Université de Montréal

Méthode de recherche à grand voisinage pour un problème de tournées de véhicules avec flotte privée et transporteur externe

par

Frédéric Aka Bilé EDOUKOU

Département d'informatique et de recherche opérationnelle

Faculté des arts et des sciences

Mémoire présenté

en vue de l'obtention du grade de

Maître ès sciences (M.Sc.)

en informatique

Avril, 2016

Université de Montréal

Ce mémoire intitulé

Méthode de recherche à grand voisinage pour un problème de tournées de véhicules avec flotte privée et transporteur externe

présenté par:

Frédéric Aka Bilé EDOUKOU

a été évalué par un jury composé des personnes suivantes:

Guy Lapalme, président-rapporteur

Patrice Marcotte, membre du jury

Jean-Yves Potvin, directeur de recherche

Résumé

Dans ce mémoire, nous étudions un problème de tournées de véhicules dans lequel une flotte privée de véhicules n'a pas la capacité suffisante pour desservir les demandes des clients. Dans un tel cas, on fait appel à un transporteur externe. Ce dernier n'a aucune contrainte de capacité, mais un coût est encouru lorsqu'un client lui est affecté.

Il n'est pas nécessaire de mettre tous les véhicules de la flotte privée en service si cette approche se révèle plus économique. L'objectif consiste à minimiser le coût fixe des véhicules, puis le coût variable de transport et le coût chargé par le transporteur externe.

Notre travail consiste à appliquer la métaheuristique de recherche adaptative à grand voisinage sur ce problème. Nous comparons nos résultats avec ceux obtenus précédemment avec différentes techniques connues sur les instances de Christofides et celles de Golden.

Mots-clés : Tournées de véhicules, flotte privée, transporteur externe, recherche adaptative, grand voisinage, métaheuristique.

Abstract

In this master thesis, we study a vehicle routing problem in which a private fleet does not have sufficient capacity to serve all customers. Therefore, an external common carrier is required. The external common carrier has no constraint of capacity, but there is a cost when a customer it assigned to it.

It is not necessary for all the vehicles of the private fleet to be used. The objective is to minimize the sum of the fixed cost of the private fleet, the variable routing cost and the external carrier cost.

Our work applies the adaptative large neighborhood search metaheuristic on this problem. We compare our results with those obtained previously with different well-known techniques on the benchmark instances of Christofides and Golden.

Keywords : Vehicle routing problem, private fleet, common carrier, adaptative large neighborhood search, metaheuristic.

Table des matières

Liste des tableaux

Liste des figures

Liste des sigles

ALNS : Adaptative Large Neighborhood Search

AVNS : Adaptative Variable Neighborhood Search

BRBL : Bolduc Renaud Boctor and Laporte

CCAO : Convex hull, Cheapest insertion, Angle selection and Or-opt

CG : Column Generation

CPTP : Capacited Profitable Tour Problem

DPCS : Distribution Problem with Carrier Service

GA : Genetic Algorithm

IDEA : Iterated Density Estimation Evolutionary Algorithm

I^3 : Initialization, Insertion, and Improvement

MDVRPPC: Multi-Depot Vehicle Routing Problem with Private fleet and common Carrier

MS-ILS : Multi-Start Iterated Local Search

ML-LS : Multi-Start Local Search

MLP : Modified Linear Programming

PCTSP : Prize Collecting Traveling Salesman Problem

PDPTW : Pickup and Delivery Problem With Time Windows

RCSP : Routing and Carrier Selection Problem

RIP : Randomized construction, Improvement and Pertubation

SCP : Set Covering Problem

SRI : Selection, Routing and Improvement

STSP : Symmetric Traveling Salesman Problem

VNS : Variable Neighborhood Search

TOP: Team Orienteering Problem

TS : Tabu Search

TS+ : Tabu Search amelioré

TSP : Traveling Salesman Problem

TSPG : Traveling Salesman Problem Généralisé

TS+ : Tabou Search avec chaînes d'éjections

UHGS: Unified Hybrid Genetic Search

VRP : Vehicle Routing Problem

VRPPC : Vehicle Routing Problem with Private fleet and common Carrier

VRPPFCC: Vehicle Routing Problem with Private Fleet and Common Carrier

VRPTW : Vehicle Routing Problem with Time Windows

2-VRPP : 2-ressources Vehicle Routing Problem with Profits

À ma famille,
elle a toujours été
ma source d'inspiration.

Remerciements

Je remercie vivement mon valeureux et dynamique directeur de recherche, Jean-Yves Potvin, pour m'avoir proposé ce sujet de recherche passionnant et apporté son soutien ainsi que de nombreuses idées nécessaires à sa réalisation. C'est d'ailleurs lui qui m'a finalement convaincu que faire une maîtrise en informatique au DIRO (Département d'Informatique et de Recherche Opérationnelle) serait plus avantageux pour moi qu'un baccalauréat. Ses compétences scientifiques, sa patience et le financement qu'il m'a accordé (Janvier 2014 - Décembre 2015), m'ont permis de surmonter de nombreuses difficultés liées à ce travail.

Je remercie le directeur des études supérieures du DIRO, Philippe Langlais, pour son appui dans l'obtention d'une prolongation (session Hiver 2016) pour achever mon mémoire et pour la composition d'un jury incarnant les deux axes de recherche du DIRO. Je remercie également les professeurs Guy Lapalme et Patrice Marcotte, deux anciens directeurs du DIRO et experts hautement respectés au Canada respectivement en intelligence artificielle (IA) et en recherche opérationnelle (RO), pour leur participation au jury et pour leur bonne lecture de mon travail. Leurs suggestions et remarques reflétant deux profils complémentaires, m'ont permis d'améliorer la qualité de mon travail. C'est un honneur pour moi d'avoir une appréciation finale de leur part qui m'enchante.

Je profite de cette occasion pour rendre un hommage au mathématicien Jacques St-Pierre fondateur du DIRO en 1966 et de la DESI (Direction de l'Enseignement de Service en Informatique) en 1984 à l'Université de Montréal (UdeM) pour sa grande vision. Je remercie très sincèrement tous les professeurs, les chargés de cours, les auxiliaires d'enseignement et le personnel administratif du DIRO et de la DESI pour la continuité, leur travail contribue à faire de Montréal un bon pôle recherche mondial en informatique et en recherche

opérationnelle. J'avais auparavant une vague idée de la RO, mais maintenant j'ai compris sa portée interdisciplinaire. La vie nécessite souvent une planification et des prises de décisions, d'où la RO. Mes remerciements vont aux organisateurs, conférenciers et participants du vingtième anniversaire du DIRO. En effet les travaux exposés, de même que les discussions durant cet évènement, m'ont permis non seulement de connaître l'histoire de l'informatique au Canada depuis sa genèse jusqu'à maintenant mais surtout les grandes avancées et les limites actuelles de cette discipline.

Je tiens à remercier infiniment Serge Bisaillon, analyste principal et soutien logiciel du CIRRELT (Centre Interuniversitaire de Recherche sur les Réseaux d'Entreprise, la Logistique et le Transport) pour les nombreuses séances de travail qui ont permis d'améliorer la qualité de mon code. Je remercie le personnel administratif et informatique du CIRRELT, et le personnel informatique du GERAD (Groupe d'Études et de Recherche en Analyse des Décisions) pour leur soutien. Mes remerciements vont également à Calcul Canada, à Calcul Québec et à leurs équipes de soutien technique. Je remercie également tous les thésards et les post-docs du CIRRELT et du DIRO avec qui j'ai eu des séances de discussion concernant mes algorithmes.

Je remercie les professeurs Le Van N'Guyen, Yves Claude et Michel Reid chargés de cours à la DESI de l'Université de Montréal qui m'ont recruté pour des tâches d'enseignement en informatique pendant dix sessions académiques (Janvier 2013 - Avril 2016). Ce fut une bonne expérience pédagogique de travailler avec eux. Cela m'a aussi apporté une source de financement pour subvenir aux besoins de ma famille. Je remercie la Biliothèque de mathématiques et informatique de l'UdeM et son personnel, pour ma documentation et pour son environnement de travail très studieux mis à notre disposition.

Je voudrais également exprimer ma joie envers moi même car c'est un vieux rêve enfoui dans ma petite tête qui se réalise maintenant après quinze années au prix de plusieurs difficultés. En effet en Mai 2001, vivant à cette époque en France, je suis allé en voyage d'exploration aux États-Unis pendant trois mois. Un passage au quartier général de la NASA à Washington D.C. me fit prendre conscience que la maîtrise des mathématiques et de l'informatique

permettraient d'apporter des réponses à de grands problèmes et d'explorer des horizons plus lointains en science. J'avais donc ce rêve de maîtriser l'informatique, après mon doctorat en mathématiques, pour m'attaquer à de grands problèmes en science. Je suis maintenant libéré de cette dette qui était stockée dans la base de données de mon subconscient.

Enfin, je tiens à remercier ma famille à qui j'ai dû imposer de nombreux sacrifices. Le temps de passer à une autre étape de notre vie a grandement sonné!

Chapitre 1

Introduction

Mon sujet de recherche se situe dans le domaine de l'optimisation combinatoire. Il s'agit d'une discipline à l'interface des mathématiques et de l'informatique dans laquelle on traite des problèmes où l'on doit extraire un meilleur élément dans un ensemble fini. Quoique fini, l'ensemble comporte en général un grand nombre d'éléments, rendant impossible la solution par énumération de toutes les solutions possibles. On est donc amené à mettre en évidence certaines structures du modèle étudié et à élaborer différentes méthodes de résolution [51].

En ce qui concerne les problèmes d'optimisation combinatoire, il faut noter que ce sont plus souvent des heuristiques et des métaheuristiques qui sont utilisées et qui ont eu plus de succès. Ce sont souvent des algorithmes de nature stochastique qui progressent vers un extremum global d'une fonction objectif. Ces algorithmes s'arrêtent après avoir satisfait un critère d'arrêt. Les méthodes exactes ont été aussi utilisées et ont pu produire des résultats. Cependant pour des problèmes de très grande complexité, leur efficacité est réduite [36].

1.1 Problématique des tournées de véhicules

Dans le cadre de ce travail, nous allons nous intéresser à un problème de transport. Plus spécifiquement, on a une flotte privée constituée de m véhicules tous localisés à un dépôt. Chaque véhicule a une capacité maximale fixée Q_k et un coût fixe f_k lorsqu'il est mis en service.

Définition 1.1. *Soit une une flotte privée de véhicules. On dit qu'elle est homogène si tous*

ses véhicules ont la même capacité maximale et le même coût fixe de mise en service. Elle est dite hétérogène dans le cas contraire.

Soit n le nombre de clients qui sont répartis dans la zone desservie. Chaque client i a une demande q_i et on souhaite desservir tous les clients. Chaque fois qu'un véhicule est utilisé, il part du dépôt, dessert des clients et termine son itinéraire au dépôt. Au cours de chaque tournée d'un véhicule, il passe une seule fois chez les clients qu'il visite. La charge qu'il transporte ne peut excéder sa capacité maximale. D'autre part, la capacité totale de tous les véhicules de la flotte privée est un paramètre majeur du problème. En effet elle peut être supérieure ou égale à la somme de toutes les demandes des clients. C'est le problème de base appelé en anglais VRP (Vehicle Routing Problem). Il a été étudié par plusieurs spécialistes du domaine du transport avec une panoplie d'outils algorithmiques. Plusieurs articles ont été publiés sur cette question [19, 21, 36, 37, 56].

Dans le problème des tournées de véhicules, il peut arriver que la capacité de la flotte privée est strictement inférieure à la somme de toutes les demandes des clients. Dans ce contexte il est tout à fait naturel de faire appel à un transporteur externe. Le transporteur externe est indépendant de la flotte privée et n'a aucune contrainte de capacité. Seulement, pour chaque client affecté au transporteur externe un coût lui est associé. Dans la terminologie du domaine, ce coût est appelé une pénalité et ce problème est appélé VRPPC (Vehicle Routing Problem with Private Fleet and Common Carrier). Il n'est donc pas obligatoire de mettre tous les véhicules de la flotte privée en service, ce qui importe c'est de minimiser le coût de la fonction objectif. Laquelle fonction est la somme de trois coûts: le coût fixe par véhicule, le coût variable de transport et le coût chargé par le transporteur externe.

Ce problème a été relativement peu étudié dans le domaine du transport. C'est d'ailleurs ce qui le rend intéressant. Les chercheurs qui ont travaillé sur le VRPPC [5, 6, 14, 16, 25, 32, 45, 54] l'ont abordé avec des techniques comme la recherche tabou, les algorithmes génétiques, les algorithmes évolutionnaires, la relaxation lagrangienne etc... Nous allons résoudre ce problème au moyen de la métaheuristique de recherche adaptative à grand voisinage ALNS (pour Adaptative Large Neighborhood Search) qui a permis d'avoir des résultats intéressants pour plusieurs problèmes en optimisation combinatoire. Notre objectif

n'est pas de trouver la solution optimale pour notre problème, mais de trouver une bonne solution pouvant améliorer les résultats connus sur les instances de Christofides et celles de Golden [6].

1.2 La métaheuristique ALNS

Nous donnons ici quelques définitions qui vont nous permettre de mieux appréhender cette métaheuristique.

Définition 1.2. *[44] Soit x une solution courante. Une solution partiellement détruite est une copie de x, obtenue en supprimant certains de ses éléments.*

Définition 1.3. *[44] Soit x une solution courante. Un opérateur de destruction est une fonction qui, agissant sur x, retourne x_d, une solution partiellement détruite.*

Définition 1.4. *[44] Un opérateur de reconstruction est une fonction qui agit sur une solution partiellement détruite, et la répare. Il réinsère les éléments supprimés par l'opérateur de destruction dans la solution partiellement détruite. Il retourne donc une solution réalisable construite à partir d'une solution partiellement détruite.*

Définition 1.5. *[44] Soit x une solution courante. Une solution temporaire x^t est une solution réalisable obtenue par application de la composée d'un opérateur de destruction et de reconstruction sur x. Moyennant certaines hypothèses, la solution temporaire x^t pourrait être promue au statut de solution courante.*

La philosophie qui sous-tend la métaheuristique ALNS est similaire au processus de la sélection naturelle. Elle part de plusieurs opérateurs de destruction et de reconstruction à qui on donne au départ les mêmes probabilités d'action. Par la suite, selon les performances réalisées par chacun d'eux en cours de résolution, leur probabilité d'action et de survie est mise à jour. Plusieurs versions de cette métaheuristique ont été présentées par S. Ropke et D. Pisinger. Ici nous retiendrons celle rapportée dans [48]. Nous reviendrons dans la section 4.3 (chapitre 4) du mémoire sur cette métaheuristique.

1.3 Organisation

Le mémoire est organisé de la manière suivante. Dans le chapitre 2, nous passons en revue tout ce qui a été fait au niveau du VRPPC depuis le premier article de Volgenant et Jonker en 1987 jusqu'à présent. Nous expliquons comment le problème du voyageur de commerce (appelé en anglais Traveling Salesman Problem ou TSP), qui occupe une place centrale en optimisation combinatoire depuis la seconde moitié du vingtième siècle, peut être vu comme un VRPPC à un seul véhicule. Dans le chapitre 3, nous donnons la formulation mathématique du VRPPC. Deux modélisations mathématiques sont présentées: la première est de Chu [11] et la seconde de M.-C. Bolduc et al. [6]. Dans le chapitre 4, nous décrivons l'algorithme ALNS utilisé pour résoudre le VRPPC. Le chapitre 5 est consacré à la présentation des résultats numériques. D'abord nous expliquons comment les instances tests ont été conçues [6, 10, 24]. Nous rapportons ensuite nos résultats sur les instances avec flotte hétérogène. Le chapitre 6 présente la conclusion de notre travail et suggère de nouvelles perspectives.

Chapitre 2

Revue de littérature sur le VRPPC

Dans ce chapitre, nous allons passer en revue le VRPPC. Nous allons distinguer deux variantes, soit celle où il n'y a qu'un seul véhicule dans la flotte privée et celle où il y en a plusieurs.

2.1 VRPPC à véhicule unique

Nous allons d'abord rappeler le problème du voyageur de commerce qui a occupé une position centrale en optimisation combinatoire. Il constitue un cas spécial du VRPPC à véhicule unique.

Définition 2.1. ***TSP*** *Étant donné un ensemble de n villes, de distances connues, le problème du voyageur de commerce consiste à trouver un cycle de longueur minimale passant exactement une fois par chaque ville et qui revient au point de départ.*

Définition 2.2. *Soit un ensemble de n villes. La pénalité d'une ville est le prix payé par le voyageur s'il ne passe pas par cette ville. La récompense d'une ville est le prix obtenu par le voyageur s'il passe par cette ville.*

En 1987 T. Volgenant et R. Jonker [60] ont présenté une version généralisée du TSP.

Définition 2.3. ***TSPG*** *Soient un ensemble de n villes, de distances connues, et de pénalités connues. Le coût d'un cycle est la somme de sa longueur et des pénalités des villes qui n'en font pas partie. Le problème du voyageur de commerce généralisé (TSPG en abrégé), consiste à trouver le cycle de coût minimal passant au plus une fois par chaque ville.*

Le TSPG est un VRPPC à un seul véhicule. De même le TSP est un cas particulier du TSPG avec des pénalités assez larges. Plusieurs problèmes d'optimisation combinatoire sont des cas spéciaux du TSPG.

Définition 2.4. *Soit le TSPG avec n sommets. On dit qu'un sommet est spécifié, si on impose une visite obligatoire du voyageur à ce sommet. Le TSPG est symétrique, si le coût du déplacement pour chaque paire de sommets ne dépend pas de l'orientation.*

T. Volgenant et R. Jonker [60] ont montré le résultat fondamental suivant:

Théorème 2.1. *[60] Soit le TSPG avec n sommets et a arêtes. Il existe une transformation faisant du TSPG un TSP avec* $2n + 1$ *sommets et* $a + 3n - 1$ *arêtes.*

Remarque 2.1.

1. *Cette croissance dans la taille du problème implique que la transformation mise au point dans le théorème 2.1 est difficilement utilisable pour la résolution du TSPG standard. Cependant pour des variantes du TSPG où certains sommets sont spécifiés, la transformation ci-dessus peut être pratique. En effet il n'y a pas lieu de dupliquer les sommets spécifiés, vu qu'une visite requise à un sommet correspond à une large pénalité s'il n'y a pas de visite.*

2. *Si le TSPG original est symétrique, le TSP correspondant sera asymétrique pour les sommets additionnels. Un TSP symétrique peut être obtenu en utlisant une autre transformation de R. Jonker et T. Volgenant [30] pour des TSP partiellement symétriques. Cette transformation consiste à dupliquer chaque sommet de distance asymétrique et à ajouter une arête entre de tels sommets et leurs duplicatas. L'application de cette nouvelle transformation double le nombre de sommets pour lesquels l'asymétrie apparaît; on obtient par ce procédé un STSP (Symmetric Traveling Salesman Problem) ayant trois fois la taille originale du TSPG.*

Définition 2.5. *Soient un ensemble de n villes et de coûts de déplacement connus. On dit que les coûts de déplacement sont soumis aux inégalités triangulaires, si pour deux villes quelconques, il est toujours moins coûteux de se déplacer directement qu'en passant par une troisième.*

En 1989 E. Balas [2] a présenté un cas particulier du TSPG avec des contraintes additionnelles, une substitution des distances entre les villes par des coûts de déplacement non nécessairement soumis aux inégalités triangulaires.

Définition 2.6. ***PCTSP*** *[2] Soient un ensemble de n villes, de coûts de déplacement connus, de récompenses connues et de pénalités connues. Le coût d'un cycle est la somme de son coût de déplacement et des pénalités des villes qui n'en font pas partie. Le problème PCTSP (Prize Collecting Traveling Salesman Problem) consiste à trouver le cycle de coût minimal passant au plus une fois par chaque ville, permettant de recueillir au moins un montant prescrit de récompenses.*

En 1993, D. Bienstock et al. [4] ont étudié un cas particulier du PCTSP où les coûts de déplacement satisfont aux inégalités triangulaires et où, soit on s'affranchit des récompenses des villes, soit le montant prescrit des récompenses est nul. En assimilant les pénalités pour les villes non visitées aux coûts du transporteur externe, on obtient le VRPPC à véhicule unique. Les auteurs utilisent pour leur étude l'algorithme d'approximation MLP (Modified Linear Programming relaxation) qui est basé sur l'algorithme de Christofides pour le problème du voyageur de commerce [8] et une méthode pour arrondir les solutions fractionnaires d'un problème de programmation linéaire relaxé en entiers. Ils montrent que si Z^* est le coût de la solution optimale du PCTSP, et que Z_{pire}^{MLP} est le coût de la solution du PCTSP produite par MLP dans le pire cas, alors $Z_{pire}^{MLP}/Z^* \leq 2.5$.

En 1995, M. Diaby et R. Ramesh [15] ont étudié le VRPPC à un seul véhicule sous l'appellation DPCS (Distribution Problem with Carrier Service). Au moyen d'un algorithme optimal basé sur le branch-and-bound et certaines inégalités valides, ils ont résolu le VRPPC à un seul véhicule pour des instances contenant jusqu'à 200 sommets. Chaque arête comporte en plus de son coût, un temps de parcours. De plus une contrainte de temps maximal est imposée au véhicule de la flotte privée.

2.2 VRPPC à plusieurs véhicules

En 1983, M. O. Ball et al. [3] ont étudié un problème de collecte et livraison qui, si l'on considère une paire sommet collecte - sommet livraison comme un sommet, peut être vu comme l'ancêtre du VRPPC à plusieurs véhicules. Dans leur travail, la taille de la flotte privée consitue une variable du problème. De même, les véhicules de la flotte privée sont répartis dans exactement trois dépôts. Chaque paire collecte-livraison est affectée d'un coefficient entier qui représente le nombre de passages requis pour satisfaire ce sommet. Le

transporteur externe est utilisé pour amoindrir le coût d'un service d'une paire collecte-livraison. Les algorithmes utilisés par ces auteurs sont de deux types: la stratégie de type "route-first, cluster-second"et un algorithme glouton d'insertion.

