Savings Based Ants divide and conquer the vehicle routing problem ». In : *Computers& Operations Research* 31.4 (2004), p. 563-591. ISSN : 0305-0548. DOI : https://doi.org/10.1016/S0305-0548(03)00014-5. URL : http://www.sciencedirect.com/science/article/pii/S0305054803000145.

[87] Jorge Restrepo, Javier Rosero Garcia et Sandra Téllez Gutiérrez. « Performance Testing of Electric Vehicles on operating conditions in Bogotá DC, Colombia ». In : sept. 2014.

[88] Francesc Robust, Carlos F. Daganzo et Reginald R. Souleyrette. « Implementing vehicle routing models ». In : *Transportation Research Part B : Methodological* 24.4 (1990), p. 263-286. ISSN : 0191-2615. DOI : https://doi.org/10.1016/0191-2615(90)90002-G. URL : http://www.sciencedirect.com/science/article/pii/019126159090002G.

[89] Stefan Ropke et David Pisinger. « An Adaptive Large Neighborhood Search Heuristic for the Pickup and Delivery Problem with Time Windows ». In : *Transportation Science* 40.4 (2006), p. 455-472. DOI : 10.1287/trsc.1050.0135. eprint : https://doi.org/10.1287/trsc.1050.0135. URL : https://doi.org/10.1287/trsc.1050.0135.

[90] R. Russell et W. Igo. « An assignment routing problem ». In : *Networks* 9.1 (1979), p. 1-17. DOI : 10.1002/net.3230090102. eprint : https://onlinelibrary.wiley.com/doi/pdf/10.1002/net.3230090102. URL : https://onlinelibrary.wiley.com/doi/abs/10.1002/net.3230090102.

[91] Ons Sassi. « Planification de la recharge et optimisation des tournées dans le cas de flottes captives ». 2015LORR0303. Thèse de doct. 2015. URL : http://www.theses.fr/2015LORR0303/document.

[92] Ons Sassi, Wahiba Ramdane-Cherif et Ammar Oulamara. « Iterated Tabu Search for the Mix Fleet Vehicle Routing Problem with Heterogenous Electric Vehicles ». In : *Modelling, Computation and Optimization in Information Systems and Management Sciences*. Sous la dir. d'Hoai An Le Thi, Tao Pham Dinh et Ngoc Thanh Nguyen. Cham : Springer International Publishing, 2015, p. 57-68. isbn : 978-3-319-18161-5.

[93] Ons Sassi, Wahiba Ramdane-Cherif et Ammar Oulamara. « Multi-Start Iterated Local Search for the Mixed fleet Vehicle Routing Problem with Heterogeneous Electric Vehicles and Time-Dependent Charging Costs ». In : *Lecture Notes in Computer Science* 9026 (2015), p. 138-149.

[94] Ons Sassi, Wahiba Ramdane-Cherif et Ammar Oulamara. « Multi-start Iterated Local Search for the Mixed Fleet Vehicle Routing Problem with Heterogenous Electric Vehicles ». In : *Evolutionary Computation in Combinatorial Optimization*. Sous la dir. de Gabriela Ochoa et Francisco Chicano. Cham : Springer International Publishing, 2015, p. 138-149. isbn : 978-3-319-16468-7.

[95] Ons Sassi, Wahiba Ramdane-Cherif et Ammar Oulamara. « Vehicle Routing Problem with Mixed fleet of conventional and heterogenous electric vehicles and time dependent charging costs ». working paper or preprint. Oct. 2014. url : https://hal.archives-ouvertes.fr/hal-01083966.

[96] Michael Schneider, Andreas Stenger et Dominik Goeke. « The Electric Vehicle-Routing Problem with Time Windows and Recharging Stations ». In : *Transportation Science* 48.4 (2014), p. 500-520. doi : 10.1287/trsc.2013.0490. eprint : https://doi.org/10.1287/trsc.2013.0490. url : https://doi.org/10.1287/trsc.2013.0490.

[97] Paul Shaw. « Using Constraint Programming and Local Search Methods to Solve Vehicle Routing Problems ». In : *Principles and Practice of Constraint Programming — CP98*. Sous la dir. de Michael Maher et Jean-Francois Puget. Berlin, Heidelberg : Springer Berlin Heidelberg, 1998, p. 417-431. isbn : 978-3-540-49481-2.