En 1990, Klincewics et al. [34] ont étudié le VRPPC à plusieurs véhicules en utilisant une approche basée sur la distribution géographique des clients. L'ensemble des clients est partitionné en plusieurs secteurs. Pour chaque secteur, ils déterminent le véhicule de la flotte privée affecté à ce secteur, de même que les clients qui sont affectés au transporteur externe.

En 2005, C.-W. Chu [11] présente une modélisation mathématique du VRPPC. Il résout ce problème avec un algorithme heuristique en trois étapes. La première étape consiste à affecter à la flotte privée (dans la mesure de sa capacité) les clients ayant les plus grands coûts d'affectation au transporteur externe. Les clients restants sont affectés au transporteur externe. Une solution initiale est créée grâce à une modification de l'algorithme des gains de Clarke et Wright [12]. Par la suite, il applique une procédure rafffinée composée d'échanges intra-routes et extra-routes sur la solution initiale. Une fois une meilleure solution trouvée après la phase d'amélioration, la meilleure solution est mise à jour. Notons que les résultats expérimentaux sont présentés pour 5 instances (de très petite taille) avec flotte hétérogène de l'auteur, nommées par M.-C. Bolduc et al. [5] Chu1, Chu2, Chu3, Chu4 et Chu5.

En 2007, M.-C. Bolduc et al. [5] ont résolu le VRPPC à plusieurs véhicules avec une heuristique nommée SRI (Selection, Routing and Improvement). La procédure de sélection consiste à choisir les h premiers clients selon le ratio e_i/q_i (où e_i est le coût du transporteur externe du client i et q_i la demande du client i) et à les affecter au transporteur externe. Ici h est le plus petit entier tel que

$$\sum_{i=1}^{h-1} q_i < \sum_{i=1}^{n} q_i - \sum_{k=1}^{m} Q_k \leq \sum_{i=1}^{h} q_i \tag{2.1}$$

Notons que la première inégalité dans (2.1) n'est pas stricte dans les travaux des autres auteurs [5, 6, 14, 45], ce qui ne conduit pas à l'unicité de h. L'heuristique SRI crée deux solutions initiales dans un souci de diversification. Ces deux solutions initiales sont construites grâce à une version modifiée de l'heuristique de Clarke et Wright [12]. Les routes sont créées une à une, en nombre égal au nombre de véhicules de la flotte privée. Une

fois qu'une route est contruite, elle est affectée au véhicule de la flotte privée (non encore utilisé) de capacité minimale et suffisante pour le service. Au final, si toutes les routes construites ne visitent pas certains clients, alors ceux-ci sont affectés au transporteur externe. La procédure d'amélioration de la solution s'appuie sur une des versions de la procédure du λ-interchange proposée par Osman [42]. Une fois que la première solution initiale est créée, la procédure d'amélioration est appliquée. On crée ensuite une deuxième solution initiale, et on applique à nouveau la procédure d'amélioration. La meilleure solution obtenue est retenue comme la solution finale. En plus des cinq instances de Chu, cinq nouvelles instances que nous nommons BRB1, BRB2, BRB3, BRB4 et BRB5 sont introduites. Elles ont la même taille et la même flotte privée que celles de Chu. Les résultats obtenus sont meilleurs que ceux de Chu et sont optimaux pour les instances Chu1, Chu2, Chu3, Chu4, BRB1, BRB2 et BRB5.

En 2008, M.-C. Bolduc et al. [6] ont étudié le VRPPC avec flotte homogène et hétérogène en faisant appel à une métaheuristique nommée RIP (Randomized construction, Improvement and Pertubation). Le RIP est constitué de cinq procédures. La première est une heuristique aléatoire généralisant l'heuristique des gains de Clarke et Wright [12]. Dans cette heuristique, le gain est défini par $s_{ij} = d_{i0} + d_{0j} - \lambda_{ij} d_{ij}$ où 0 est le depôt, d_{ij} le coût de déplacement entre i et j, et λ_{ij} est choisi aléatoirement suivant une distribution uniforme dans un intervalle $[\underline{\lambda}, \overline{\lambda}]$. La deuxième procédure est une procédure d'échange de type 4-opt* [47] consistant à supprimer quatre arêtes appartenant à une même route et à insérer quatre nouvelles arêtes choisies sous certaines conditions. Elle est appliquée sur chaque route de la solution courante. La troisième procédure est une version restreinte du 2-interchange [42] opérant sur deux chaînes appartenant à deux routes différentes contenant au moins trois clients chacune. Aucune de ces deux chaînes ne doit contenir le dépôt. La quatrième procédure permet le transfert de clients de la flotte privée au transporteur externe et inversement. La dernière procédure correspond à une perturbation où, à chaque étape, un client est choisi aléatoirement. Si ce client est desservi par la flotte privée, il est échangé avec le voisin le plus proche dans les autres routes. S'il est externe, il est échangé avec le voisin desservi par la flotte privée le plus proche. Un procédé pour tester la réalisabilité de la solution après plusieurs permutations est mis au point. Les résultats obtenus sont

meilleurs que ceux obtenus par Chu et par l'heuristique SRI. Des résultats sont présentés sur de nouvelles instances: 14 instances avec flotte homogène de Christofides CE-01,. . . , CE-14, 20 instances avec flotte homogène de Golden G-01,. . . , G-20, 14 instances avec flotte hétérogène de Christofides CE-H-01,. . . , CE-H-14 et 20 instances avec flotte hétérogène de Golden G-H-01, . . . , G-H-20.

En 2009, J.-F. Côté et J.-Y. Potvin [14] ont résolu le VRPPC avec flotte homogène avec la recherche tabou (TS). La solution initiale est générée de façon similaire à celle de M.-C. Bolduc et al. [5]. Ils utilisent aussi deux structures de voisinage: Shift et Swap. Des résultats sont rapportés pour les 14 instances avec flotte homogène de Christofides et les 20 instances avec flotte homogène de Golden. Ils sont meilleurs que ceux obtenus par l'heuristique RIP. En 2011, J.-Y. Potvin et M.-A. Naud [45] en utilisant une recherche tabou améliorée (TS+), basée sur les chaînes d'éjection [22, 18] comme deuxième structure de voisinage, ont étendu les travaux de [14] aux instances avec flotte hétérogène de Christofides et Golden. La solution initiale générée au départ s'inspire de la phase de prétraitement de [5]. Elle utilise également les propriétés de dispersion des clients de même que la notion d'enveloppe convexe [47]. Sur toutes les instances avec flotte hétérogène de Christofides et de Golden, les résultats rapportés sont meilleurs que ceux de RIP. En ce qui concerne les instances avec flotte homogène, il n'y a que les 5 instances G-11, G-12, G-15, G-18 et G-19 où la recherche tabou TS [14] performe mieux que TS+.

K. Huang et C.-P. Hsu [25], ont rapporté en 2011 une nouvelle approche pour le VRPPC avec flotte homogène grâce à une transformation de ce dernier en SCP (Set Covering Problem). Ils ont développé une heuristique combinant les techniques de génération de colonnes (CG) et de relaxation lagrangienne pour résoudre le SCP correspondant. Les résultats numériques présentés sur les 14 instances avec flotte homogène de Christofides ne sont pas meilleurs que ceux de RIP, TS+, TS. Cependant, le temps d'exécution et la structure assez simple de leur algorithme sont remarquables. Cela satisfait la philosophie prônée par G. Laporte et F. Semet [37]. Toujours en 2011, J. Euchi, H. Chabchoub et A. Yassine [16] grâce à un algorithme évolutionnaire appelé IDEA ont obtenu des résultats meilleurs que ceux de TS+ pour les instances avec flotte homogène à l'exception de quatre instances: G-01, G-03,

G-06 et G-07. De même pour les instances avec flotte hétérogène de Christofides, IDEA performe mieux que TS+ sauf sur CE-H-05. Sur les 20 instances avec flotte hétérogène de Golden, TS+ performe mieux que IDEA pour les 8 instances: G-H-01, G-H-03, G-H-05, G-H-06, G-H-08, G-H-16, G-H-18 et G-H-20. Notons que les résultats de J. Euchi, H. Chabchoub et A. Yassine [16] bien qu'intéressants sont peu connus dans la communauté des chercheurs travaillant sur le VRPPC. En effet, les publications postérieures ne citent jamais leurs travaux dans leurs études expérimentales de nature comparative.

En 2012, le VRPPC avec flotte homogène et hétérogène a été étudié par J. Katrica et al. [32] sous une autre appellation, soit RCSP (Routing and Carrier Selection Problem). Leur étude est faite grâce à un algorithme génétique (GA). Dans leur implantation, chaque gène est représenté par la distance relative entre le client courant et les autres clients qui ne sont pas encore servis. Une matrice des distances est définie. Le code génétique de chaque individu est consitué de $n-1$ gènes, vu que le dépôt est représenté par un client qui n'a pas de demande. La procédure commence au dépôt avec le premier véhicule. On prend le gène qui correspond au véhicule courant et au client courant à partir du code génétique. Si ce gène a la valeur r, le $r+1$ ième client le plus proche non servi est choisi à partir de la matrice des distances. Ce client est ajouté à la route du véhicule courant si sa capacité le lui permet. Dans le cas contraire, le véhicule courant retourne au dépôt et le client courant est relié au dépôt; un nouveau véhicule est utilisé et la procédure recommence. De façon générale l'heuristique de Kratica et al. [32] GA, performe aussi bien que SRI sur les instances de Chu et de Bolduc. Sur Chu3 et BRB3, GA donne les valeurs optimales. Sur les instances avec flotte homogène et hétérogène de Christofides et Golden, GA est dominé par TS+.

En 2013, A. Stenger et al. [54] ont étudié le problème MDVRPPC (Multi-Depot Vehicle Routing Problem with Private Fleet and Common Carrier). Ce problème est une généralisation du VRPPC en ce sens qu'on n'a plus l'unicité du dépôt et du transporteur externe, mais un ensemble de dépôts $\mathcal{D}$ et plusieurs transporteurs externes. L'ensemble $\mathcal{D}$ est partitionné en deux sous-ensembles $\mathcal{D}_p$ et $\mathcal{D}_e$. Les véhicules de la flotte privée sont positionnés aux dépôts de $\mathcal{D}_p$ et, pour chaque dépôt leur nombre ne peut excéder k_{max}. A chaque dépôt de $\mathcal{D}_e$ est positionné un unique transporteur externe, celui-ci a un rayon de livraison ne

pouvant excéder r_{max}. De plus, chaque client requiert un temps de service, chaque route a une durée ne pouvant excéder t_{max} et, chaque dépôt $\mathfrak{d}$ de $\mathcal{D}$ dessert des clients pour lesquels la somme totale de leurs demandes n'excède pas $w_{\mathfrak{d}}$. Les auteurs ont résolu ce problème avec une heuristique AVNS (Adaptative Variable Neighborhood Search) qui est une extension de l'heuristique VNS (Variable Neighborhood Search) développée par N. Mladenovic et P. Hansen [40]. Les structures de voisinage utilisées sont basées sur des opérateurs d'échanges de sommets cycliques proposés par P. M. Thompson et J. B. Orlin [57]. Ils sont caractérisés par trois paramètres: le nombre de dépôts d'où proviennent les routes, le nombre de routes intervenant dans ces échanges cycliques (Ω) et la longueur maximale de la suite de sommets à échanger (Γ_{max}). Pour le MDVRPPC 51 échanges cycliques ont été utilisés avec $\Gamma_{max} \leq 10$, $2 \leq \Omega \leq 4$. La procédure de recherche locale est composée de:
– deux échanges intra-route (2-Opt et une version restreinte du Or-Opt [41])
– deux échanges inter-route (un opérateur swap et un λ-interchange).
La phase de prétraitement pour affecter les h premiers clients aux transporteurs externes est identique à celle de [5]. Ici encore, on doit imposer que la seconde inégalité de l'équation (15) de [54] soit une inégalité stricte pour que h soit unique. Les auteurs ont appliqué leur méthode au cas particulier du VRPPC. Sur les instances avec flotte homogène de Christofides CE-01, ..., CE-14 et celles de Golden G-01, ..., G-08, AVNS performe mieux que RIP, TS, et TS+. Sur les instances avec flotte homogène de Golden G-09,..., G-20, TS+ performe mieux que AVNS.

En 2014 T. Vidal et al. [59] ont étudié le 2-VRPP (2-resources Vehicle Routing Problem with Profits). Dans ce problème, on a une flotte de m véhicules identiques basée à un dépôt. À chaque arc (i, j) est associée une ressource de consommation $r_{ij} \in \mathbb{R}^+$ et un profit $p_{ij} \in \mathbb{R}$. Ce problème consiste à construire m routes ou moins, maximisant le profit total tout en respectant les contraintes de ressources sur chacune des routes. T. Vidal et al. [59] ont montré que le TOP (Team Orienteering Problem) [1, 7], le CPTP (Capacited Profitable Tour Problem) [7] et le VRPPC (noté VRPPFCC) sont tous des cas particuliers du 2-VRPP. Pour la résolution du 2-VRPP il y a 3 étapes. La première étape consiste à construire une solution complète. Dans une telle solution, tous les clients sont affectés aux véhicules et le séquençage est fait de façon aléatoire. Il peut y avoir des routes pour

lesquelles les contraintes de ressources ne sont pas satisfaites, et des routes satisfaisant ces contraintes mais de mauvaise qualité. La deuxième étape consiste à produire une vraie solution à partir de la solution complète. Ici un algorithme de sélection (SELECT algorithm) basé sur le plus court chemin au sens des contraintes de ressource est appliqué sur chaque route. SELECT récupère la sous-suite de clients satisfaisant les contraintes de ressources qui maximise les profits. Ce problème est résolu grâce à la programmation dynamique [29]. La troisième étape consiste à utiliser une procédure de recherche locale considérant des structures de voisinage larges avec un nombre exponentiel d'insertions et de retraits des clients. Ces structures de voisinage sont incorporées dans trois métaheuristiques MS-LS (Multi-Start Local Search), MS-ILS (Multi-Start Iterated Local Search) et le UHGS (Unified Hybrid Genetic Search). En ce qui concerne le VRPPC, la performance de UHGS est généralement supérieure à celle de MS-LS, de MS-ILS, et de AVNS sur les instances avec flotte homogène de Christofides et de Golden. T. Vidal et al. [59] prétendent que UHGS fournit un grand nombre de meilleures solutions connues (27/34) incluant 20 nouvelles solutions, mais aucune comparaison n'est établie avec les résultats de IDEA [16].

Toujours en 2014, S. Huijink et al. [26] ont abordé le VRPPC avec flotte homogène en utilisant la recherche tabou. Ils utilisent une phase d'initialisation très particulière. Ils ne supposent pas a priori que tous les véhicules sont utilisés. Le nombre de véhicules utilisés est déterminé par le choix d'un entier aléatoire compris entre 0 et m. La phase d'initialisation comporte trois étapes:

– La première étape consiste à déterminer la tête de série *seed* du premier véhicule. La tête de série est le client i pour lequel le ratio $\frac{e_i - d_{0,i}}{q_i}$ est le plus grand, où $d_{0,i}$ est la distance du client i au dépôt 0.

– La deuxième étape consiste à remplir la route en choisissant le client pour lequel $\frac{e_i - d_{\text{seed},i}}{q_i}$ est le plus grand; on continue le processus jusqu'à ce qu'on atteigne la moitié de la capacité du véhicule.

– La troisième étape consiste à insérer les clients restants dans les routes construites si c'est possible. L'insertion se fait dans une route le permettant et pour le client i pour lequel $\frac{e_i - \text{Insertion}}{q_i}$ est le plus grand et positif, où Insertion est le coût d'insertion classique; ce processus s'arrête s'il n'y a plus d'insertion réalisable ou profitable. Tous les clients restants

sont affectés au transporteur externe.

Dans la recherche tabou, une nouvelle technique nommée reseeding permet d'explorer certaines parties non-encore atteintes de l'espace de recherche. Le reseeding consiste d'abord à partitionner l'ensemble des routes d'une solution en deux sous-ensembles distincts (DT et NDT) selon que le coût total des opérations du véhicule correspondant est strictement plus petit ou plus grand que la somme des coûts d'affectation au transporteur externe des clients servis par ce véhicule. Ensuite, tous les clients appartenant aux routes NDT sont mis dans la banque de clients à insérer; pour les routes DT, on conserve dans celles-ci un pourcentage assez grand de clients pour que la moitié de la capacité du véhicule soit occupée. Les autres clients des routes NDT sont mis dans la banque de clients à insérer. Finalement tous les clients de cette banque sont insérés au moyen de la technique utilisée dans la troisième étape de la phase d'initialisation. La recherche tabou utilisée par S. Huijink et al. [26] est baptisée R-TS. Sur les instances avec flotte homogène de Christofides et de Golden, UHGS et R-TS ont la même performance pour CE-01, CE-02, CE-06 et CE-11. Sur les autres instances avec flotte homogène de Christofides et de Golden UHGS performe mieux que R-TS sauf sur CE-07, G-10, G-11 et G-12.

Toujours en 2014, après la publication de leur article [26], Huijink et al. [27] ont réétudié le VRPPC avec flotte homogène et hétérogène au moyen deux nouvelles métaheuristiques AVNS-Fast et AVNS-Slow. Contrairement à la métaheuristique AVNS présentée dans [54] qui utilise un seul type de voisinage, AVNS-Fast et AVNS-Slow utilisent plusieurs structures de voisinages complètement différentes:

– Deux variantes des échanges cycliques nommées CyclicDriven et CyclicNotDriven pour lesquelles différentes méthodes de sélection des véhicules et des clients sont définies

– Les sauts (Jumps), s'appuient sur la partition des routes faite dans [26]. Ils permettent d'abord de supprimer aléatoirement un premier ensemble de clients des routes DT, de sélectionner aléatoirement un autre ensemble de clients des routes NDT, pour lesquels le coût total d'affectation au transporteur externe est supérieur ou égal à celui du premier ensemble et d'insérer ces clients dans les routes DT.

– La division (Split), consiste à diviser en deux la route d'un véhicule. La division est faite en se référant à la plus grande distance entre deux clients consécutifs appartenant à

la route.

– La bombe (Bomb), choisit aléatoirement un client comme centre et détruit un certain nombre de clients dans une région autour de ce centre.

– La création (Create), permet de créer une nouvelle route à partir des routes NDT. La tête de série de la nouvelle route est le client i des routes NDT pour lequel la valeur $e_i - d_{0,i}$ est la plus grande. On remplit la nouvelle route des clients des routes NDT en suivant un processus similaire à celui de la deuxième étape de la phase d'initialisation présentée plus haut.

– La destruction (Destroy), consiste à choisir aléatoirement une route et à la détruire. Les clients appartenant à cette route sont mis dans la banque de clients à insérer

– Le réensemencement (Reseeding), consiste d'abord à détruire toutes les routes NDT et à conserver dans chaque route DT un certain pourcentage des clients i ayant le plus grand ratio $\frac{e_i - d_{0,i}}{Q_k}$. Tous les clients supprimés sont mis dans une banque. Ensuite on utilise la méthode d'insertion explicitée dans la troisième étape de la phase d'initialisation pour insérer les clients de la banque.

Sur les instances avec flotte homogène de Christofides et de Golden, de façon générale UHGS et AVNS-Slow performent mieux que AVNS-Fast. De même AVNS-Slow domine UGHS sauf sur 8 instances: CE-04, CE-05, G-05, G-13, G-14, G-15, G-16 et G-18.

Devant la multitude des méthodes présentées ci-dessus et des résultats obtenus sur les instances avec flottes homogènes et hétérogènes de Christofides et de Golden, nous retenons qu'il n'y a pas vraiment un algorithme qui soit le meilleur sur toutes les instances. L'agorithme qui s'en rapproche le plus est IDEA [16], mais il est peu connu et peu cité.

L'algorithme ALNS de par sa structure se distingue de la recherche tabou (TS, TS+) et des algorithmes évolutionnaires (GA, IDEA). Elle ressemble davantage à la métaheuristique AVNS et ses dérivées (AVNS-Slow et AVNS-Fast) avec pour particularité l'usage de grands voisinages. Notre travail consiste maintenant à utiliser l'algorithme ALNS dans l'espoir d'obtenir des résultats qui pourront être compétitifs, en particulier sur les instances avec flotte hétérogène de Christofides et de Golden [6] qui ont été moins étudiées que les instances avec flotte homogène.

Chapitre 3

Modélisation mathématique du VRPPC

La problématique du VRPPC a été mentionnée dès 1987, cependant ce n'est qu'en 2005 qu'un modèle mathématique a été proposé par C.-W. Chu [11]. Par la suite un deuxième modèle plus raffiné a été proposé en 2008 par M.-C. Bolduc et al. [6]. Dans ce chapitre nous présenterons chacun de ces modèles.

La structure mathématique appropriée pour décrire le VRPPC repose sur la notion de graphe. Soit $G = (\tilde{V}, A)$ un graphe orienté complet sur l'ensemble des sommets $\tilde{V} = \{0, 1, \ldots, n\}$ et A l'ensemble des arcs. Le sommet 0 représente le dépôt pendant que les autres sommets représentent les n clients; on désigne par $V = \{1, \ldots, n\}$ l'ensemble des clients. On définit sur A une matrice des coûts $c = (c_{ij})$, où c_{ij} est le coût du déplacement du sommet i au sommet j, qui correspond habituellement à la distance entre ces sommets. Si les coûts de déplacement dépendent du véhicule, alors c_{ij} sera remplacé par c_{ijk}, où $k \in \{1, \ldots, m\}$. Dans notre cas, les coûts sont les mêmes pour tous les véhicules.

3.1 Modèle de Chu

Il y a une grande similitude entre les variables et les équations définissant le modèle de T. Volgenant et R. Jonker [60] pour le TSPG et celles de C.-W. Chu [11] pour le VRPPC, quoique le second ne fasse pas référence au premier. La raison de cette similitude est que le

VRPPC à plusieurs véhicules est une extension du TSPG à plusieurs voyageurs, où les cycles définis par ces voyageurs ont le même point de départ et aucun autre point en commun.

Dans cette section, nous allons d'abord définir les variables permettant de modéliser le VRPPC et ensuite donner le modèle de programmation linéaire tel que conçu par C.-W. Chu [11]. Chu a défini les trois types de variables binaires dans le but de modéliser le VRPPC:

$$x_{ijk} = \begin{cases} 1, & \text{si le véhicule } k \text{ se déplace de } i \text{ à } j \\ 0, & \text{sinon} \end{cases}$$

$$z_i = \begin{cases} 1, & \text{si le client } i \text{ est affecté au transporteur externe} \\ 0, & \text{sinon} \end{cases}$$

$$y_{ik} = \begin{cases} 1, & \text{si le client } i \text{ est visité par le véhicule } k \\ 0, & \text{sinon} \end{cases}$$

Notons que $y_{0k} = 1$, signifie que le véhicule k est mis en service.

Voici maintenant le modèle de programmation linéaire en nombres entiers formulé par C.-W.

Chu pour le VRPPC:

$$\min \sum_{k=1}^{m} f_k + \sum_{i=0}^{n} \sum_{\substack{j=0 \\ j \neq i}}^{n} \sum_{k=1}^{m} c_{ijk} x_{ijk} + \sum_{i=1}^{n} e_i z_i \tag{3.1}$$

$$\sum_{k=1}^{m} y_{0k} = m \tag{3.2}$$

$$\sum_{k=1}^{m} y_{ik} + z_i = 1, \quad i = 1, \ldots, n \tag{3.3}$$

$$\sum_{i=1}^{n} q_i y_{ik} \leq Q_k, \quad k = 1, \ldots, m \tag{3.4}$$

$$\sum_{\substack{j=0 \\ j \neq i}}^{n} x_{ijk} = y_{ik}, \quad i = 0, \ldots, n; \quad k = 1, \ldots, m \tag{3.5}$$

$$\sum_{\substack{j=0 \\ j \neq i}}^{n} x_{jik} = y_{ik}, \quad i = 0, \ldots, n; \quad k = 1, \ldots, m \tag{3.6}$$

$$\sum_{i \in S} \sum_{\substack{j \in S \\ j \neq i}} x_{ijk} \leq |S| - 1, \quad \forall \quad S \subset \{1, \ldots, n\}$$

$$i = 1, \ldots, n; \quad k = 1, \ldots, m \tag{3.7}$$

$$x_{ijk} \in \{0, 1\}, \quad i, j \in \{0, \ldots, n\} \quad i \neq j; \quad k = 1, \ldots, m \tag{3.8}$$

$$y_{ik} \in \{0, 1\}, \quad i = 0, \ldots, n; \quad k = 1, \ldots, m \tag{3.9}$$

$$z_i \in \{0, 1\}, \quad i = 1, \ldots, n \tag{3.10}$$

Dans ce modèle, l'équation (3.1) est la fonction objectif. La contrainte (3.2) assure que tous les véhicules sont utilisés pour servir les clients. En réalité, cette condition est trop restrictive. La contrainte (3.3) assure que chaque client est servi par un véhicule de la flotte privée ou par le transporteur externe. La contrainte (3.4) est la contrainte de capacité: aucun véhicule ne peut être chargé au-delà de sa capacité maximale. Les contraintes (3.5) et (3.6) assurent qu'un véhicule qui arrive chez un client repart de ce client. La contrainte (3.7) est la contrainte d'élimination des sous-tours.

3.2 Modèle BRBL (Bolduc Renaud Boctor et Laporte)

Le modèle BRBL correspond à celui proposé par M.-C. Bolduc et al. [6]. Ce modèle est une version raffinée du modèle de C.-W. Chu. En effet, Chu impose l'utilisation de tous

les véhicules de la flotte privée. Le modèle BRBL s'affranchit de cette condition. Il y a maintenant au plus m véhicules de la flotte privée qui sont utilisés.

Le modèle BRBL utilise les mêmes variables que celles du modèle de Chu en plus d'une variable réelle u_{ik} qui est une borne supérieure sur la charge du véhicule k dès qu'il quitte le client i. La fonction objectif devient alors

$$\sum_{k=1}^{m} f_k y_{0k} + \sum_{i=0}^{n} \sum_{\substack{j=0 \\ j\neq i}}^{n} \sum_{k=1}^{m} c_{ijk} x_{ijk} + \sum_{i=1}^{n} e_i z_i \tag{3.11}$$

De même, la contrainte (3.2) de Chu est modifiée et donne,

$$\sum_{j=1}^{n} \sum_{k=1}^{m} x_{0jk} = \sum_{i=1}^{n} \sum_{k=1}^{m} x_{i0k} \leq m \tag{3.12}$$

Les contraintes (3.3), (3.4), (3.5) et (3.6) ne changent pas. Dans ce modèle, la contrainte (3.7) d'élimination des sous-tours non connectés au dépôt de Chu est remplacée par celle de Kara et al. [31],

$$\begin{aligned} u_{ik} - u_{jk} + Q_k x_{ijk} + (Q_k - q_i - q_j) x_{jik} &\leq Q_k - q_j, \\ i, j \in \{1, \ldots, n\} \quad i \neq j; \quad k = 1, \ldots, m & \end{aligned} \tag{3.13}$$

où u_{ik} est une borne supérieure sur la charge du véhicule k après avoir servi le client i. Cette contrainte est en fait une extension de la contrainte de Kulkarni et Bhave [35] en ce sens qu'elle utilise tous les termes de celle-ci en plus de l'ajout du terme $(Q_k - q_i - q_j)x_{jik}$.

Remarque 3.1. *1. Les variables z_i peuvent être ignorées par l'introduction d'un véhicule fictif 0 de capacité $Q_0 = \sum_{i=1}^{n} q_i$ utilisé pour visiter tous les clients affectés au transporteur externe. L'ajout du véhicule fictif suggère une nouvelle formulation de la contrainte (3.3) de Chu et une nouvelle définition de la matrice des coûts $\tilde{c} = (\tilde{c}_{ijk})$ où*

$$\tilde{c}_{ijk} = \begin{cases} c_{ijk} + f_k & i = 0; j = 1, \ldots, n; k = 1, \ldots, m \\ c_{ijk} & j = 0; i = 1, \ldots, n; k = 1, \ldots, m \\ e_j & i = 0, \ldots, n; j = 1, \ldots, n; k = 0 \\ 0 & i = 1, \ldots, n; j = 0; k = 0 \end{cases}$$

2. Les variables y_{ik} peuvent être ignorées. En effet, on a:

$$\sum_{\substack{j=0\\ i\neq j}}^{n} x_{ijk} = \sum_{\substack{j=0\\ j\neq i}}^{n} x_{jik} = y_{ik}, \quad i = 0, \ldots, n; \quad k = 1, \ldots, m \tag{3.14}$$

Avec cette nouvelle fonction objectif (3.11), les nouvelles contraintes et la remarque ci-dessus, on obtient un modèle de programmation linéaire mixte pour le VRPPC avec $m+1$ véhicules:

$$\min \sum_{i=0}^{n} \sum_{\substack{j=0\\ j\neq i}}^{n} \sum_{k=0}^{m} \tilde{c}_{ijk} x_{ijk} \tag{3.15}$$

$$\sum_{j=1}^{n} \sum_{k=0}^{m} x_{0jk} = \sum_{i=1}^{n} \sum_{k=0}^{m} x_{i0k} \leq m+1 \tag{3.16}$$

$$\sum_{k=0}^{m} \sum_{\substack{j=1\\ j\neq i}}^{n} x_{ijk} = 1, \quad i = 1, \ldots, n \tag{3.17}$$

$$\sum_{i=1}^{n} \sum_{\substack{j=1\\ j\neq i}}^{n} q_i x_{ijk} \leq Q_k, \quad k = 0, \ldots, m \tag{3.18}$$

$$\sum_{\substack{j=0\\ j\neq h}}^{n} x_{hjk} = \sum_{\substack{i=0\\ i\neq h}}^{n} x_{ihk}, \quad h = 0, \ldots, n; \quad k = 0, \ldots, m \tag{3.19}$$

$$u_{ik} - u_{jk} + Q_k x_{ijk} + (Q_k - q_i - q_j) x_{jik} \leq Q_k - q_j,$$
$$i, j \in \{1, \ldots, n\} \quad i \neq j; \quad k = 0, \ldots, m \tag{3.20}$$

$$x_{ijk} \in \{0, 1\}, \quad i, j \in \{0, \ldots, n\} \quad i \neq j; \quad k = 0, \ldots, m \tag{3.21}$$

$$u_{ik} \geq 0, \quad i = 1, \ldots, n; \quad k = 0, \ldots, m \tag{3.22}$$

La première formulation mathématique a été utilisée par Chu pour résoudre le VRPPC sur les instances Chu1, Chu2, Chu3, Chu4 et Chu5. Elle a été par la suite utilisée par Bolduc et al. afin d'obtenir les véritables solutions optimales pour chacune de ces instances. En effet, l'implantation de Chu comportait une erreur et a produit des résultats erronés sur les instances Chu3, Chu4 et Chu5. De même Bolduc et al. ont utilisé ce modèle pour déterminer des solutions optimales sur les instances BRB1, BRB2, BRB3, BRB4 et BRB5, permettant ainsi de mesurer la performance de l'heuristique de Chu et de l'heuristique SRI sur ces très petites instances.

Le modèle mathématique BRBL est très sophistiqué. En raison de la trop grande complexité des deux familles d'instances (Christofides et Eilon, Golden), la résolution du VRPPC à travers l'aide du modèle mathématique est impossible. En effet:

- la relation (3.16) fournit une inéquation linéaire et une équation linéaire dont les variables sont x_{ijk}.
- les relations (3.17), (3.18) et (3.19) fournissent respectivement n, m+1 et (m+1)(n+1) équations linéaires dont les variables sont x_{ijk}.
- la relation (3.20) fourni $(m+1)n$ inéquations linéaires dont les variables sont x_{ijk} et u_{ik}.
- la relation (3.22) fourni $(m+1)n$ inéquations linéaires dont les variables sont u_{ik}.

On doit minimiser la fonction objectif sur un système de $(n+2)(m+1)+n+1$ équations linéaires et $2n(m+1)+1$ inéquations linéaires comportant $n(n+2)(m+1)$ variables dont $n(n+1)(m+1)$ variables binaires x_{ijk}. Rien que pour la plus petite instance avec flotte hétérogène de Christofides et Eilon CE-H-01 constituée de 50 clients et dont la flotte privée hétérogène ne contient que 4 véhicules, nous avons à résoudre un système de programmation linéaire mixte de 812 équations et inéquations (311 équations linéaires et 501 inéquations linéaires) à 13000 inconnues (12750 variables binaires x_{ijk} et 250 variables réelles positives u_{ik}). Ceci excède les capacités des solveurs actuellement disponibles si l'on veut obtenir des solutions en un temps raisonable. Nous ferons donc appel à notre métaheuristique ALNS qui, permet d'obtenir de bonnes solutions pour le VRPPC en des temps relativement rapides sur des instances de taille conséquente.

Chapitre 4

Description de l'algorithme ALNS pour le VRPPC

Ce chapitre qui constitue la partie centrale de notre travail, décrit toute l'architecture de l'algorithme ALNS. Il est organisé de la manière suivante. Dans la section 4.1, nous explicitons la représentation utilisée pour les clients et les routes. Dans la section 4.2, nous définissons un certain nombre de mesures utiles par rapport à notre problème. Dans la section 4.3, nous donnons la description générale de notre ALNS. Dans la section 4.4, nous expliquons le processus de la création de la solution initiale. Les sections 4.5 et 4.6 sont consacrées respectivement à la description des opérateurs de destruction et de reconstruction. Enfin, on étudie dans les deux dernières sections, les critères d'acceptation d'une nouvelle solution et d'arrêt de l'algorithme, et la mise à jour adaptative des poids des opérateurs.

4.1 Représentation des clients et des routes

Dans cette section, nous donnons la représentation utilisée pour modéliser les clients et les routes. Cette représentation découle de la modélisation mathématique précédente du VRPPC. Elle est fondamentale pour tout ce chapitre.

Un client i est un 6-uplet formé des attributs suivants:

- *code*: numéro du client i

- x_i : abscisse du client i
- y_i : ordonnée du client i
- q_i : demande du client i
- e_i : coût du transporteur externe affecté à ce client
- z_i : variable de type booléen prenant la valeur 1 si le client i est desservi par le transporteur externe et 0 s'il est desservi par un véhicule régulier.

Une route correspond à une séquence de clients. Elle dépend principalement du véhicule qui fait le parcours (véhicule associé k) et est représentée par un 8-uplet formé des attributs suivants:

- $numero$: numéro du véhicule associé k
- $trajet$: liste des clients de la route
- $first$: client de départ
- $last$: client d'arrivée, soit le dernier client visité avant de retourner au dépôt,
- $card(k)$: nombre de clients dans la route du véhicule associé k
- Q_k: capacité maximale du véhicule associé k
- $chargeVeh_k$: charge résiduelle courante du véhicule associé k
- f_k: coût fixe du véhicule associé k

4.2 Les invariants

Notre approche de résolution du VRPPC passe par la détermination de quatre mesures qui joueront un rôle important par la suite. Il s'agit de la charge excédentaire, de l'indice critique, de la distance maximale et de l'écart maximal. Nous donnons ici une définition de chacun de ces termes et l'algorithme pour calculer les trois derniers. L'algorithme permettant de calculer la charge excédentaire étant trivial.

4.2.1 La charge excédentaire et l'indice critique

Définition 4.1. *On appelle charge excédentaire du VRPPC notée E, la différence entre la somme totale des demandes des clients et la capacité totale de tous les véhicules. On a,*

$$E = \sum_{i=1}^{n} q_i - \sum_{k=1}^{m} Q_k.$$

Définition 4.2. *On suppose que les clients sont triés en ordre croissant en fonction de leur demande q_i. On appelle indice critique du VRPPC, le plus petit entier h tel que si les h premiers clients sont affectés au transporteur externe, la capacité totale de la flotte privée devient suffisante pour desservir les $n - h$ autres clients. On a la relation suivante:*

$$\sum_{i=1}^{h-1} q_i < E \leq \sum_{i=1}^{h} q_i$$

Soient V l'ensemble de tous les clients du problème et E la charge excédentaire, l'algorithme ci-dessous permet de déterminer l'indice critique h.

Pseudo-Code 4.1. *Algorithme de l'indice critique*
Fonction *attribut_h_indice_critique(V, E)*

1. *somPartiel =0, h=0;*
2. *Pour tout client $i \in V$ faire*
   (a) *somPartiel = somPartiel + q_i;*
   (b) *h = h +1;*
   (c) *Si somPartiel $\geq E$, Arrêter;*
3. *Retourner h;*

Remarque 4.1. *Notons que la notion d'indice critique a été exploitée dans plusieurs travaux antérieurs au nôtre [6, 14, 45, 54], mais notre définition est beaucoup plus précise à cause de la présence d'une inégalité stricte entre les deux premiers termes dans la définition 4.2. On peut le voir dans l'exemple ci-dessous.*

Exemple 4.1. *Considérons le problème VRPPC où la capacité totale de tous les véhicules de la flotte privée est 38. On suppose qu'il y a exactement 7 clients de demande 1, 1, 2, 2, 8, 10, et 20. Ici $E = 6$ et $h = 4$ car $1 + 1 + 2 < 6 \leq 1 + 1 + 2 + 2$. Mais si nous utilisons la définition de h selon les travaux [6, 14, 45, 54], h pourrait aussi prendre la valeur 5, car $1 + 1 + 2 + 2 \leq 6 \leq 1 + 1 + 2 + 2 + 8$, ce qui permettrait d'affecter plus de clients qu'il n'en faut au transporteur externe.*

4.2.2 La distance maximale et l'écart maximal

Définition 4.3. *On appelle distance maximale du VRPPC, notée $d_{\max}$, la distance entre les deux clients les plus éloignés. On a,*

$$d_{\max} = \max_{\substack{i,j \in V \\ i \neq j}} d_{i,j}$$

Définition 4.4. *On appelle écart maximal du VRPPC, notée $g_{\max}$, la différence entre la demande la plus grande et la demande la plus petite. On a,*

$$g_{\max} = \max_{\substack{i,j \in V \\ i \neq j}} |q_i - q_j|$$

L'algorithme ci-dessous permet de déterminer la distance maximale. Celui de l'écart maximal s'écrit de la même manière.

Pseudo-Code 4.2. *Algorithme pour le calcul de la distance maximale*
Fonction *attribut_distance_maximale(V)*

1. *Initialisation de la distance maximale entre deux clients*
   $g_{\max} = -\infty$;
2. *Initialisation de W*
   $W = V$;
3. *Pour tout client $i \in V$ faire*
   (a) $W = W \setminus \{i\}$;
   (b) *Pour tout client $j \in W$ faire*
      i. *Calculer $d_{i,j}$;*
      ii. *Si $g_{\max} < d_{i,j}$ alors $g_{\max} = d_{i,j}$;*
4. *Retourner $g_{\max}$;*

4.3 Description générale de ALNS

L'algorithme ALNS demande d'abord la construction d'une solution initiale. Il fonctionne de la manière suivante. À chaque itération de l'algorithme, un opérateur de destruction et un opérateur de reconstruction sont choisis. L'opérateur de destruction retire un certain nombre de clients de la solution courante produisant ainsi une solution partielle. Les clients

retirés sont transmis à l'opérateur de reconstruction qui les réinsère dans la solution partiellement détruite. La nouvelle solution remplace alors la solution courante si un critère d'acceptation est satisfait. Enfin, si le coût de la nouvelle solution est meilleur que celui de la meilleure solution observée jusqu'à date, alors elle est retenue comme meilleure solution. Cette procédure est répétée jusqu'à ce que la condition d'arrêt soit satisfaite.

Dans la description ci-dessous sous forme pseudo-code, x désigne la solution courante, x^t est la solution temporaire, x^b est la meilleure solution observée durant la recherche, Ω^- et Ω^+ représentent respectivement l'ensemble des opérateurs de destruction et de reconstruction. Chacun des opérateurs de destruction et de reconstruction est affecté respectivement d'un poids $\rho^- \in \mathbb{R}^{|\Omega^-|}$, $\rho^+ \in \mathbb{R}^{|\Omega^+|}$. La fonction *accept* désigne le critère d'acceptation.

Pseudo-Code 4.3. *Pseudo-Code de l'ALNS*

1. *Construire une solution initiale* x_0
2. *Initialiser la solution courante et la meilleure solution*
   $x = x_0$; $x^b = x$;
3. *Initialiser les poids des opérateurs*
   $\rho^- = (1, 1, ..., 1)$; $\rho^+ = (1, 1, ..., 1)$
4. *Tant que la condition d'arrêt n'est pas respectée faire*
   (a) *Choisir un opérateur de destruction* $d \in \Omega^-$ *et un opérateur de reconstruction* $r \in \Omega^+$
   (b) $x^t = r(d(x))$;
   (c) *Si* $accept(x, x^t)$,
   $x = x^t$
   (d) *Si* $c(x^t) < c(x^b)$,
   $x^b = x^t$
   (e) *Mettre à jour* ρ^- *et* ρ^+.
5. *Retourner* x^b;

Nous allons maintenant décrire en détails chacune des composantes de cette métaheuristique.

4.4 Création de la solution initiale

La procédure de création de la solution initiale se divise en quatre phases, soit la phase de prétraitement, la phase d'affectation des clients aux véhicules de la flotte privée, la phase de construction des cycles des TSP correspondants et la phase de création des routes de la flotte privée.

4.4.1 La phase de prétraitement

Nous avons utilisé la phase de traitement élaborée par M.-C. Bolduc et al. [5]. Cette technique a été utilisée par la suite par Côté et Potvin [14] et Potvin et Naud [45]. Elle consiste à:

1. Trier les clients en ordre croissant selon le ratio $\frac{e_i}{q_i}$, pour $i = 1, \ldots, n$.
2. Déterminer l'indice critique h.
3. Affecter les h premiers clients au transporteur externe.

4.4.2 Affectation des clients aux véhicules de la flotte privée

La phase de prétraitement assure que la capacité totale de la flotte privée est suffisante pour satisfaire les demandes de tous les autres clients (i.e. les clients internes, ceux qui n'ont pas été affectés au transporteur externe). Les clients internes seront maintenant affectés aux véhicules suivant leur répartition géographique en faisant appel à un indice de dispersion et à l'enveloppe convexe.

L'indice de dispersion

Cette approche permet d'affecter une famille de clients géographiquement proches à un véhicule. Elle a été mise au point par Gendreau et al. [20].

Définition 4.5. *Soient R un sous-ensemble de V et $d_{i,j}$ la distance entre deux clients i et j de V. On appelle indice de dispersion de R, le réel $\Gamma(R)$ défini par:*

$$\Gamma(R) = \begin{cases} \frac{\sum_{i,j \in R} d_{i,j}}{|R|} & \text{si} \quad |R| > 0 \\ 0 & \text{si} \quad |R| = 0 \end{cases} \tag{4.1}$$

L'algorithme ci-dessous permet de calculer l'indice de dispersion d'un ensemble non vide de clients. Il correspond à la moyenne des distances entre deux clients quelconques de cet ensemble. Plus cet indice est bas, plus le rapprochement géographique des clients est important.

Pseudo-Code 4.4. *Algorithme pour le calcul de l'indice de dispersion d'un ensemble*
Fonction *indice_de_dispersion(R)*

1. *Initialisation de la somme*
   $somme = 0$*;*
2. *Initialisation de* W
   $W = R$*;*
3. *Pour tout client* $i \in R$ *faire*
   (a) $W = W \setminus \{i\}$ *;*
   (b) *Pour tout client* $j \in W$ *faire*
      i. *Calculer* $d_{i,j}$*;*
      ii. $somme = somme + d_{i,j}$*;*
4. $\Gamma(R) = somme/|R|$ *;*
5. *Retourner* $\Gamma(R)$*;*

Soit F_p l'ensemble des clients qui doivent être affectés à la flotte privée après la phase de prétraitement et F_e l'ensemble des clients qui doivent être affectés au transporteur externe. On dénote par V_k l'ensemble des clients affectés au véhicule k et par β le facteur de perturbation stochastique. L'algorithme ci-dessous permet de créer des routes où les clients sont répartis selon l'indice de dispersion.