[98] Marius M. Solomon. « Algorithms for the Vehicle Routing and Scheduling Problems with Time Window Constraints ». In : *Operations Research* 35.2 (1987), p. 254-265. doi : 10.1287/opre.35.2.254. eprint : https://doi.org/10.1287/opre.35.2.254. url : https://doi.org/10.1287/opre.35.2.254.

[99] Christos D. Tarantilis, Emmanouil E. Zachariadis et Chris T. Kiranoudis. « A Hybrid Guided Local Search for the Vehicle-Routing Problem with Intermediate Replenishment Facilities ». In : *INFORMS Journal on Computing* 20.1 (2008), p. 154-168. doi : 10.1287/ijoc.1070.0230. eprint : https://doi.org/10.1287/ijoc.1070.0230. url : https://doi.org/10.1287/ijoc.1070.0230.

[100] Michaël Torregrossa. *Tout savoir sur le plan national pour le développement des véhicules électriques et hybrides*. url : http://www.avem.fr/actualite-tout-savoir-sur-le-plan-national-pour-le-developpement-des-vehicules-electriques-et-hybrides-1014.html.

[101] Paolo Toth et Daniele Vigo. « 2. Branch-And-Bound Algorithms for the Capacitated VRP ». In : *The Vehicle Routing Problem*, p. 29-51. doi : 10.1137/1.9780898718515.ch2. eprint : https://epubs.siam.org/doi/pdf/10.1137/1.9780898718515.ch2. url : https://epubs.siam.org/doi/abs/10.1137/1.9780898718515.ch2.

[102] Paolo Toth et Daniele Vigo. « An Exact Algorithm for the Vehicle Routing Problem with Backhauls ». In : *Transportation Science* 31 (nov. 1997), p. 372-385. doi : 10.1287/trsc.31.4.372.

[103] Paolo Toth, Daniele Vigo, Paolo Toth et Daniele Vigo. *Vehicle Routing : Problems, Methods, and Applications, Second Edition*. Philadelphia, PA, USA : Society for Industrial et Applied Mathematics, 2014. isbn : 1611973589, 9781611973587.

[104] Thibaut Vidal, Gilbert Laporte et Piotr Matl. « A concise guide to existing and emerging vehicle routing problem variants ». In : *European Journal of Operational Research* (2019). issn : 0377-2217. doi : https://doi.org/10.1016/j.ejor.2019.10.010. url : http://www.sciencedirect.com/science/article/pii/S0377221719308422.

[105] M. Wen, E. Linde, S. Ropke, P. Mirchandani et A. Larsen. « An adaptive large neighborhood search heuristic for the Electric Vehicle Scheduling Problem ». In : *Computers& Operations Research* 76 (2016), p. 73-83. issn : 0305-0548. doi : https://doi.org/10.1016/j.cor.2016.06.013. url : http://www.sciencedirect.com/science/article/pii/S0305054816301460.

[106] A. Zhang, J.E. Jee Eun Kang et C. Changhyun Kwon. « Incorporating demand dynamics in multi-period capacitated fast-charging location planning for electric vehicles ». In : *Transportation Research Part B* 103 (2017), p. 5-29.

Résumé

Le secteur des transports est responsable d'un fort accroissement de la consommation énergétique et des émissions de polluants. La mobilité électrique constitue une bonne alternative, et offre un mode de transport durable et cohérent avec les exigences de l'environnement.

Les activités de R&D sur le Véhicule Électrique (VE) font l'objet de nombreux travaux de recherche et peuvent être classées en quatre catégories de problématiques : modélisation des usages des véhicules électriques et Smart Grid, dimensionnement de l'infrastructure de recharge, optimisation de la recharge et optimisation des tournées des véhicules électriques. Concernant l'optimisation des tournées de véhicules électriques, cette problématique a été largement étudiée ces dernières années et plusieurs variantes du modèle de base, nommé EVRP pour Electric Vehicle Routing Problem, ont été étudiées, mais les modèles proposés ne concernent qu'un horizon à court terme, ignorant ainsi les décisions tactiques liées à l'affectation des clients et à la planification de la recharge sur un horizon temporel.