Pseudo-Code 4.5. *Répartition des clients selon l'indice de dispersion*
Fonction *affectation_clients_vehicules(F_p)*

1. *Pour* k *=1,. . . , m faire* $V_k = \emptyset$*;*
2. *Pour tout client* $i \in F_p$ *faire*
   (a) $d_{\min} = +\infty$*;*
   (b) *pos = Faux;*
   (c) *Pour* k *=1,. . . , m faire*

i. *Calculer l'indice de dispersion probabiliste. On ajoute le client i aux clients desservis par le véhicule k.*

A. r = *1+rand(0,*β*);*

B. $d = r \times \Gamma(V_k)$*;*

ii. *Mettre à jour la charge du véhicule k*
$chargeCourante = q_i + chargeVeh_k$*;*

iii. *Si* $d < d_{\min}$ *ET* $chargeCourante \leq Q_k$

A. $chargePartielle = chargeCourante$*;*

B. $k^* = k$*;*

C. $d_{\min} = d$*;*

D. $pos = Vrai$*;*

(d) *Si* $pos == Vrai$

i. *Affecter le client i au véhicule choisi* k^*
$V_{k^*} = V_{k^*} \bigcup \{i\}$*;*

ii. *Modifier la charge du véhicule choisi* k^**.*
$chargeVeh_{k^*} = chargePartielle$*.*

iii. $F_p = F_p \setminus \{i\}$

Sinon

i. *Affecter le client i au transporteur externe*
$F_e = F_e \bigcup \{i\}$

ii. $F_p = F_p \setminus \{i\}$

Notons que $\Gamma(V_k)$ à l'étape 3.a de l'algorithme ci-dessus est calculée sur la base de la formule (4.1).

4.4.3 Construction d'une solution initiale

Pour chaque sous-ensemble de sommets affectés à un véhicule, on détermine son enveloppe convexe. On construit une solution du TSP correspondant en insérant les sommets restants dans cette enveloppe convexe.

1. Enveloppe Convexe

Définition 4.6. *Soit* $\mathcal{E}$ *un espace euclidien et* V *un sous-ensemble de* $\mathcal{E}$*. L'enveloppe convexe de* V *est le plus petit sous-ensemble convexe de* $\mathcal{E}$ *contenant* V*. En ce qui nous concerne,* V *est un ensemble de sommets.*

Il faut rappeler que l'enveloppe convexe d'un ensemble de sommets a été beaucoup utilisée en optimisation combinatoire. Un résultat majeur a été obtenu par M. M. Flood [17] en 1956.

Théorème 4.1. *[17] Soient V un sous-ensemble de sommets de $\mathcal{E}$ et $\mathcal{F}_{ecv}(V)$ les points à la frontière de l'enveloppe convexe de V. Tout TSP euclidien sur V a une solution optimale où ses sommets sont visités dans le même ordre que celui des sommets de $\mathcal{F}_{ecv}(V)$.*

On la retrouve dans plusieurs algorithmes de résolution du problème du voyageur de commerce, par exemple:
– l'algorithme d'insertion de W.R. Stewart [55]
– l'algorithme CCAO (Convex hull, Cheapest insertion, Angle selection and Or-opt) développé par B.L. Golden et al. [23]
– l'algorithme I^3(Initialization, Insertion, and Improvement) développé par J. Renaud et al. [47].
Elle a été également utilisée pour le VRPPC dans [45]. Ici pour chaque route obtenue à partir de l'indice de dispersion, l'enveloppe clockwise (approximation de l'enveloppe convexe) de l'ensemble des sommets appartenant à cette route est construite.

Soit $V = \{1, \ldots, n\}$ l'ensemble des sommets. Chaque sommet i est représenté par sa coordonnée cartésienne (x_i, y_i), réciproquement $P(x_i, y_i)$ correspond au sommet de cordonnées x_i et y_i. Soit V_k l'ensemble des sommets affectés au véhicule k selon l'indice de dispersion et H_k l'enveloppe clockwise de cet ensemble. L'algorithme suivant construit H_k:

Pseudo-Code 4.6. *Algorithme de l'enveloppe clockwise*
Fonction *enveloppe_clockwise(V_k)*

1. *Étape 0: Initialisation*
   $\overline{x} = -\infty$ *et* $H_k = \varnothing$
2. *Étape 1: Northernmost point to the East-North/East*
   (a) $R = \{i \in V_k \setminus H_k : x_i > \overline{x}\}$
   (b) *while* ($R \neq \varnothing$)
      i. $R = \{i \in V_k \setminus H_k : x_i > \overline{x}\}$
      ii. $y = \max\limits_{i \in R} y_i$ *et* $x = \min\limits_{\substack{i \in R \\ y_i == y}} x_i$

iii. $\overline{x} = x, \quad \overline{y} = y, \quad R = R \setminus \{P(\overline{x}, \overline{y}) \text{ et } \quad H_k = H_k \cup \{P(\overline{x}, \overline{y})\}$

3. *Étape 2: Easternmost point to the South-East/South*

   (a) $R = \{i \in V_k \setminus H_k : y_i < \overline{y}\}$

   (b) *while* ($R \neq \varnothing$)

      i. $R = \{i \in V_k \setminus H_k : y_i < \overline{y}\}$

      ii. $x = \max\limits_{i \in R} x_i$ *et* $y = \max\limits_{\substack{i \in R \\ x_i == x}} y_i$

      iii. $\overline{x} = x, \quad \overline{y} = y, \quad R = R \setminus \{P(\overline{x}, \overline{y}) \text{ et } \quad H_k = H_k \cup \{P(\overline{x}, \overline{y})\}$

4. *Étape 3: Southernmost point to West-South/West*

   (a) $R = \{i \in V_k \setminus H_k : x_i < \overline{x}\}$

   (b) *while* ($R \neq \varnothing$)

      i. $R = \{i \in V_k \setminus H_k : x_i < \overline{x}\}$

      ii. $y = \min\limits_{i \in R} y_i$ *et* $x = \max\limits_{\substack{i \in R \\ y_i == y}} x_i$

      iii. $\overline{x} = x, \quad \overline{y} = y, \quad R = R \setminus \{P(\overline{x}, \overline{y}) \text{ et } \quad H_k = H_k \cup \{P(\overline{x}, \overline{y})\}$

5. *Étape 4: Westernmost point to the North-South/West*

   (a) $R = \{i \in V_k \setminus H_k : y_i > \overline{y}\}$

   (b) *while* ($R \neq \varnothing$)

      i. $R = \{i \in V_k \setminus H_k : y_i > \overline{y}\}$

      ii. $x = \min\limits_{i \in R} x_i$ *et* $y = \max\limits_{\substack{i \in R \\ x_i == x}} y_i$

      iii. $\overline{x} = x, \quad \overline{y} = y, \quad R = R \setminus \{P(\overline{x}, \overline{y}) \text{ et } \quad H_k = H_k \cup \{P(\overline{x}, \overline{y})\}$

Exemple 4.2. *Considérons le problème TSP où les sommets à visiter sont :* $P_1(4.5, 6)$, $P_2(8, 4)$, $P_3(7, 2)$, $P_4(4.5, 1)$, $P_5(2, 2)$, $P_6(0.5, 5)$, $P_7(5.5, 4.5)$, $P_8(5.7, 2)$, $P_9(4.5, 3)$ *et* $P_{10}(1.5, 4)$. *La technique Clockwise commence avec* $\overline{x} = -\infty$ *et* $H_k = \varnothing$. *L'étape 1 de l'algorithme permet graduellement d'ajouter les sommets* P_1, P_7 *et* P_2 *dans* H_k. *L'étape 2 permet ensuite d'ajouter les sommets* P_3 *et* P_4. *L'étape 3 permet d'ajouter les sommets* P_5, P_{10} *et* P_6. *A l'étape 4,* $R = \varnothing$, *on ne peut ajouter les points* P_8 *et* P_9. *Ci-dessous l'enveloppe clockwise des 10 sommets ci-dessus.*

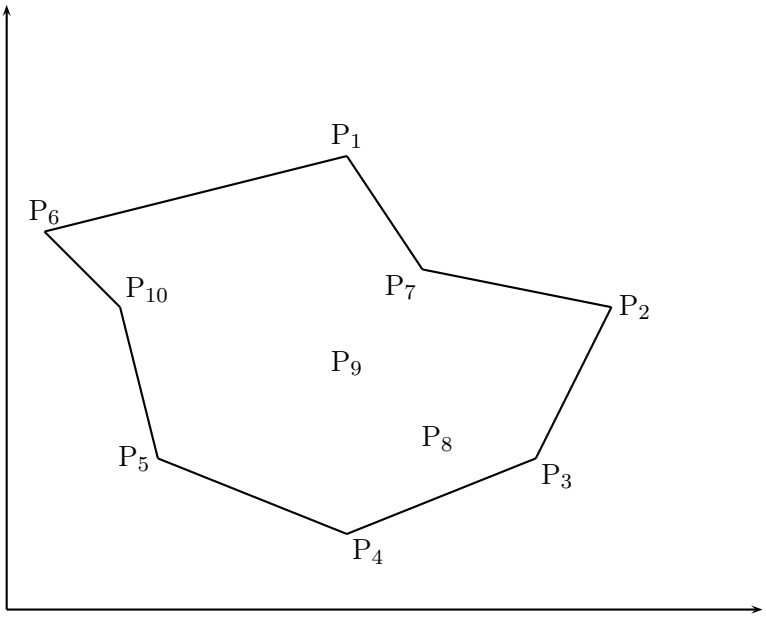

Figure 4.1: Enveloppe Clockwise

La figure ci-dessous, donne l'enveloppe convexe des sommets de l'Exemple 4.2 ci-dessus. Elle est déduite de la Figure 4.1. En effet en joignant les sommets P_1 et P_2 on forme un plus grand angle. Il en est de même pour les sommets P_6 et P_5.

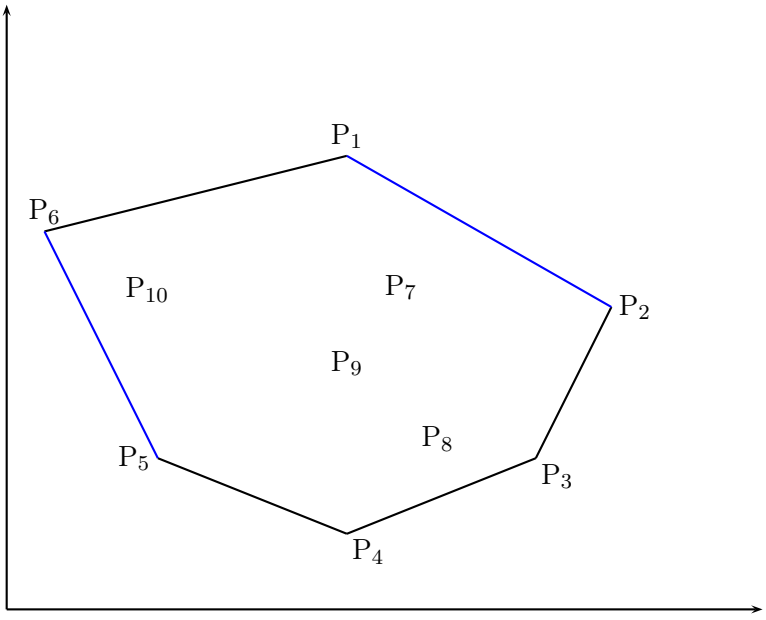

Figure 4.2: Enveloppe Convexe

Remarque 4.2. *Comme on peut le constater à la Figure 4.2, l'enveloppe convexe contient souvent très peu de sommets et ne donne pas beaucoup d'informations sur la construction du cycle du TSP et sa forme finale. Ainsi, nous utiliserons l'enveloppe clockwise pour la construction des routes.*

2. Insertion des clients restants

Les sommets n'appartenant pas à l'enveloppe clockwise sont insérés grâce à un critère de minimisation à deux niveaux. Le premier niveau de minimisation consiste à déterminer pour chaque sommet i, n'appartenant pas à l'enveloppe clockwise, la paire de sommets consécutifs $(j(i), j(i)+1)$ dans la route permettant une insertion qui entraine une augmentation minimale de la distance. Le second niveau de minimisation choisit la paire de sommets consécutifs parmi toutes celles choisies au premier niveau réalisant le minimum du ratio $\frac{d_{i,j(i)}+d_{i,j(i)+1}}{d_{j(i),j(i)+1}}$. Dans le cas où le minimum est atteint pour plus d'une paire de sommets, on choisit aléatoirement l'une d'entre elles. L'algorithme suivant construit toute la route du véhicule k:

Pseudo-Code 4.7. *Algorithme pour l'insertion des clients restants*
Fonction *insertion_clockwise(H_k, V_k)*

1. $S = V_k \setminus H_k$
2. *while* ($S \neq \varnothing$)
   (a) $\forall \quad i \in S$, $\{(j(i), j(i)+1)\} = \underset{(j,j+1)\in H_k}{\operatorname{argmin}} \{d_{i,j} + d_{i,j+1} - d_{j,j+1}\}$
   (b) $(j(i^*), i^*, j(i^*)+1) = \underset{(j(i),j(i)+1)\in H_k}{\operatorname{argmin}} \frac{d_{i,j(i)}+d_{i,j(i)+1}}{d_{j(i),j(i)+1}}$
   (c) *Insérer* i^* *entre les clients* $j(i^*)$ *et* $j(i^*)+1$
   (d) $S = S \setminus \{i^*\}$ $H_k = H_k \cup \{i^*\}$

Exemple 4.3. *Dans l'exemple 4.2, l'enveloppe clockwise est* $H_k = \{P_1, P_7, P_2, P_3, P_4, P_5, P_{10}, P_6\}$ *et l'ensemble des sommets n'appartenant pas à l'enveloppe clockwise est* $S = \{P_8, P_9\}$. *Le premier niveau de minimisation permet de retenir la paire* (P_3, P_4) *comme position où doit se faire l'insertion de* P_8 *et les paires* (P_3, P_4) *et* (P_4, P_5) *pour l'insertion du sommet*

P_9 (car le coût d'insertion est le même pour ces deux arêtes). Le second niveau de minimisation permet de choisir en définitive le sommet P_8 et la paire (P_3, P_4). Une fois le sommet P_8 inséré entre les sommets P_3 et P_4, $S = \{P_9\}$. Le premier niveau de minimisation permet de retenir la paire (P_8, P_4) comme position où doit se faire l'insertion de P_9. Comme S ne contient qu'un seul sommet et qu'il n'y a qu'une seule position d'insertion de ce sommet par rapport au premier niveau de minimisation, on n'a plus besoin du second niveau de minimisation. On en déduit que la paire (P_8, P_4) est en définitive la position où se fait l'insertion de P_9. Ci-dessous la solution du TSP de l'exemple 4.2.

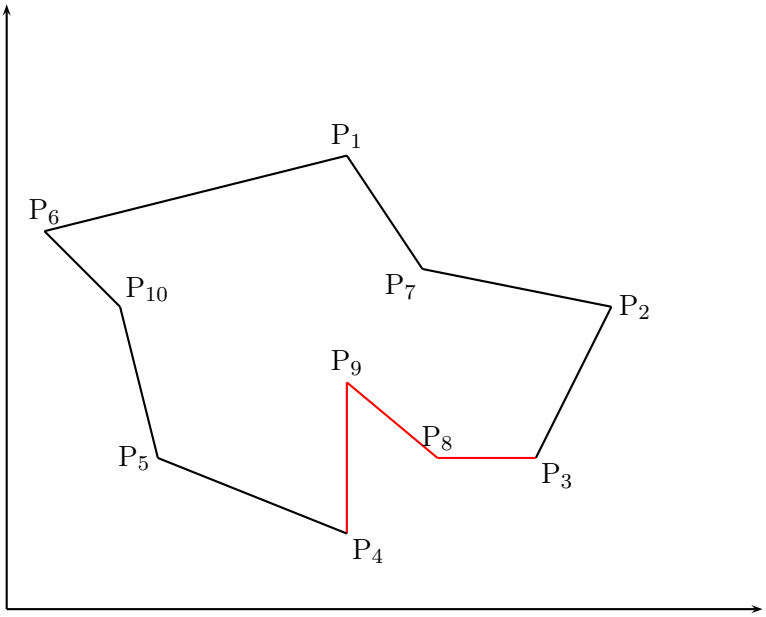

Figure 4.3: Solution du TSP

3. Insertion du dépôt

Cette phase s'appuie sur les trois précédentes. On obtient une solution, c'est à dire un cycle, pour chaque problème de voyageur de commerce que nous avons résolu précédemment (un par véhicule). Il ne reste plus qu'à insérer le dépôt dans chaque cycle. L'insertion du dépôt est faite de la même façon que dans l'algorithme pour l'insertion des clients restants.

Exemple 4.4. *Considérons l'exemple de la section précédente. Supposons que le dépôt a pour coordonnées* $D(3.5, 4.5)$. *Alors en appliquant l'algorithme pour l'insertion des clients restants, on obtient que la paire* (P_6, P_1) *permet la minimisation de premier niveau sur l'ensemble des paires formées de deux sommets consécutifs du cycle. La figure ci-dessous donne la route finale obtenue pour l'Exemple 4.2.*

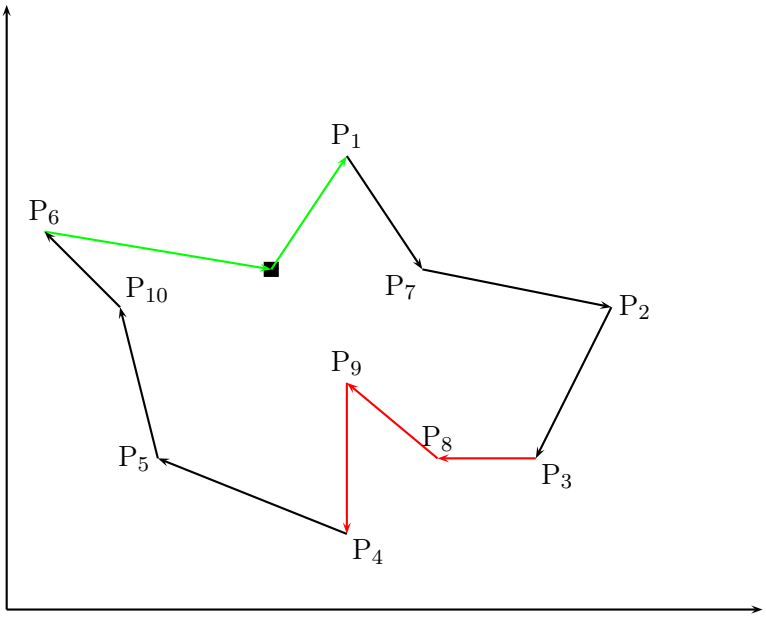

Figure 4.4: Route Complète

4.5 Les opérateurs de destruction

Les opérateurs de destruction ont pour but de retirer un nombre prescrit de clients de la solution courante. La structure de notre métaheurisitique ALNS pour la résolution du VRPPC comporte trois opérateurs de destruction: destruction aléatoire, destruction orientée véhicule et destruction selon Shaw.

4.5.1 Destruction Aléatoire

Il s'agit de l'opérateur de destruction le plus simple. On supprime aléatoirement un nombre donné de clients dans les routes d'une solution. Deux cas de figure peuvent se présenter. Le premier cas consiste à ne supprimer que des clients dans les routes de la flotte privée. Le second cas consiste à supprimer des clients des routes de la flotte privée et tous les clients du transporteur externe. Dans chaque cas, les clients supprimés sont conservés dans une liste $\mathcal{C}$. On construit d'abord un tableau dont la taille correspond au nombre de véhicules. Chaque entrée indique le nombre de clients à supprimer dans chaque route.

Pseudo-Code 4.8. *Algorithme de répartition des clients à supprimer*
Fonction *randremov_position_client(q)*

1. *somme1 = 0, somme2 =0;*
2. *Pour k =1,. . . , m faire*
   (a) *Générer un entier aléatoire x compris entre 0 et q*
   x = rand(0, q);
   (b) *Si somme1 < q ET somme2 +x ≤ q, alors tab[k] = x;*
   (c) *Sinon*
      i. *Si k == m, tab[k] = q-somme2;*
      ii. *Sinon, tab[k] = 0;*
   (d) *somme1 = somme1 + tab[k], somme2 = somme1;*
3. *Retourner tab;*

Voici ci-dessous la description de l'opérateur de destruction aléatoire.

Pseudo-Code 4.9. *Algorithme de la destruction aléatoire*
Fonction *randremov_supprimer_client_route_normale(Liste_routes, q, $\mathcal{C}$)*
tab = randremov_position_client(q);

Pour k =1,. . . , m faire

1. *Déterminer le nombre de clients à supprimer dans la route du véhicule k*
   nombre = tab[k];
2. *while(nombre > 0 ET card(k) > 0)*
   (a) *Choisir aléatoirement le client i à supprimer dans la route du véhicule k*
   i = random(Liste_routes[k]);

(b) *Soustraire la demande q_i du client i de la charge du véhicule k*
chargeCourante = $chargeVeh_k$ - q_i;

(c) *Supprimer le client i de l'ensemble des clients affectés au véhicule k*
$V_k = V_k \setminus \{i\}$;

(d) *Ajouter le client i dans la liste de clients $\mathcal{C}$*
$\mathcal{C} = \mathcal{C} \bigcup \{i\}$;

(e) *Diminuer de 1 le nombre d'éléments dans la route du véhicule k*
nombre = nombre -1, card(k) = card(k) - 1;

(f) *Modifier la nouvelle charge chargeVeh du véhicule k*
$chargeVeh_k$ = chargeCourante;

4.5.2 Destruction Orientée Véhicule

L'une des difficultés majeurs dans la résolution des problèmes de tournées de véhicules est de réduire le nombre de véhicules utilisés pour servir tous les clients. De façon générale, il est très difficile de conduire la recherche vers la réduction du nombre de véhicules. C'est donc pour pallier à ce problème qu'on fait appel à l'opérateur de destruction orienté véhicule.

Cet opérateur de destruction consiste à détruire tous les clients appartenant à une même route. Il a été mentionné très succinctement dans l'article de L.-M. Rousseau et al. [50]. De façon plus précise la suppression de q clients grâce à cet opérateur se décrit de la manière suivante. On choisit une route aléatoire et on supprime tous ses clients. Si on n'a pas atteint q, on choisit aléatoirement une autre route et on répète. Tous les clients supprimés sont mis dans une liste $\mathcal{C}$.

Pseudo-Code 4.10. *Algorithme de la destruction orientée véhicule*
Fonction *samveh_supprimer_des_clients_de_route(Liste_routes, q, $\mathcal{C}$)*

1. *somme = 0;*
2. *while(somme < q)*

(a) *Déterminer le nombre de clients restants à supprimer*
qPrime = q - somme;

(b) *Choisir aléatoirement une route k à l'intérieur de laquelle on veut supprimer des clients.*
k = random(Liste_routes);

(c) Si qPrime < card(k)

i. On supprime exactement qPrime clients dans la route k.
Pour $l = 1, 2, \cdots, qPrime$ *faire*

A. Identifier aléatoirement le client i à supprimer dans la route du véhicule k
i = random(Liste_routes[k]);

B. Soustraire la demande q_i *du client i de la charge du véhicule k*
$chargeVeh_k$ = $chargeVeh_k$ - q_i;

C. Supprimer le client i de l'ensemble des clients affectés au véhicule k
$V_k = V_k \setminus \{i\}$;

D. Ajouter le client i dans $\mathcal{C}$;
$\mathcal{C} = \mathcal{C} \bigcup \{i\}$;

ii. Diminuer le nombre d'éléments dans la route du véhicule k
card(k) = card(k) - qPrime;

iii. Mettre à jour le nombre de clients déjà supprimés
somme = somme + qPrime ;

Sinon,

i. Supprimer tous les clients de la route k
$V_k = \emptyset$;

ii. Ajouter tous les clients supprimés de la route k dans $\mathcal{C}$;
$\mathcal{C} = \mathcal{C} \bigcup Liste\_routes[k]$;

iii. Modifier la charge du véhicule k
$chargeVeh_k$ = 0;

iv. Mettre à jour le nombre de clients déjà supprimés
somme = somme + card(k) ;

4.5.3 Destruction selon Shaw

Cet opérateur de destruction a été développé par P. Shaw [52, 53] pour le problème PDPTW (Pickup and Delivery Problem with Time windows). Par la suite certaines modifications ont été apportées par D. Pisinger et S. Ropke [44, 43, 48] pour le rendre encore plus performant. La philosophie qui le sous-tend est de supprimer des clients qui ont une grande similarité entre eux. En effet si nous supprimons des clients qui n'ont rien en commun et qui sont très différents les uns des autres, on pourrait ne rien avoir comme gain après leur réinsertion. D'où la nécessité de définir une métrique $R(i, j)$ permettant de définir la notion

de similarité entre deux clients i et j. Plus la valeur de $R(i,j)$ est faible, plus les clients i et j sont similaires.

Définition 4.7. *Soit i un client et L une liste de clients ne contenant pas i. On dit que L est triée selon la métrique R avec i pour repère, si les clients de L sont ordonnés de manière telle que:*

$$l < l' \quad \Longrightarrow \quad R(i, L[l]) < R(i, L[l'])$$

Quand il s'agit de trier une liste, nous utilisons le tri rapide (quicksort) [33]. Dans un souci de diversification de la recherche pour le VRPPC nous définissons trois métriques différentes.

La première métrique

Etant donné deux clients i et j, leur similarité se mesure par rapport à leur distance.

$$R(i,j) = d_{i,j}$$

La deuxième métrique

Pour deux clients i et j, on définit leur similarité par rapport à la différence de leurs demandes.

$$R(i,j) = |q_i - q_j|$$

La troisième métrique

Elle combine les deux métriques précédentes où chacune d'elle est pondérée par un coefficient. Elle utilise les deux invariants du problème relatif à la distance maximale et à la différence maximale des demandes, à savoir $d_{\max}$ et $g_{\max}$. En clair, pour deux clients i et j, on définit leur similarité de la façon suivante.

$$R(i,j) = \varphi \frac{d_{i,j}}{d_{\max}} + \psi \frac{|q_i - q_j|}{g_{\max}}$$

Les paramètres φ et ψ sont des réels tels que $0 \leq \varphi,\ \psi \leq 1$ et $\varphi + \psi = 1$. Une étude sera faite pour déterminer les valeurs conduisant aux meilleurs résultats.

Cet opérateur de destruction détermine d'abord de façon aléatoire un client de la solution courante et le retire de celle-ci. Le client supprimé est mis dans une liste $\mathcal{C}$. La première itération consiste à choisir aléatoirement un client r de $\mathcal{C}$ et à trier la liste L formée des clients de la solution s n'appartenant pas à $\mathcal{C}$ selon la métrique R avec r pour repère. On choisit par la suite le client situé à la position $y^p|L|$ dans la liste L (où y est un nombre choisi aléatoirement dans l'intervalle $[0, 1[$ et $p > 1$ est un paramètre déterministe), pour l'ajouter à $\mathcal{C}$. On répète tant que le nombre d'éléments de $\mathcal{C}$ n'atteint pas q (le nombre de clients à supprimer dans les routes de la flotte privée). Voici ci-dessous l'algorithme pour l'opérateur de destruction selon Shaw. Il supprime des clients dans les routes de la flotte privée. La mesure R y intervenant se réfère à l'une des trois mesures evoquées ci-dessus.

Pseudo-Code 4.11. *Algorithme de destruction selon Shaw*
Fonction *shawremov_supprimer_client_mesure(Liste_routes, q, $\mathcal{C}$, p)*

1. *Récupérer la liste des clients des routes de la flotte privée F_p*
   $F_p = \bigcup_{k=1}^{m} Liste\_routes[k]$;
2. *Choisir aléatoirement un client i dans F_p et le supprimer de cette liste.*
   $i = rand(F_p)$, $F_p = F_p \setminus \{i\}$;
3. *Mettre à jour le nombre d'éléments de la liste F_p*
   $card(F_p) = card(F_p) - 1$;
4. *Déterminer le véhicule k^* desservant le client i, c'est à dire*
   $i \in Liste\_routes[k^*]$;
5. *Soustraire la demande q_i du client i de la charge du véhicule k^**
   $chargeVeh_{k^*} = chargeVeh_{k^*} - q_i$;
6. *Supprimer le client i de l'ensemble des clients affectés au véhicule k^**
   $V_{k^*} = V_{k^*} \setminus \{i\}$;
7. *Ajouter le client i, supprimé de F_p, dans $\mathcal{C}$;*
   $\mathcal{C} = \mathcal{C} \bigcup \{i\}$;
8. *Réduire de 1 le nombre d'éléments dans la route du véhicule k^**
   $card(k^*) = card(k^*) - 1$;
9. *Tant que $|\mathcal{C}| < q$ faire*

(a) *Choisir aléatoirement un client r dans $\mathcal{C}$.*
$r = rand(\mathcal{C})$;

(b) *Trier F_p avec le tri rapide selon la métrique R avec r pour repère*
F_p = *quickSort(R, F_p, r)*;

(c) *Choisir un nombre aléatoire y dans [0, 1[*
$y = rand(0, 1.0)$;

(d) *Déterminer la position du client qu'on doit ajouter dans $\mathcal{C}$*
position = $\lfloor y^p * card(F_p) \rfloor$;

(e) *Récupérer le client j dont la position a été calculée et le supprimer de F_p*
$j = F_p[position]$, $F_p = F_p \setminus \{j\}$;

(f) *Mettre à jour le nombre d'éléments de la liste F_p*
$card(F_p) = card(F_p) - 1$;

(g) *Déterminer le véhicule k^* desservant le client j, c'est à dire*
$j \in Liste\_routes[k^*]$;

(h) *Soustraire la demande q_j du client j de la charge du véhicule k^**
$chargeVeh_{k^*}$ = $chargeVeh_{k^*}$ - q_j;

(i) *Supprimer le client j de l'ensemble des clients affectés au véhicule k^**
$V_{k^*} = V_{k^*} \setminus \{j\}$;

(j) *Ajouter le client j dans $\mathcal{C}$;*
$\mathcal{C} = \mathcal{C} \bigcup \{j\}$;

(k) *Réduire de 1 le nombre d'éléments dans la route du véhicule k^**
$card(k^*) = card(k^*) - 1$;

4.6 Les opérateurs de reconstruction

Les opérateurs de reconstruction ont pour but de réinsérer les clients qui ont été retirés de la solution courante. La structure de notre métaheurisitique ALNS pour la résolution du VRPPC comporte deux opérateurs de reconstruction: l'heuristique vorace et l'heuristique des regrets. Pour chacun de ces opérateurs, on conserve l'information sur l'insertion d'un client sous la forme d'un triplet. Les deux premières composantes sont les deux clients consécutifs i et j d'une route entre lesquels on veut insérer le nouveau client et la troisième composante représente le coût de cette insertion. L'application $pr_{1,2}$ extrait l'arête (i, j) du triplet tandis que pr_3 extrait le coût de l'insertion.

Définition 4.8. *Soit V l'ensemble des clients. On définit les applications pr_3 et $pr_{1,2}$ de la manière suivante*

$$\begin{array}{rccc} pr_3 : & V^2 \times \mathbb{R} & \longrightarrow & \mathbb{R} \\ & (i,\ j, & t) & \longmapsto t \end{array}$$

$$\begin{array}{rccc} pr_{1,2} : & V^2 \times \mathbb{R} & \longrightarrow & V^2 \\ & (i,\ j, & t) & \longmapsto (i,\ j) \end{array}$$

On définit sur $V^2 \times \mathbb{R}$ la relation $\preceq$ de telle sorte:

$\forall\, \mathcal{X}, \mathcal{Y} \in V^2 \times \mathbb{R}$, $\mathcal{X} \preceq \mathcal{Y} \Leftrightarrow pr_3(\mathcal{X}) \leq pr_3(\mathcal{Y})$

4.6.1 L'Heuristique Vorace

L'heuristique vorace est une procédure itérative qui insère à chaque itération le client qui entraîne la plus petite augmentation du coût de la solution temporaire. On a ici un algorithme simple et de faible complexité. Il ne conduit pas à la route optimale mais donne une bonne solution. De façon générale, les clients supprimés sont insérés dans la solution partiellement détruite (ensemble de plusieurs routes) ou affectés au transporteur externe en suivant quatre étapes. Nous donnons ici la description de chacune d'elles et le pseudo-code de la fonction décrivant la première étape. Le pseudo-code de la seconde étape se déduit aisement de la première.

Première étape

Ici on suppose donnée une solution partiellement détruite x_d et un client l. Cette étape consiste à déterminer la paire (i_l, j_l) qui réalise le minimum de $\frac{d_{i,l}+d_{l,j}}{d_{i,j}}$ sur toutes les paires (i, j) de clients consécutifs appartenant aux routes de x_d et la route k^* la contenant. On retient $(i_l^{k^*}, j_l^{k^*})$ où se trouve le meilleur endroit pour insérer l. Ci-dessous, le pseudo-code de l'algorithme.

Pseudo-Code 4.12. *Algorithme de la meilleure position d'insertion pour un client*
Fonction *bgh_cout_insertion(x_d, l)*

1. *Initialisation de la meilleure position pour l'insertion de l*
   $arcO^l = (0, 0, +\infty)$, $rappDecision_l = +\infty$;

2. *Initialisation du nombre de routes ne pouvant servir l*
   $rP = 0$;

3. *Pour* $k = 1, \ldots, m$ *faire*

   (a) *On définit l'ensemble de tous les arcs de la route k:*

   $$\mathcal{A}_k = \{(i, j), \textit{si le véhicule } k \textit{ se déplace de } i \textit{ à } j\}$$

   (b) *Si* $q_l + chargeVeh_k \leq Q_k$, *alors*

      i. *Initialisation de la variable* ameliore. *Elle permet de savoir s'il existe un arc de $\mathcal{A}_k$ qui améliore le coût de l'insertion de l.*
      ameliore $= Faux$;

      ii. *Pour tout* $(i, j) \in \mathcal{A}_k$ *faire*

      A. $\text{somme}_{i,j} = d_{i,l} + d_{l,j}$;

      B. $\text{valeur}_{i,j} = d_{i,j}$;

      C.
      $$\text{rapp}_{i,j} = \begin{cases} \text{somme}_{i,j}/\text{valeur}_{i,j}, & si \quad \text{valeur}_{i,j} > 0 \\ \text{somme}_{i,j}/2, & sinon \end{cases}$$

      D. $T^k_{i,j} = (i, j, \text{rapp}_{i,j})$

      E. *Mettre à jour la meilleure position pour l'insertion de l*
      Si $T^k_{i,j} \preceq arcO^l$
      - $arcO^l = T^k_{i,j}$;
      - ameliore $=$ *Vrai;*

      iii. *Mettre à jour la route où doit se faire l'insertion de l*
      Si ameliore $== Vrai$, *alors* $k^* = k$;

   Sinon, $rP = rP + 1$;

4. *Déterminer le meilleur endroit pour insérer l*
   Si $rP == m$, *alors le client l ne peut être inséré dans aucune route de la flotte privée.*
   $rappDecision_l = +\infty$

Sinon,

$$\begin{cases} (i_l^{k^*}, j_l^{k^*}) = pr_{1,2}(arcO^l); \\ rappDecision_l = pr_3(arcO^l); \end{cases}$$

Deuxième étape

On applique la fonction de la première étape sur tous les clients de $\mathcal{C}$. A chaque application, on retient rappDecision$_l$ et $(i_l^{k^*}, j_l^{k^*})$. On choisit le client l^*, pour lequel rappDecision$_l$ est minimal, la route k^{**} à l'intérieur de laquelle aura lieu son éventuelle insertion et les deux clients $i_{l^*}^{k^{**}}$, $j_{l^*}^{k^{**}}$ de k^{**} entre lesquels l^* doit s'insérer.

Troisième étape

Elle consiste d'abord à comparer le coût minimal de l'insertion du client l^* au coût du transporteur externe qui lui est associé. Si le coût du transporteur externe est plus avantageux par rapport à l'insertion de l^* dans la route k^{**}, alors l^* est affecté au transporteur externe. Dans le cas contraire, on procède à son insertion dans la route k^{**}.

Quatrième étape

Elle consiste à appliquer les étapes 2 et 3 jusqu'à ce que tous les clients retirés de la solution aient été considérés par l'heuristique vorace de réinsertion.

4.6.2 L'Heuristique des regrets

Le concept du regret a été originellement introduit par F. Tilman et T. Cain [58]. Il a été généralisé par la suite par Potvin et Rousseau [46] pour étudier le VRPTW (Vehicle Routing Problem with Time Windows). L'heuristique vorace est aveugle et se retrouve souvent à la fin avec des clients difficiles à insérer. En effet, l'insertion d'un client dans la solution ne nous donne aucune information sur les répercussions de cette insertion sur les prochains clients à insérer. Elle peut même avoir pour conséquence de retarder l'insertion des clients les plus difficiles à insérer, au moment où il reste peu ou pas d'alternatives réalisables. L'heuristique des regrets essaie de résoudre ce problème en incorporant une sorte de visibilité à long terme dans l'ordre des clients à insérer. Elle réinsère les clients

pour lesquels la différence entre le coût de la seconde meilleure insertion et le coût de la meilleure insertion est le plus grand. En effet, la perte est plus grande pour ces clients si la meilleure insertion devient non réalisable dû à l'insertion d'autres clients. La mesure de regret a aussi été utilisée par S. Ropke et D. Pisinger [43, 48].

Définition 4.9. *Soit i un client de la liste des clients $\mathcal{C}$. Soit x_{ij} la variable indiquant la route pour laquelle i a le j ième plus petit coût d'insertion et $\Delta f_{i,x_{ij}}$ ce coût. On appelle valeur de regret de i notée r_i, la différence entre le coût d'insertion de i à la deuxième meilleure position et à la meilleure position. On a,*

$$r_i = \Delta f_{i,x_{i2}} - \Delta f_{i,x_{i1}}.$$

On appelle client de regret maximal, le client noté i^* pour lequel

$$r_{i^*} = \max_{i \in \mathcal{C}} r_i.$$

On gagnerait mieux à insérer d'abord le client de regret maximal dans sa meilleure position, autrement son insertion sera plus coûteuse. L'heuristique des regrets se compose de quatre étapes. Nous donnons ici la description de chacune d'elles et le pseudo-code de la fonction pour la première étape. Le pseudo-code de la seconde étape se déduit aisement de la première.

Première étape

Ici on suppose donnée une solution partiellement détruite x_d et une liste de clients $\mathcal{C}$ n'appartenant pas à x_d. Cette étape consiste à calculer la valeur de regret r_l pour un client quelconque l de $\mathcal{C}$. Ci-dessous, le pseudo-code de l'algorithme.

Pseudo-Code 4.13. *Algorithme pour le calcul de la valeur de regret d'un client*
Fonction *regheuAm_calcul_valeur_de_regret(x_d, l)*

1. *Initialisation de la première et de la deuxième meilleures positions pour l'insertion de l*
   $arcO^l = (0, 0, +\infty)$, $arcA^l = (0, 0, +\infty)$, $rappDecision_l = +\infty$
2. *Initialisation du nombre de routes ne pouvant servir l*
   $rP = 0$;
3. *Pour $k = 1, \ldots, m$ faire*

(a) *On définit l'ensemble de tous les arcs de la route k:*

$$\mathcal{A}_k = \{(i,j), \textit{si le véhicule } k \textit{ se déplace de } i \textit{ à } j\}$$

(b) *Si* q_l + $chargeVeh_k \leq Q_k$, *alors*

i. *Initialisation des variables* $\text{amelior}e_1$ *et* $\text{amelior}e_2$. *Elles comptent les arcs de* $\mathcal{A}_k$ *qui améliorent les deux meilleurs coûts d'insertion de* l.
$\text{amelior}e_1 = Faux$, $\text{amelior}e_2 = Faux$;

ii. *Pour tout* $(i,j) \in \mathcal{A}_k$ *faire*

A. $\text{somme}_{i,j} = d_{i,l} + d_{l,j}$;

B. $\text{valeur}_{i,j} = d_{i,j}$;

C.
$$\text{rapp}_{i,j} = \begin{cases} \text{somme}_{i,j}/\text{valeur}_{i,j}, & si \quad \text{valeur}_{ij} > 0 \\ \text{somme}_{i,j}/2, & sinon \end{cases}$$

D. $T^k_{i,j} = (i, j, \text{rapp}_{i,j})$

E. *Mettre à jour les deux meilleures positions pour l'insertion de* l
Si $T^k_{i,j} \preceq arcO^l$
- $arcT = arcO^l$;
- $arcO^l = T^k_{i,j}$;
- $arcA^l = arcT$;
- $\text{amelior}e_1 = Vrai$, $\text{amelior}e_2 = Vrai$;

Si $arcO^l \preceq T^k_{i,j}$ *ET* $T^k_{i,j} \preceq arcA^l$
- $arcA^l = T^k_{i,j}$;
- $\text{amelior}e_2 = Vrai$;

iii. *Mettre à jour la route où doit se faire l'insertion de* l

A. *Si* $\text{amelior}e_1 == Vrai$, *alors* ${k_1}^* = k$;

B. *Si* $\text{amelior}e_2 == Vrai$, *alors* ${k_2}^* = k$;

Sinon, $rP = rP + 1$;

4. *Déterminer les deux meilleures positions pour insérer* l
Si $rP == m$, *alors le client* l *ne peut être inséré dans aucune route de la flotte privée.*
$rappDecision_l = +\infty$,

Sinon,

- $(i_l^{k_1^*}, j_l^{k_1^*}) = pr_{1,2}(arcO^l)$;
- $(i_l^{k_2^*}, j_l^{k_2^*}) = pr_{1,2}(arcA^l)$;
- $rappDecision_l = pr_3(arcO^l)$;
- *Calculer la valeur de regret du client l*
  $r_l = pr_3(arcA^l) - pr_3(arcO^l)$;

Deuxième étape

On applique la fonction de la première étape sur tous les clients de $\mathcal{C}$. A chaque application, on retient $(i_l^{k_1^*}, j_l^{k_1^*})$, $(i_l^{k_2^*}, j_l^{k_2^*})$ et r_l. On choisit le client l^* de $\mathcal{C}$ de regret maximal r_{l^*}, la route k_1^{**} à l'interieur de laquelle aura lieu son éventuelle insertion, la paire $(i_{l^*}^{k_1^{**}}, j_{l^*}^{k_1^{**}})$ de k_1^{**} entre lesquels l^* doit s'insérer, la route k_2^{**} et la paire $(i_{l^*}^{k_2^{**}}, j_{l^*}^{k_2^{**}})$ de k_2^{**}.

Troisième étape

Elle consiste d'abord à comparer les coûts minimaux de l'insertion du client l^* dans les routes k_1^{**} et k_2^{**} à e_{l^*}, le coût du transporteur externe qui lui est associé. Si e_{l^*} est plus avantageux par rapport à l'insertion de l^* dans la route k_1^{**}, alors l^* est affecté au transporteur externe. Si e_{l^*} est moins avantageux par rapport à l'insertion de l^* dans la route k_2^{**}, alors l^* est inséré dans la route k_1^{**}. Si $e_{l^*} - \mathrm{pr}_3(arcO^l) > \mathrm{pr}_3(arcA^l) - e_{l^*}$, alors l^* est inséré dans la route k_1^{**}. Dans le cas contraire il est affecté au transporteur externe.

Quatrième étape

Elle consiste à appliquer les étapes 2 et 3 jusqu'à ce que tous les clients retirés de la solution aient été considérés par l'heuristique des regrets de réinsertion.

4.7 Le critère d'acceptation et d'arrêt

Le critère d'acception est évoqué à la ligne 3.c du pseudo-code 4.3. Il permet à la solution temporaire x^t d'être promue au statut de solution courante x.

4.7.1 Acceptation normale

De façon naturelle, la solution temporaire x^t est acceptée si $\mathrm{coût}(x^t) \leq \mathrm{coût}(x)$.

4.7.2 Acceptation probabiliste

La condition d'acceptation normale ne saurait rendre l'algorithme ALNS efficace en ce sens que la recherche pourrait rester coincée dans un minimum local. Pour pallier à ce problème on peut accepter des solutions moins bonnes.

Soit $T > 0$ une variable température initialisée au départ de l'algorithme à une valeur T_{start}. Dans la pratique T_{start} est calculée en fonction de la valeur de la solution initiale du problème. Nous utilisons la technique développée par S. Ropke et D. Pisinger [49] pour déterminer cette valeur. Soit z' le coût de la solution courante. La valeur de T_{start} est fixée de telle sorte que toute solution temporaire x^t dont le coût est $1 + w$ fois plus grand que z' est acceptée avec une probabilité de 0.5. Le réel w est appelé le paramètre de contrôle de la température de départ.