Dans cette thèse, nous proposons une extension temporelle du EVRP appelée PEVRP ou problème de planification des tournées de véhicules électriques. Un horizon de plusieurs périodes ou jours est considéré, une fréquence de visite est donnée pour chaque client (exemple 2 fois par semaine), et une flotte homogène de véhicules électriques est disponible, dont la capacité d'emport, la capacité de recharge de la batterie, ainsi que le temps de travail sont limités. Nous considérons une recharge partielle des véhicules. Le but est d'affecter les clients aux périodes, en respectant les fréquences et des contraintes temporelles (décision tactique), et de construire les tournées dans chaque période (décision opérationnelle), en respectant les contraintes ci-dessus de façon à minimiser un coût total sur l'horizon incluant un coût de recharge. La prise en compte d'un horizon de planification pour l'optimisation des tournées avec des véhicules électriques permet d'anticiper les recharges sur l'horizon pour mieux optimiser la consommation énergétique des véhicules.

Nous proposons trois nouveaux modèles de PEVRP. Le premier modèle considère une flotte limitée de véhicules et un nombre de visites sur l'horizon limité à un pour chaque client. Le deuxième modèle généralise le précédent modèle en relaxant la contrainte sur la limitation des visites, et en autorisant de visiter plusieurs fois un client sur l'horizon, tout en respectant la contrainte d'une visite par jour au maximum par client et des contraintes d'espacement de visites exprimées en un ensemble de combinaisons imposées par le client ; et enfin, le dernier modèle, le plus général, considère le deuxième modèle en relaxant aussi la contrainte de limitation du nombre de véhicules.

Nous proposons des heuristiques spécifiques aux problèmes étudiés et des approches métaheuristiques à base de recherche à grand voisinage. Ces méthodes utilisent des procédures spécifiques pour la prise en compte des contraintes de consommation d'énergie des véhicules permettant d'ajuster, et d'optimiser la recharge des solutions parcourues pour maintenir des solutions faisables au cours du processus d'optimisation. Un nombre important de nouvelles instances ont été générées pour valider les méthodes proposées.

Mots-clés: Recherche opérationnelle

Abstract

The transport sector is responsible for a considerable increase in energy consumption and pollutant emissions. Electric mobility is a good alternative and offers a sustainable mode of transport that is coherent with the environmental requirements.

Research activities on the Electric Vehicle (EV) are the subject of numerous studies and can be classified into four categories of issues : modeling the use of electric vehicles and Smart Grid, dimensioning the recharging infrastructure, optimization of recharging and optimization of electric vehicle routing. Concerning the electric vehicle routing optimization, this issue has been widely studied in recent years and several variants of the basic model, called EVRP for Electric Vehicle Routing Problem, have been studied, but the proposed models concern only a short-term horizon, thus ignoring tactical decisions related to customer allocation and recharging planning over a time horizon.

In this thesis, we propose a temporal extension of the EVRP, called PEVRP or Periodic Electric Vehicle Routing Problem. A horizon of several periods or days is considered, a visit frequency is given for each customer (for example twice a week), and a homogeneous fleet of electric vehicles is available, whose load capacity, battery recharging capacity and working time are limited. We consider a partial recharging of the vehicles. The goal is to assign customers to periods, respecting frequencies and time constraints (tactical decision), and to build the routes in each period (operational decision), respecting the above constraints in order to minimize a total cost over the horizon including a recharging cost. Considering a planning horizon for the optimization of electric vehicle routing problems lets anticipating recharging over the horizon to better optimize the vehicles' energy consumption.

We propose three new PEVRP models. The first model considers a limited fleet of vehicles and a number of visits on the horizon limited to one for each customer. The second model generalizes the previous model by relaxing the limitation of visits, and by allowing visiting a customer several times on the horizon, while respecting the constraint of a maximum of one visit per day per customer and the spacing constraints of visits defined as a set of combinations imposed by the customer ; and finally, the last model, the more general one, considers the second model by also relaxing the constraint of limiting the number of vehicles.

We propose specific heuristics for studied problems and metaheuristic approaches based on large neighborhood research. These methods use specific procedures to consider the vehicle energy consumption constraints. These procedures aim to adjust, and to optimize the recharging of solutions encountered in order to maintain feasible solutions during the optimization process. A large number of new instances have been generated to validate the proposed methods.

Keywords: optimization