Après un certain nombre d'itérations de l'algorithme ALNS, la température courante T est mise à jour suivant la formule $T = cT$ où c est un réel appelé taux de refroidissement, avec $0 < c < 1$. Si $\text{coût}(x^t) > \text{coût}(x)$, alors la solution temporaire x^t est acceptée avec une probabilité égale à

$$\exp(-\frac{\text{coût}(x^t) - \text{coût}(x)}{T})$$

4.7.3 Le critère d'arrêt

Ici il s'agit d'un certain nombre *nbIteration* d'itérations spécifié. Dans notre travail $nbIteration = 50\,000$.

4.8 La mise à jour adaptative des poids

Une heuristique peut difficilement être performante pour toutes les instances d'un problème donné. L'objectif qu'on poursuit grâce à la mise à jour des poids des opérateurs, c'est de choisir les opérateurs qui performent le mieux pour chaque instance. Nous expliquons dans cette section, comment se fait le choix des opérateurs, et donnons la formule de mise à jour des poids des opérateurs.

4.8.1 Choix des opérateurs de destruction et de reconstruction

Au début de l'algorithme on suppose que tous les opérateurs ont le même poids. Les poids ρ^-, ρ^+ sont utilisés afin de choisir les opérateurs en utilisant le principe de la roulette.
Soit ϕ_j^- la probabilité de choisir le j ième opérateur de destruction, selon la formule de la roulette:

$$\phi_j^- = \frac{\rho_j^-}{\sum_{k=1}^{|\Omega^-|} \rho_k^-}$$

La probabilité de choisir le j ième opérateur de reconstruction se fait de la même manière. On genère ces probabilités grâce à la technique développée dans [38] pp.459-465.

4.8.2 Mise à jour des scores des opérateurs dans un segment

Définition 4.10. *[48] Un segment est un nombre nbFixe d'itérations consécutives de l'algorithme ALNS. Dans notre travail nous avons* $nbFixe = 100$.

Définition 4.11. *[48] On appelle score d'un opérateur, une valeur permettant de mesurer sa performance à chaque itération de l'algorithme ALNS. Au début de chaque segment, le score de chaque opérateur est réinitialisé à zéro.*

Soient α, β et γ trois paramètres réels. Le score d'un opérateur est incrémenté de :

- α, si la dernière opération de destruction-reconstruction impliquant cet opérateur conduit à l'obtention d'une nouvelle meilleure solution globale x^b.
- β, si la dernière opération de destruction-reconstruction impliquant cet opérateur conduit à l'obtention d'une solution qui est meilleure que la solution courante.
- γ, si la dernière opération de destruction-reconstruction impliquant cet opérateur conduit à l'obtention d'une solution qui n'est pas meilleure que la solution courante mais qui est acceptée selon le critère d'acceptation probabiliste.

4.8.3 Formule de mise à jour des poids

En 2010, D. Pisinger et S. Ropke [44] ont donné une formule permettant de mettre à jour le poids des opérateurs de destruction et de reconstruction à chaque itération de l'algorithme ALNS. Dans le cadre de notre travail, nous préférons utiliser la formule des mêmes auteurs dans leurs publications antérieures [48, 43] où les poids sont mis à jour à la fin de chaque

segment selon le score obtenu.

Supposons qu'après avoir parcouru le segment s, le score cumulatif de l'opérateur de destruction i est score(i) et que cet opérateur a été utilisé n_i fois au cours de ce segment. Alors le poids qui sera utilisé au cours du segment $s+1$ pour cet opérateur correspond à:

$$\rho^-_{i,s+1} = \lambda \, \frac{\text{score}(i)}{n_i} + (1-\lambda)\rho^-_{i,s}, \quad i = 1, \ldots, |\Omega^-|$$

Le paramètre réel $\lambda \in [0,1]$ intervenant dans la formule ci-dessus est appelé facteur de réaction. Il contrôle la sensisibilité des poids par rapport aux performances des opérateurs. Ainsi, plus λ est grand, plus la performance de l'opérateur i au cours du segment s a un impact important lors de la mise à jour du poids $\rho^-_{i,s}$. À l'inverse, plus λ est petit moins cette performance a un impact lors de la mise à jour du poids $\rho^-_{i,s}$. Les poids des opérateurs de reconstruction sont mis à jour de la même manière.

Chapitre 5

Résultats numériques

Ce chapitre présente les résultats numériques obtenus en lançant notre algorithme ALNS sur deux familles d'instances avec flottes hétérogènes. Il est organisé de la manière suivante. Dans la section 5.1, nous décrivons les instances. Dans la section 5.2, nous déterminons les valeurs des invariants associés à chaque instance. Dans la section 5.3, nous faisons une étude afin de déterminer un critère d'arrêt commun pour des instances ayant un nombre de clients appartenant à un intervalle spécifique. Cette étude conduit dans la section 5.4 à une première classification des deux familles d'instances selon le nombre de clients. On étudie également la variation du coût de la solution du VRPPC sur les représentants des différentes classes d'instances, de même que le temps moyen de calcul. Dans la section 5.5, l'étude de la sensibilité des paramètres fait intervenir deux nouvelles classifications selon la distance maximale et l'écart maximal. Nous déduisons ensuite de ces trois classifications une classification unifiée, dont les représentants constituent notre ensemble échantillon. Dans la section 5.6, nous faisons une analyse de sensibilité pour déterminer les meilleures valeurs des paramètres de notre algorithme ALNS sur les instances de l'ensemble échantillon. Enfin, nous présentons les résultats expérimentaux finaux sur toutes les instances.

5.1 Description des instances tests

Les instances tests se divisent en deux sous-ensembles: 14 instances de Christofides et Eilon notées CE-H et 20 instances de Golden G-H, toutes avec une flotte hétérogène. Les véhicules de la flotte privée n'ont pas tous les mêmes capacités. Pour toutes ces instances,

les coordonnées des clients et les demandes des clients sont identiques à celles des instances originales conçues pour le problème VRP [24]. Des modifications ont été apportées par M.-C. Bolduc et al. [6] pour les rendre conformes au VRPPC.

5.1.1 Le nombre de véhicules

La capacité totale des véhicules $\overline{Q}$ est fixée à

$$\overline{Q} = 0.8\overline{q}$$

où $\overline{q}$ est la demande totale des clients. Les flottes des véhicules dans chaque instance sont constituées de trois types de véhicules A, B et C en nombres respectifs designés par m_A, m_B et m_C. Les capacités et les coûts fixes de ces trois types de véhicules sont 80 % (type A), 100 % (type B) et 120 % (type C) de ceux de la flotte privée pour les instances homogènes, exception faite des deux premières instances de Christofides et Eilon, et de la cinquième instance de Golden.

5.1.2 Les coûts variables

Pour chaque unité de distance parcourue, le coût variable est fixé à 1. Le coût fixe du véhicule est égal à la longueur moyenne des routes des véhicules dans la meilleure solution connue de l'instance VRP originale, arrondie à la vingtaine près.

5.1.3 Les coûts du transporteur externe

Définition 5.1. *On appelle densité du VRP original, notée $\overline{n}$, le nombre moyen de clients desservis par chacun des véhicules dans la meilleure solution connue pour le problème VRP original.*

Définition 5.2. *On appelle coefficient d'éloignement du client i dans le VRP original le réel μ_i défini par*

$$\mu_{\mathrm{i}} = \begin{cases} 1 & si \quad q_i \in [q_{min},\ q_{min} + \eta[\\ 1.5 & si \quad q_i \in [q_{min} + \eta,\ q_{min} + 2\eta[\\ 2 & si \quad q_i \in [q_{min} + 2\eta,\ q_{max}[\end{cases}$$

où $\eta = g_{max}/3$, avec g_{max} l'écart entre la plus grande demande q_{max} et la plus petite demande q_{min}.

Définition 5.3. *Le coût du transporteur externe e_i pour un client i dans le VRPPC est défini par :*

$$e_i = \frac{f_k}{\overline{n}} + \mu_i c_{0i}$$

où f_k est le coût fixe du véhicule k et c_{0i} le coût du déplacement du dépôt au client i.

		A			B			C		
Instance	n	m_A	Q_A	f_A	m_B	Q_B	f_B	m_C	Q_C	f_C
CE-H-0l	50	2	160	140	2	192	168	0	0	0
CE-H-02	75	4	112	80	5	168	120	0	0	0
CE-H-03	100	2	160	112	2	200	140	2	240	168
CE-H-04	150	2	160	96	4	200	120	3	240	144
CE-H-05	199	7	160	80	5	200	100	2	240	120
CE-H-06	50	1	128	112	2	160	140	1	192	168
CE-H-07	75	4	112	96	3	140	120	2	168	144
CE-H-08	100	1	160	128	1	200	160	4	240	192
CE-H-09	150	4	160	96	3	200	120	3	240	144
CE-H-l0	199	2	160	96	5	200	120	6	240	144
CE-H-11	120	2	160	144	2	200	180	2	240	216
CE-H-12	100	2	160	96	3	200	120	3	240	144
CE-H-13	120	1	160	208	4	200	260	1	240	312
CE-H-14	100	1	160	96	1	200	120	5	240	144

Table 5.1: Instances hétérogènes de Christofides et Eilon

		A			B			C		
Instance	n	m_A	Q_A	f_A	m_B	Q_B	f_B	m_C	Q_C	f_C
G-H-01	240	3	440	656	1	550	820	3	660	984
G-H-02	320	2	560	848	2	700	1060	4	840	1272
G-H-03	400	3	720	1104	3	900	1380	2	1080	1656
G-H-04	480	2	800	1376	4	1000	1720	2	1200	2064
G-H-05	200	2	720	1296	2	900	1620	0	0	0
G-H-06	280	3	720	1360	2	900	1700	1	1080	2040
G-H-07	360	3	720	1168	1	900	1460	3	1080	1752
G-H-08	440	1	720	1184	2	900	1480	5	1080	1776
G-H-09	255	6	800	48	3	1000	60	3	1200	72
G-H-10	323	3	800	48	3	1000	60	6	1200	72
G-H-11	399	6	800	64	8	1000	80	1	1200	96
G-H-12	483	6	800	64	6	1000	80	4	1200	96
G-H-13	252	6	800	48	4	1000	60	10	1200	72
G-H-14	320	11	800	48	2	1000	60	11	1200	72
G-H-15	396	7	800	48	9	1000	60	10	1200	72
G-H-16	480	12	800	48	6	1000	60	11	1200	72
G-H-17	240	4	160	32	7	200	40	6	240	48
G-H-18	300	7	160	48	9	200	60	6	240	72
G-H-19	360	9	160	48	7	200	60	10	240	72
G-H-20	420	16	160	48	6	200	60	10	240	72

Table 5.2: Instances hétérogènes de Golden

5.2 Détermination des valeurs des invariants

Il s'agit dans cette section de déterminer pour chaque instance la somme totale des demandes des clients ($\overline{q}$), la capacité totale de tous les véhicules de la flotte privée ($\overline{Q}$). De même, on détermine la charge excédentaire (E), l'indice critique (h), la distance maximale (d_{max}) et l'écart maximal (g_{max}) tels que définis dans la section 4.2. Nous rapportons dans les

tableaux 5.3 et 5.4 les valeurs de ces invariants respectivement pour les instances hétérogènes de Christofides et Eilon, pour les instances hétérogènes de Golden.

instance	$\overline{q}$	$\overline{Q}$	E	h	d_{max}	g_{max}
CE-H-01	777	704	73	3	85.6329	38
CE-H-02	1364	1288	76	3	85.276	36
CE-H-03	1458	1200	258	11	91.8314	40
CE-H-04	2235	1840	395	17	91.8314	40
CE-H-05	3186	2600	586	25	91.8314	40
CE-H-06	777	640	137	6	85.6329	38
CE-H-07	1364	1204	160	7	85.276	36
CE-H-08	1458	1320	138	7	91.8314	40
CE-H-09	2235	1960	275	12	91.8314	40
CE-H-10	3186	2760	426	18	91.8314	40
CE-H-11	1375	1200	175	14	114.978	33
CE-H-12	1810	1640	170	5	96.1769	40
CE-H-13	1375	1200	175	13	114.978	33
CE-H-14	1810	1560	250	8	96.1769	40

Table 5.3: Invariants pour les instances hétérogènes de Christofides et Eilon

instance	$\overline{q}$	$\overline{Q}$	E	h	d_{max}	g_{max}
G-H-01	4800	3850	950	32	360	20
G-H-02	6400	5880	520	18	480	20
G-H-03	8000	7020	980	33	600	20
G-H-04	9600	8000	1600	67	720	20
G-H-05	4000	3240	760	32	600	20
G-H-06	5600	5040	560	19	600	20
G-H-07	7200	6300	900	30	600	20
G-H-08	8800	7920	880	30	600	20
G-H-09	13429	11400	2029	10	30	271
G-H-10	15195	12600	2595	15	34	275
G-H-11	16980	14000	2980	19	38	277
G-H-12	18701	15600	3101	20	42	280
G-H-13	25136	20800	4336	17	42.4264	240
G-H-14	28672	24000	4672	18	48.0833	248
G-H-15	32244	26600	5644	24	53.7401	253
G-H-16	35772	28800	6972	31	59.397	258
G-H-17	4320	3480	840	21	39.0625	30
G-H-18	5400	4360	1040	26	48.8282	30
G-H-19	6480	5240	1240	31	61.0352	30
G-H-20	7560	6160	1400	35	76.294	30

Table 5.4: Invariants pour les instances hétérogènes de Golden

5.3 Le critère d'arrêt

L'intensité de la recherche est fonction du nombre d'itérations. Celui-ci a un impact sur le temps de calcul pour exécuter l'algorithme ALNS sur chaque instance. Il est aussi évident que le nombre d'itérations pour arriver à une bonne solution dépend également du nombre de clients de cette instance.

Notre objectif est de déterminer le nombre d'itérations à partir duquel l'amélioration du coût de la fonction objectif est négligeable. Pour cela, nous partitionnons l'ensemble des instances en différentes classes selon une structure bien déterminée. Dans chaque classe nous choisissons un représentant.

Notons qu'aucune étude n'a été faite sur la structure des instances par les auteurs qui ont abordé le VRPPC. Potvin et Naud [45] choisissent aléatoirement un certain nombre d'instances comme instances de référence pour optimiser les valeurs des paramètres. Étant donné que le nombre de clients d'une instance est la notion jouant le rôle majeur dans la complexité des algorithmes de résolution, nous décidons de construire une structure de partionnement des instances se basant sur celle-ci.

Soit $\mathcal{F}$ une famille d'instances, on détermine n_{min} et n_{max} qui sont respectivement le plus petit nombre de clients et le plus grand nombre de clients des instances de $\mathcal{F}$. On détermine également n_{moy} et n_s qui sont respectivement la moyenne et l'écart type du nombre de clients des instances de $\mathcal{F}$.

Définition 5.4. *Soit $\mathcal{F}$ une famille d'instances. On appelle coefficient de stratification de $\mathcal{F}$ notée n_{st}, la partie entière du rapport entre la différence maximale du nombre de clients des instances de $\mathcal{F}$ et l'écart type du nombre de clients des instances. On a,*

$$n_{st} = \lfloor \frac{n_{max} - n_{min}}{n_s} \rfloor.$$

Définition 5.5. *Soit $\mathcal{F}$ une famille d'instances. On appelle strate tout intervalle semi-ouvert de longeur $\frac{n_s}{n_{st}}$, d'origine n_l ($n_{min} - \frac{n_s}{n_{st}} < n_l \leq n_{max}$) tel qu'il existe un entier l vérifiant $|n_l - n_{moy}| = l\frac{n_s}{n_{st}}$. Une strate est dite non vide s'il existe une instance de $\mathcal{F}$ dont le nombre de clients appartient à cette strate; elle est dite vide dans le cas contraire.*

Définition 5.6. *Soit $\mathcal{F}$ une famille d'instances. On appelle nombre stratifié de gauche noté l^- (respectivement de droite noté l^+) le nombre de strates situées à gauche (respectivement à droite) du nombre moyen de clients des instances de $\mathcal{F}$. On a,*

$$l^- = \min_{l \in \mathbb{N}} \{n_{moy} - l * \frac{n_s}{n_{st}} \leq n_{min}\}, \quad l^+ = \min_{l \in \mathbb{N}} \{n_{max} < n_{moy} + l * \frac{n_s}{n_{st}}\}.$$

Le nombre de strates est $l^+ + l^-$.

Définition 5.7. *Soit $\mathcal{F}$ une famille d'instances. On appelle stratification filtrée de $\mathcal{F}$, l'ensemble des strates non vides de $\mathcal{F}$.*

Définition 5.8. *Soit $\mathcal{F}$ une famille d'instances. Une classe de $\mathcal{F}$ est un sous-ensemble $\mathcal{C}$ d'instances, telle que toutes les instances de $\mathcal{C}$ ont un nombre de clients appartenant à une même strate.*

Définition 5.9. *Soit $\mathcal{F}$ une famille d'instances et $\mathcal{C}$ une classe de $\mathcal{F}$. On appelle représentant de $\mathcal{C}$, toute instance de $\mathcal{C}$ dont le nombre de clients est le plus proche du nombre moyen de clients des instances de $\mathcal{C}$.*

Définition 5.10. *Soit $\mathcal{F}$ une famille d'instances et $\mathcal{C}$ une classe de $\mathcal{F}$. On dit que $\mathcal{C}$ est uniforme si chacune de ses instances peut jouer le rôle de représentant de $\mathcal{C}$. On dit que $\mathcal{F}$ est uniforme si chacune de ses classes est uniforme.*

Lorsqu'une classe contient une seule instance, elle est uniforme et l'unique instance de celle-ci est son représentant. Lorsqu'elle contient plusieurs instances pouvant jouer le rôle de représentant de la classe, on choisit aléatoirement l'une d'entre elles.

5.4 Classification selon le nombre de clients (Type a)

Nous donnons dans cette section pour chacune des deux familles d'instances (CE-H et G-H) d'abord la stratification filtrée. Ensuite, nous donnons la classification correspondante et les représentants pour chacune des classes. Enfin, sur chaque représentant nous lançons l'algorithme ALNS avec un certain nombre d'itérations pour étudier la variation de la fonction objectif et estimer le temps de calcul.

5.4.1 Classification des instances de Christofides et Eilon

La famille CE-H contient 14 instances avec $n_{min} = 50$ et $n_{max} = 199$. La moyenne des clients par instance est $n_{moy} = 113.47$ avec un écart type $n_s = 47.33$. Le coefficient de stratification est $n_{st} = 3$. Des 11 strates S_i, il y a 5 qui sont vides. On en déduit exactement 6 classes d'instances pour la famille CE-H. Le tableau 5.5 donne la répartition du nombre de clients des instances appartenant à chacune de ces classes. Le tableau 5.6 donne la classification proprement dite des instances de CE-H. La famille CE-H étant uniforme, on

choisit aléatoirement une instance dans chaque classe comme son représentant. Le tableau 5.7 donne le représentant pour chaque classe de la famille CE-H.

I_a	II_a	III_a	IV_a	V_a	VI_a
50	75	100	120	150	199
50	75	100	120	150	199
		100			
		100			

Table 5.5: Stratification filtrée de CE-H

Classe					
I_a	II_a	III_a	IV_a	V_a	VI_a
CE-H-01	CE-H-02	CE-H-03	CE-H-11	CE-H-04	CE-H-05
CE-H-06	CE-H-07	CE-H-08	CE-H-13	CE-H-09	CE-H-10
		CE-H-12			
		CE-H-14			

Table 5.6: Classification des instances hétérogènes de CE-H

Classe	I_a	II_a	III_a	IV_a	V_a	VI_a
Représentant	CE-H-01	CE-H-02	CE-H-14	CE-H-11	CE-H-09	CE-H-05

Table 5.7: Choix des représentants des classes de CE-H

Tous les résultats de cette section sont obtenus en lançant la recherche sur les représentants mentionnés dans le tableau 5.7. Pour chaque classe, nous avons lancé l'algorithme ALNS sur son représentant avec un nombre d'itérations égal à 50000. Plusieurs tests ont été réalisés. On constate que l'amélioration du coût de la fonction objectif devient très faible entre les itérations 15000 et 25000 pour l'instance CE-H-01. Pour les autres représentants l'amélioration s'affaiblit entre les itérations 40000 et 50000. Par souci d'uniformité, nous décidons de lancer l'ALNS avec 50000 itérations sur CE-H-01, CE-H-02, CE-H-14, CE-H-

11, CE-H-09 et CE-H-05. Les figures 5.1, 5.2, 5.3, 5.4, 5.5 et 5.6 représentent l'évolution du coût de la solution en fonction du nombre d'itérations. Le temps de calcul moyen pour l'exécution de l'ALNS est de 45 secondes pour CE-H-01, 55 secondes pour CE-H-02, 155 secondes pour CE-H-14, 550 secondes pour CE-H-11, 470 secondes pour CE-H-09 et 933 secondes pour CE-H-05.

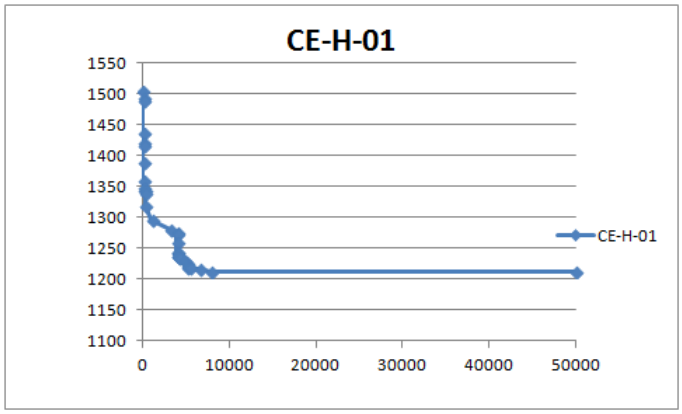

Figure 5.1: Valeur de la fonction objectif en fonction du nombre d'itérations sur CE-H-01

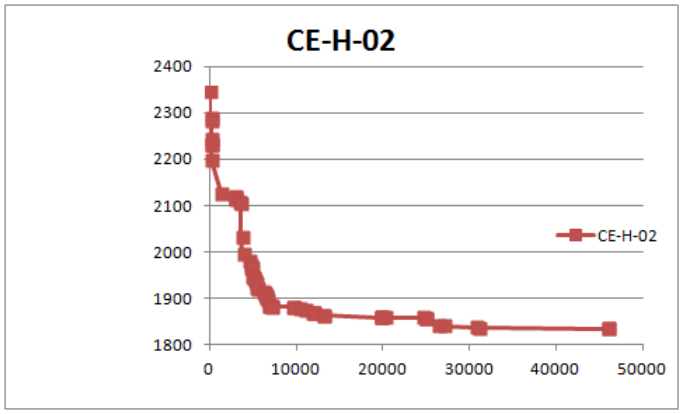

Figure 5.2: Valeur de la fonction objectif en fonction du nombre d'itérations sur CE-H-02

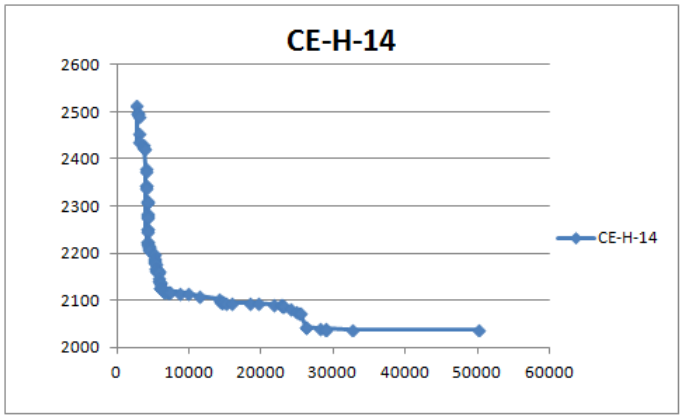

Figure 5.3: Valeur de la fonction objectif en fonction du nombre d'itérations sur CE-H-14

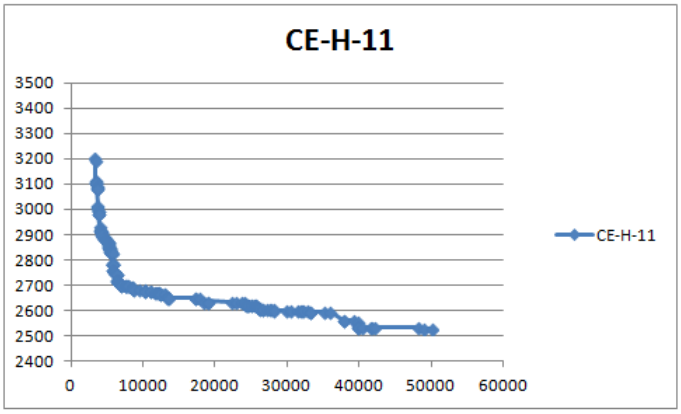

Figure 5.4: Valeur de la fonction objectif en fonction du nombre d'itérations sur CE-H-11

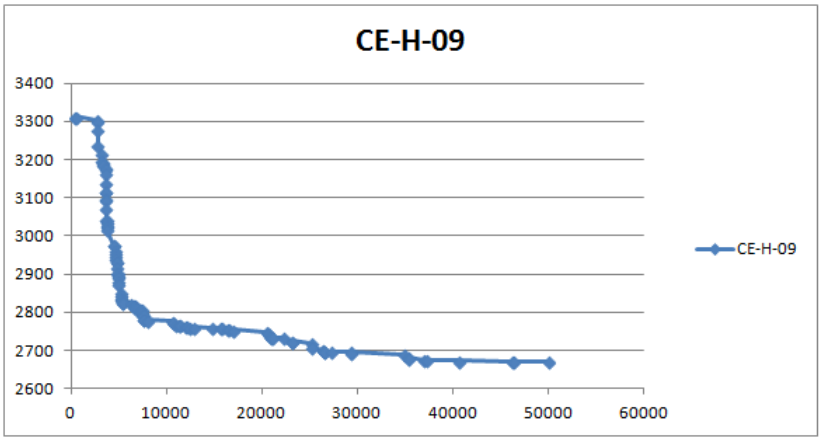

Figure 5.5: Valeur de la fonction objectif en fonction du nombre d'itérations sur CE-H-09

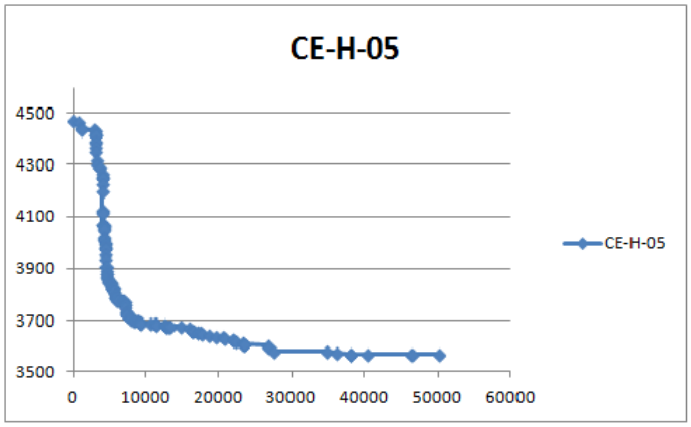

Figure 5.6: Valeur de la fonction objectif en fonction du nombre d'itérations sur CE-H-05

5.4.2 Classification des instances de Golden

La famille G-H contient 20 instances avec $n_{min} = 200$ et $n_{max} = 483$. La moyenne des clients par instance est $n_{moy} = 347.81$ avec un écart type $n_s = 87.91$. Le coefficient de stratification est $n_{st} = 3$. Des 11 strates S_i, il y a une seule qui est vide. On en déduit exactement 10 classes d'instances pour la famille G-H. Le tableau 5.8 donne la répartition du nombre de clients des instances appartenant à chacune de ces classes. Le tableau 5.9 donne la classification proprement dite des instances de G-H. Les classes I, II, IV, VIII et IX sont formées chacune d'une seule instance, celle-ci constitue son représentant. Pour les classes II et VII, le nombre moyen de clients étant 246.75 et 398.33, on en déduit que

G-H-13 et G-H-11 sont les représentants respectifs. La classe VI étant uniforme, on choisit aléatoirement une de ses instances comme représentant. Pour les classes V et X, le nombre moyen de clients est 321 et 481; dans chaque classe deux instances peuvent jouer le rôle de représentant. On choisit aléatoirement une instance pour chacune d'elle. Le tableau 5.10 donne le représentant pour chaque classe de la famille G-H.

I_a	II_a	III_a	IV_a	V_a	VI_a	VII_a	$VIII_a$	IX_a	X_a
200	240	280	300	320	360	396	420	440	480
	240			320	360	399			480
	252			323		400			483
	255								

Table 5.8: Stratification filtrée de G-H

Classe									
I_a	II_a	III_a	IV_a	V_a	VI_a	VII_a	$VIII_a$	IX_a	X_a
G-H-05	G-H-01	G-H-06	G-H-18	G-H-02	G-H-07	G-H-15	G-H-20	G-H-08	G-H-04
	G-H-17			G-H-14	G-H-19	G-H-11			G-H-16
	G-H-13			G-H-10		G-H-03			G-H-12
	G-H-09								

Table 5.9: Classification des instances hétérogènes de Golden

C.	I_a	II_a	III_a	IV_a	V_a	VI_a	VII_a	$VIII_a$	IX_a	X_a
R.	G-H-05	G-H-13	G-H-06	G-H-18	G-H-02	G-H-07	G-H-11	G-H-20	G-H-08	G-H-04

Table 5.10: Choix des représentants des classes de Golden

Tous les résultats de cette section sont obtenus en lançant la recherche sur les représentants mentionnés dans le tableau 5.10. Pour chaque classe d'instances, nous avons lancé l'algorithme ALNS sur son représentant avec un nombre d'itérations égal à 50000. Plusieurs tests ont été réalisés. On constate de façon générale que l'amélioration du coût de la fonc-

tion objectif devient très faible entre les itérations 40000 et 50000. Nous décidons de lancer l'ALNS avec 50000 itérations sur G-H-05, G-H-13, G-H-06, G-H-18, G-H-02, G-H-07, G-H-11, G-H-20, G-H-08 et G-H-04. Les figures 5.7 à 5.16 représentent l'évolution du coût de la solution en fonction du nombre d'itérations. Le temps de calcul moyen pour l'exécution de l'ALNS est de 3450 secondes pour G-H-05, 640 secondes pour G-H-13, 3101 secondes pour G-H-06, 887 secondes pour G-H-18, 6690 secondes pour G-H-02, 6002 secondes pour G-H-07, 4843 secondes pour G-H-11, 931 secondes pour G-H-20, 5900 secondes pour G-H-08 et 18924 secondes pour G-H-04.

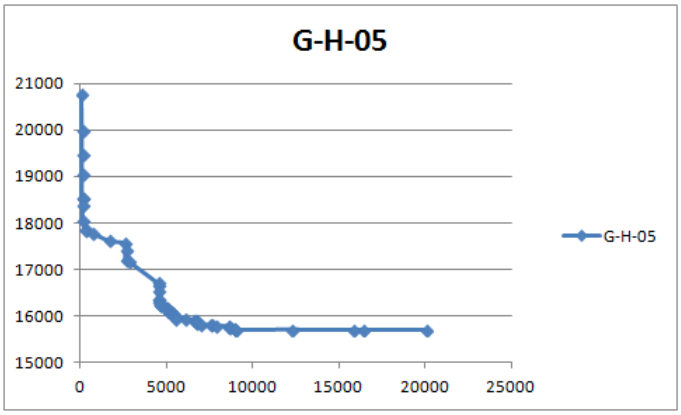

Figure 5.7: Valeur de la fonction objectif en fonction du nombre d'itérations sur G-H-05

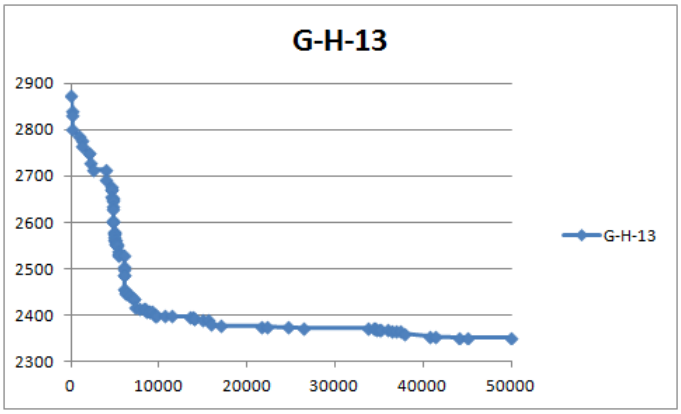

Figure 5.8: Valeur de la fonction objectif en fonction du nombre d'itérations sur G-H-13

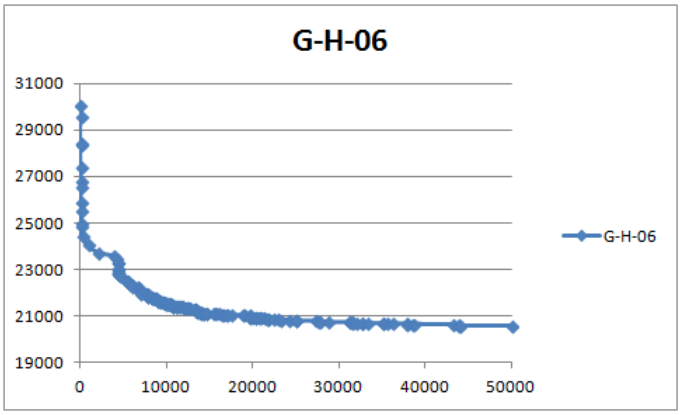

Figure 5.9: Valeur de la fonction objectif en fonction du nombre d'itérations sur G-H-06

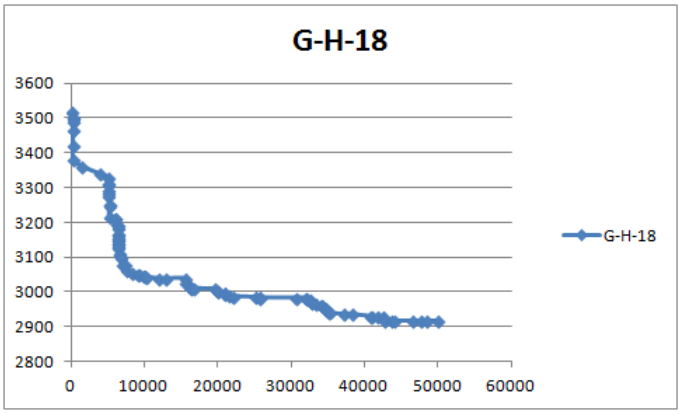

Figure 5.10: Valeur de la fonction objectif en fonction du nombre d'itérations sur G-H-18

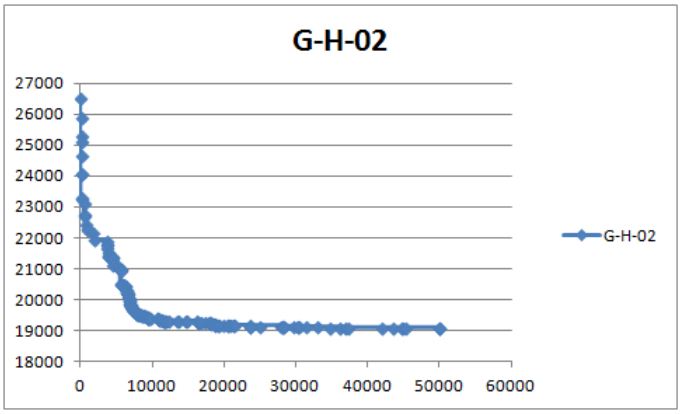

Figure 5.11: Valeur de la fonction objectif en fonction du nombre d'itérations sur G-H-02

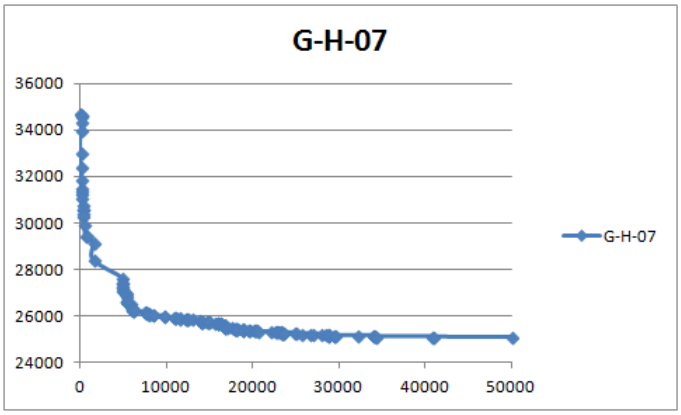

Figure 5.12: Valeur de la fonction objectif en fonction du nombre d'itérations sur G-H-07

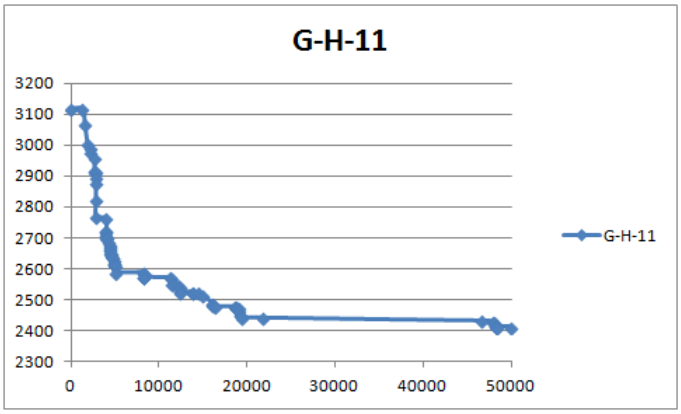

Figure 5.13: Valeur de la fonction objectif en fonction du nombre d'itérations sur G-H-11

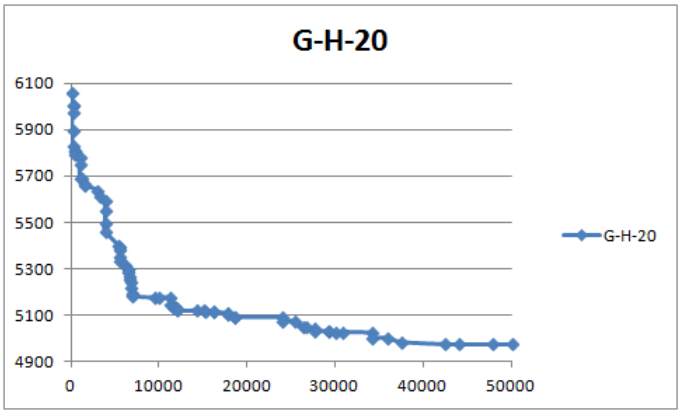

Figure 5.14: Valeur de la fonction objectif en fonction du nombre d'itérations sur G-H-20

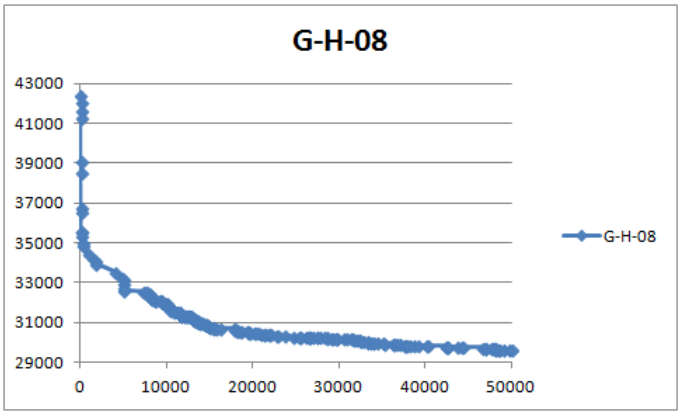

Figure 5.15: Valeur de la fonction objectif en fonction du nombre d'itérations sur G-H-08

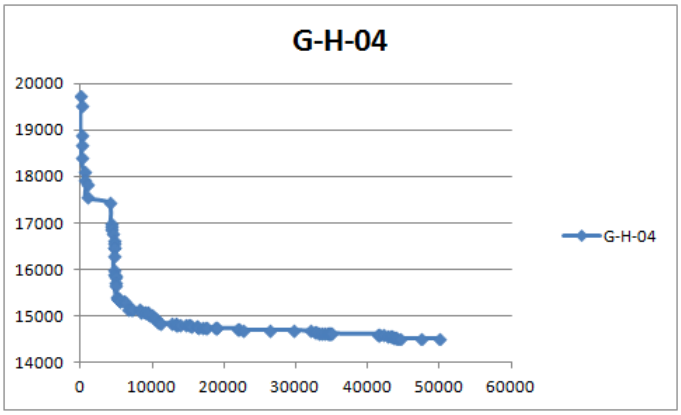

Figure 5.16: Valeur de la fonction objectif en fonction du nombre d'itérations sur G-H-04

5.5 Étude de la sensibilité des paramètres

Notre objectif est de déterminer les valeurs des paramètres conduisant aux meilleures solutions. Comme précédemment, nous partitionnons l'ensemble des instances en différentes classes selon deux caractéristiques bien déterminées: la distance maximale et l'écart maximal des instances. Dans chaque classe nous choisissons un représentant. Pour déterminer la valeur optimale d'un paramètre nous lui donnons d'abord trois types de valeurs (une petite, une moyenne et une grande) et laissons fixes les valeurs des autres paramètres. Les valeurs optimales des paramètres pour les tests réalisés sur le représentant d'une classe sont choisies pour tous les tests sur les instances de cette classe.

5.5.1 Classification selon la distance maximale (Type b)

Classification des instances de Christofides et Eilon La famille CE-H contient 14 instances avec $(d_{max})_{min} = 85.276$ et $(d_{max})_{max} = 114.978$. La distance maximale moyenne des distances est $(d_{max})_{moy} = 93.94$ avec un écart type $(d_{max})_s = 9.65$. Le coefficient de stratification est $(d_{max})_{st} = 3$. Des 10 strates S_i, il y en a 6 qui sont vides. On en déduit exactement 4 classes d'instances pour la famille CE-H. Le tableau 5.11 donne la répartition de la distance maximale des instances appartenant à chacune de ces classes. Le tableau 5.12 donne la classification proprement dite des instances de CE-H. La famille CE-H étant uniforme, on choisit aléatoirement une instance dans chaque classe comme son représentant. Le tableau 5.13 donne le représentant pour chaque classe de la famille CE-H.

I_b	II_b	III_b	IV_b
85.276	91.831	96.177	114.978
85.276	91.831	96.177	114.978
85.633	91.831		
85.633	91.831		
	91.831		
	91.831		

Table 5.11: Stratification filtrée de CE-H

Classe			
I_b	II_b	III_b	IV_b
CE-H-02	CE-H-03	CE-H-12	CE-H-11
CE-H-07	CE-H-04	CE-H-14	CE-H-13
CE-H-01	CE-H-05		
CE-H-06	CE-H-08		
	CE-H-09		
	CE-H-10		

Table 5.12: Classification des instances hétérogènes de CE-H

Classe	I_b	II_b	III_b	IV_b
Représentant	CE-H-01	CE-H-05	CE-H-12	CE-H-11

Table 5.13: Choix des représentants des classes de CE-H

Classification des instances de Golden La famille G-H contient 20 instances avec $(d_{max})_{min} = 30$ et $(d_{max})_{max} = 720$. La moyenne des distances maximales des instances est $(d_{max})_{moy} = 256.643$ avec un écart type $(d_{max})_s = 270.509$. Le coefficient de stratification est $n_{st} = 2$. Des 6 strates S_i, il y a une seule qui est vide. On en déduit exactement 5 classes d'instances pour la famille G-H. Le tableau 5.14 donne la répartition des distances maximales des instances appartenant à chacune de ces classes. Le tableau 5.15 donne la classification proprement dite des instances de G-H. Les classes II_b, III_b et V_b sont formées chacune d'une seule instance, celle-ci constitue son représentant. Pour la classe I_b, la dis-

tance maximale moyenne étant 47.739, on en déduit que G-H-14 est son représentant. La classe V_b étant uniforme, on choisit aléatoirement une de ses instances comme représentant. Le tableau 5.16 donne le représentant pour chaque classe de la famille G-H.

I_b	II_b	III_b	IV_b	V_b
30	360	480	600	720
34			600	
38			600	
39.063			600	
42			600	
42.426				
48.083				
48.828				
53.740				
59.397				
61.035				
76.294				

Table 5.14: Stratification filtrée de G-H

Classe				
I_b	II_b	III_b	IV_b	V_b
G-H-09	G-H-01	G-H-02	G-H-03	G-H-04
G-H-10			G-H-05	
G-H-11			G-H-06	
G-H-17			G-H-07	
G-H-12			G-H-08	
G-H-13				
G-H-14				
G-H-18				
G-H-15				
G-H-16				
G-H-19				
G-H-20				

Table 5.15: Classification des instances hétérogènes de Golden

C.	I_b	II_b	III_b	IV_b	V_b
R.	G-H-14	G-H-01	G-H-02	G-H-06	G-H-04

Table 5.16: Choix des représentants des classes de Golden

5.5.2 Classification selon l'écart maximal (Type c)

Classification des instances de Christofides et Eilon La famille CE-H contient 14 instances avec $(g_{max})_{min} = 33$ et $(g_{max})_{max} = 40$. La moyenne des écarts maximaux des instances est $(g_{max})_{moy} = 38.143$ avec un écart type $(g_{max})_s = 2.627$. Le coefficient de stratification est $(g_{max})_{st} = 2$. Des 7 strates S_i, il y en a 3 qui sont vides. On en déduit exactement 4 classes d'instances pour la famille CE-H. Le tableau 5.17 donne la répartition des écarts maximaux des instances appartenant à chacune des ces classes. Le tableau 5.18 donne la classification proprement dite des instances de CE-H. La famille CE-H étant uniforme, on choisit aléatoirement une instance dans chaque classe comme son représentant. Le tableau 5.19 donne le représentant pour chaque classe de la famille CE-H.

I_c	II_c	III_c	IV_c
33	36	38	40
33	36	38	40
			40
			40
			40
			40

Table 5.17: Stratification filtrée de CE-H

Classe			
I_c	II_c	III_c	IV_c
CE-H-11	CE-H-02	CE-H-01	CE-H-03
CE-H-13	CE-H-07	CE-H-06	CE-H-04
			CE-H-05
			CE-H-08
			CE-H-09
			CE-H-10
			CE-H-12
			CE-H-14

Table 5.18: Classification des instances hétérogènes de CE-H

Classe	I_c	II_c	III_c	IV_c
Représentant	CE-H-13	CE-H-07	CE-H-06	CE-H-08

Table 5.19: Choix des représentants des classes de CE-H

Classification des instances de Golden La famille G-H contient 20 instances avec $(g_{max})_{min} = 20$ et $(g_{max})_{max} = 280$. La moyenne des écarts maximaux des instances est $(g_{max})_{moy} = 119.10$ avec un écart type $(g_{max})_s = 120.738$. Le coefficient de stratification est $(g_{max})_{st} = 2$. Des 5 strates S_i, il y en a 3 qui sont vides. On en déduit exactement 2 classes d'instances pour la famille G-H. Le tableau 5.20 donne la répartition des écarts maximaux des instances appartenant à chacune des ces classes. Le tableau 5.21 donne

la classification proprement dite des instances de G-H. Les écarts maximaux moyens des deux classes I_c et II_c étant respectivement 23.333 et 262.75, on choisit aléatoirement une instance ayant pour écart maximal 20 comme représentant de la classe I_c et G-H-16 comme représentant de la classe II_c. Le tableau 5.22 donne le représentant pour chaque classe de la famille G-H.

I_c	II_c
20	240
20	248
20	253
20	258
20	271
20	275
20	277
20	280
30	
30	
30	
30	

Table 5.20: Stratification filtrée de G-H

Classe	
I_c	II_c
G-H-01	G-H-13
G-H-02	G-H-14
G-H-03	G-H-15
G-H-04	G-H-16
G-H-05	G-H-09
G-H-06	G-H-10
G-H-07	G-H-11
G-H-08	G-H-12
G-H-17	
G-H-18	
G-H-19	
G-H-20	

Table 5.21: Classification des instances hétérogènes de Golden

Classe	I_c	II_c
Représentant	G-H-06	G-H-16

Table 5.22: Choix des représentants des classes de Golden

5.5.3 Classification unifiée

Elle est bâtie sur les trois types de classification décrits précédemment. Dans chaque classse construite, toutes les instances appartiennent à la même classe pour les trois types a, b, et c. Nous avons le tableau suivant pour les instances de Christofides et Eilon:

Classe	Instance	n	d_{max}	g_{max}	$\overline{q}$
I	CE-H-01	50	85.6329	38	777
	CE-H-06	50	85.6329	38	777
II	CE-H-02	75	85.276	36	1364
	CE-H-07	75	85.276	36	1364
III	CE-H-03	100	91.8314	40	1458
	CE-H-08	100	91.8314	40	1458
IV	CE-H-12	100	96.1769	40	1810
	CE-H-14	100	96.1769	40	1810
V	CE-H-11	120	114.978	33	1375
	CE-H-13	120	114.978	33	1375
VI	CE-H-04	150	91.8314	40	2235
	CE-H-09	150	91.8314	40	2235
VII	CE-H-05	199	91.8314	40	3186
	CE-H-05	199	91.8314	40	3186

Table 5.23: Classification unifiée stricte des instances hétérogènes CE-H

Classe	Instance	n	d_{max}	g_{max}
I	G-H-05	200	600	20
II	G-H-01	240	360	20
III	G-H-17	240	39.0625	30
IV	G-H-13	252	42.4264	240
	G-H-09	255	30	271
V	G-H-06	280	600	20
VI	G-H-18	300	48.8282	30
VII	G-H-02	320	480	20
VIII	G-H-14	320	48.0833	248
	G-H-10	323	34	275
IX	G-H-07	360	600	20
X	G-H-19	360	61.0352	30
XI	G-H-15	396	53.7401	253
	G-H-11	399	38	277
XII	G-H-03	400	600	20
XIII	G-H-20	420	76.294	30
XIV	G-H-08	440	600	20
XV	G-H-04	480	720	20
XVI	G-H-16	480	59.397	258
	G-H-12	483	42	280

Table 5.24: Classification unifiée stricte des instances hétérogènes G-H

En fusionnant les classes II_b et III_b pour les instances CE-H, nous obtenons la classification suivante:

Classe	Instance	n	d_{max}	g_{max}	$\overline{q}$
I	CE-H-01	50	85.6329	38	777
	CE-H-06	50	85.6329	38	777
II	CE-H-02	75	85.276	36	1364
	CE-H-07	75	85.276	36	1364
III	CE-H-03	100	91.8314	40	1458
	CE-H-08	100	91.8314	40	1458
	CE-H-12	100	96.1769	40	1810
	CE-H-14	100	96.1769	40	1810
IV	CE-H-11	120	114.978	33	1375
	CE-H-13	120	114.978	33	1375
V	CE-H-04	150	91.8314	40	2235
	CE-H-09	150	91.8314	40	2235
VI	CE-H-05	199	91.8314	40	3186
	CE-H-05	199	91.8314	40	3186

Table 5.25: Classification unifiée large des instances hétérogènes CE-H

Pour les instances hétérogènes de Golden, nous faisons deux types de fusion: la fusion de type a et la fusion de type b. La fusion de type a consiste à fusionner:

- les classes I_a, II_a et III_a
- les classes IV_a, V_a, VI_a, VII_a et VII_a
- les classes IX_a et X_a.

La fusion de type b consiste à fusionner :

- les classes II_b et III_b
- les classes IV_b et V_b.

Nous avons la classification suivante:

Classe	Instance	n	d_{max}	g_{max}
I	G-H-05	200	600	20
	G-H-06	280	600	20
II	G-H-01	240	360	20
III	G-H-17	240	39.0625	30
IV	G-H-13	252	42.4264	240
	G-H-09	255	30	271
V	G-H-18	300	48.8282	30
	G-H-19	360	61.0352	30
	G-H-20	420	76.294	30
VI	G-H-14	320	48.0833	248
	G-H-10	323	34	275
	G-H-15	396	53.7401	253
	G-H-11	399	38	277
VII	G-H-02	320	480	20
VIII	G-H-07	360	600	20
	G-H-03	400	600	20
IX	G-H-08	440	600	20
	G-H-04	480	720	20
X	G-H-16	480	59.397	258
	G-H-12	483	42	280

Table 5.26: Classification unifiée large des instances hétérogènes G-H

5.6 Analyse de la sensibilité

Notons que tous les résultats de cette section sont obtenus en lançant la recherche sur les représentants des classes dans les tableaux 5.25 et 5.26. Les paramètres à calibrer sont: le nombre d'itérations spécifié dans le critère d'arrêt, le degré de destruction des opérateurs de destruction, le taux de refroidissement c, les coefficients d'incrémentation des scores (α, β,

γ) des opérateurs et le facteur de réaction λ. Les résultats de cette section et de la section suivantes ont été obtenus sous le système d'exploitation Linux, avec deux processeurs intel (R) Xeon X5675 de 3.07GHz.

5.6.1 Détermination du degré de destruction

En ce qui concerne les instances hétérogènes de Christofides et Eilon, la destruction aléatoire et la destruction de Shaw exigent un degré de destruction variant entre 15 à 20 % du nombre de clients pour une bonne performance. La destruction orientée véhicule, quand à elle exige un degré de destruction variant entre 20 et 40 % du nombre de clients des instances.

5.6.2 Les paramètres de mise à jour des poids

Nous avons testé plusieurs valeurs entre 0.05 et 0.99 pour le facteur de réaction λ. Nous retenons 0.10, 0.15 et 0.20 car elles conduisent à de bonnes performances des opérateurs. De même les meilleures valeurs pour le paramètre α sont 35, 30 et 25. Pour β on a 20, 15 et 10. Pour γ on a 7, 5 et 1.

5.6.3 Le taux de refroidissement

Nous avons testé plusieurs valeurs entre 0.80 et 0.99 pour le taux de refroidissement c. Nous retenons 0.89550, 0.9775 et 0.9000 car elles conduisent à de bonnes performances des opérateurs.

5.7 Résultats expérimentaux

Dans cette section, nous rapportons les résultats obtenus en appliquant l'algorithme ALNS sur les 14 instances hétérogènes de Christofides et Eilon et les 20 instances hétérogènes de Golden. Les résultats sont obtenus en lançant 20 fois l'ALNS avec 50000 itérations.

Nous rapportons dans les tableaux 5.27 et 5.30 la valeur de la solution initiale (solInit) associée à la meilleure solution, la valeur de la meilleure solution obtenue (solFinale), le taux d'amélioration de la solution initiale (tauxAm), le temps moyen d'exécution (tpsALNS),

l'itération donnant la meilleure solution (iterOr) et le nombre de véhicules utilisés (vehUtil) dans la meilleure solution. Nous avons,

$$tauxAm = 100 \times \frac{solFinale - solInit}{solInit}.$$

Nous comparons dans les tableaux 5.28, 5.29, 5.31 et 5.32 les résultats de l'algorithme ALNS aux résultats de la recherche tabou améliorée (TS+) [45] et à ceux de l'algorithme génétique (GA) [32]. Les valeurs tpsTS+ et tpsGA correspondent respectivement aux temps d'exécution des algorithmes (TS+) et (GA). De même, les valeurs gapSol_{ts+} et gapSol_{ga} correspondent respectivement au taux d'éloignement de la solution finale de l'ALNS à celles de TS+ et GA. Nous avons,

$$\text{gapSol}_{ts+} = 100 \times \frac{ALNS - TS+}{TS+} \quad \text{gapSol}_{ga} = 100 \times \frac{ALNS - GA}{GA}$$

Une valeur négative de gapSol_{ts+} ou gapSol_{ga} indique une amélioration apportée par notre ALNS.

5.7.1 Instances de Christofides et Eilon

Classe	Instance	solInit	solFinale	tauxAm	iterOr	tpsALNS	vehUtil
I	CE-H-01	1 519.220	1 191.701	21.558 %	8 367	40.159	4
	CE-H-06	1 506.109	1 204.481	20.027 %	21 562	51.350	4
moy.				20.793 %			
II	CE-H-02	2 458.512	1 797.988	26.867 %	27 949	62.930	9
	CE-H-07	2 549.140	2 034.445	20.191 %	45 130	66.953	9
moy.				23.529 %			
	CE-H-03	2 367.972	1 918.929	18.963 %	11 707	236.582	6
III	CE-H-08	2 519.437	2 005.069	20.416 %	29 266	333.525	6
	CE-H-12	2 749.742	1 918.974	30.213 %	31 454	201.742	8
	CE-H-14	2 415.464	1 908.157	21.002 %	38 061	144.726	7
moy.				22.648 %			
IV	CE-H-11	3 223.061	2 358.701	26.818 %	48 647	545.595	6
	CE-H-13	3 835.490	2 882.315	24.851 %	46 155	498.135	6
moy.				25.835 %			
V	CE-H-04	3 374.306	2 516.494	25.422 %	46 481	785.771	9
	CE-H-09	3 285.065	2 483.178	24.410 %	42 297	468.968	10
moy.				24.916 %			
VI	CE-H-05	4 310.491	3 304.410	23.340 %	43 807	779.454	14
	CE-H-10	4 284.347	3 420.407	20.165 %	48 893	932.069	13
moy.				21.753 %			

Table 5.27: Résultats globaux pour les instances hétérogènes CE-H

Du tableau 5.27, on observe que le taux d'amélioration moyen de la solution initiale obtenu par l'algorithme ALNS est de 20.79 %, 23.53 %, 22.65 %, 25.84 %, 24.92 %, 21.75 % respectivement pour les classes I, II, III, IV, V et VI. Pour l'ensemble des 14 instances avec flottes hétérogènes de Christofides et Eilon, cette moyenne est 23.16 %, ce qui est substantiel. Pour chaque classe, on constate que tous les véhicules de la flotte privée sont mis en service.

Classe	Instance	ALNS	TS+	gapSol$_{ts+}$	tpsALNS	tpsTS+
I	CE-H-01	1 191.70	1 191.70	0.00 %	40	260
	CE-H-06	1 204.48	1 206.82	-0.19 %	51	251
moy.				-0.09 %	45	256
II	CE-H-02	1 797.99	1 791.21	0.37 %	62	348
	CE-H-07	2 034.45	2 031.85	0.12 %	66	320
moy.				0.25 %	64	334
III	CE-H-03	1 918.93	1 917.96	0.05 %	236	808
	CE-H-08	2 005.07	1 986.51	0.93 %	333	845
	CE-H-12	1 918.97	1 915.05	0.20 %	201	604
	CE-H-14	1 908.16	1 907.75	0.02 %	144	672
moy.				0.30 %	229	732
IV	CE-H-11	2 358.70	2 336.51	0.94 %	545	1 339
	CE-H-13	2 882.32	2 868.13	0.49 %	498	1 365
moy.				0.72 %	521	1 352
V	CE-H-04	2 516.49	2 481.68	1.40 %	785	1 985
	CE-H-09	2 483.18	2 447.58	1.45 %	468	1 930
moy.				1.43%	626	1 957
VI	CE-H-05	3 304.41	3 143.01	5.13 %	779	3 424
	CE-H-10	3 420.41	3 272.37	4.52 %	932	3 424
moy.				4.83 %	855	3 424

Table 5.28: Comparaison avec les résultats de Potvin et Naud

Du tableau 5.28, on observe que le taux d'éloignement moyen de notre ALNS par rapport à TS+ est de -0.09 % pour la classe I, 0.25 % pour la classe II, 0.30 % pour la classe III, 0.72 % pour la classe IV, 1.43 % pour la classe V et 4.83 % pour la classe VI. On constate donc que l'écart augmente avec la taille de l'instance. Cet écart demeure toutefois fort acceptable pour les classes I à V. De plus, ALNS est plus rapide que TS+ pour toutes les classes.

Classe	Instance	ALNS	GA	gapSol_{ga}	tpsALNS	tpsGA
I	CE-H-01	1 191.70	1 203.27	-0.96 %	48.159	49.168
	CE-H-06	1 204.48	1 210.75	-0.52 %	51.350	51.531
moy.				-0.74 %	49.754	50.350
II	CE-H-02	1 797.99	1 860.84	-3.38%	62.930	94.371
	CE-H-07	2 034.45	2 108.23	-3.50 %	66.953	90.408
moy.				-3.44 %	64.942	92.390
III	CE-H-03	1 918.93	1 988.73	-3.51 %	236.582	109.335
	CE-H-08	2 005.07	2 057.75	-2.56 %	333.525	119.157
	CE-H-12	1 918.97	1 954.80	-1.83 %	201.742	109.677
	CE-H-14	1 908.16	1 988.79	-4.05 %	144.726	111.559
moy.				-2.99 %	229.144	112.432
IV	CE-H-11	2 358.70	2 381.52	-0.96 %	545.595	1 339.000
	CE-H-13	2 882.32	2 883.67	-0.05 %	498.135	1 365.000
moy.				-0.51 %	521.865	1 352.000
V	CE-H-04	2 516.49	2 622.24	-4.03 %	785.771	139.536
	CE-H-09	2 483.18	2 601.96	-4.57 %	468.968	233.717
moy.				-4.30 %	627.370	186.627
VI	CE-H-05	3 304.41	3 314.16	-0.29 %	779.454	340.371
	CE-H-10	3 420.41	3 415.50	0.00 %	932.069	382.491
moy.				-0.15 %	855.762	361.431

Table 5.29: Comparaison avec les résultats de Katrica et al.

Du tableau 5.29, on observe que le taux d'éloignement moyen de notre ALNS par rapport à GA est de -0.74 %, -3.44 %, -2.99 %, -0.51 %, -4.3 %, -0.15 % respectivement pour les classes I, II, III, IV, V et VI. On constate donc que ALNS performe mieux que GA pour toutes les classes. En effet, pour l'ensemble des 20 instances avec flottes hétérogènes de Golden, ce taux d'éloignement moyen de ALNS par rapport à GA est de -2.16 %. Au niveau du temps d'exécution, l'algorithme ALNS est plus rapide que GA pour les classes I, II, et IV, et plus lent que GA pour les classes III, V et VI.

5.7.2 Instances de Golden

Classe	Instance	solInit	solFinale	tauxAm	iterOr	tpsALNS	vehUtil
I	G-H-05	21 570.599	15 480.430	28.230 %	20 935	3 436.163	4
	G-H-06	30 327.608	20 555.904	32.220 %	44 041	3 076.908	6
moy.				30.225 %			
II	G-H-01	20 288.017	14 421.073	28.918 %	46 133	2 611.325	7
III	G-H-17	2 220.337	1 763.411	20.579 %	40 040	1 183.666	17
IV	G-H-13	2 814.690	2 351.280	16.460 %	46 155	635.657	20
	G-H-09	1 840.244	1 392.350	24.338 %	42 945	1 983.914	12
moy.				20.399 %			
	G-H-18	3 535.508	2 913.720	17.587 %	48 351	879.061	22
V	G-H-19	4 503.557	3 753.498	16.655 %	41 356	761.630	26
	G-H-20	5 965.264	4 875.449	18.269 %	38 337	928.389	32
moy.				17.504 %			
	G-H-14	3 600.911	2 906.103	19.290 %	47 985	1 360.141	24
VI	G-H-10	2 259.086	1 679.987	25.634 %	46 706	2 102.409	12
	G-H-15	4 221.552	3 471.827	17.759 %	42 482	1 840.507	26
	G-H-11	3 139.103	2 412.299	23.153 %	48 311	4 805.699	15
moy.				21.459 %			
VII	G-H-02	28 316.578	19 090.376	32.582 %	49 825	6 641.278	8
VIII	G-H-07	34 927.984	25 101.925	28.132 %	40 869	5 952.840	7
	G-H-03	39 186.115	27 224.822	30.524 %	37 558	9 078.449	8
moy.				29.328 %			
IX	G-H-08	43 550.325	29 629.829	31.964 %	49 725	5 855.142	8
	G-H-04	53 347.512	37 756.580	29.225 %	47 301	18 845.730	8
moy.				30.595 %			
X	G-H-16	5 094.826	4 090.371	19.715 %	47 281	1 666.354	29
	G-H-12	3 589.383	2 739.941	23.665 %	49 784	4 474.308	15
moy.				21.690 %			

Table 5.30: Résultats globaux pour les instances hétérogènes G-H

Du tableau 5.30, on observe que le taux d'amélioration moyen de la solution initiale obtenu par l'algorithme ALNS est de 30.23 %, 28.92 %, 20.58 %, 20.40 %, 17.50 %, 21.46 %, 32.58 %, 29.33%, 30.60 %, 21.69 % respectivement pour les classes I, II, III, IV, V, VI, VII, VIII, IX et X. Pour l'ensemble des 20 instances avec flottes hétérogènes de Golden cette moyenne est de 24.24 %, ce qui est encore substantiel. Pour chaque classe hormis la classe X, on constate que tous les véhicules de la flotte privée sont mis en service. Dans la classe il y a un véhicule qui n'est pas mis en service.

Classe	Instance	ALNS	TS+	gapSol$_{ts+}$	tpsALNS	tpsTS+
I	G-H-05	15 480.43	15 411.82	0.44 %	3 436	7 542
	G-H-06	20 555.90	19 859.30	3.50 %	3 076	19 541
moy.				1.97 %	3 256	13 541
II	G-H-01	14 421.07	14 194.13	1.59 %	2 611	11 937
III	G-H-17	1 763.41	1 664.08	5.96 %	1 183	5 226
IV	G-H-13	2 351.28	2 237.38	5.09 %	635	18 013
	G-H-09	1 392.35	1 329.27	4.74 %	1 983	18 293
moy.				4.92 %	1 309	18 153
	G-H-18	2 913.72	2 708.73	7.57 %	879	11 325
V	G-H-19	3 753.50	3 443.59	9.00 %	761	16 271
	G-H-20	4 875.45	4 314.16	13.01 %	928	29 698
moy.				9.86 %	856	19 098
	G-H-14	2 906.10	2 684.70	8.24 %	1 360	10 429
VI	G-H-10	1 679.99	1 555.59	7.99 %	2 102	15 643
	G-H-15	3 471.83	3 127.33	11.01 %	1 840	21 112
	G-H-11	2 412.30	2 195.83	9.85 %	4 805	32 079
moy.				9.27 %	2 527	19 816
VII	G-H-02	19 090.376	18 537.70	2.98 %	6 641	49 553
VIII	G-H-07	25 101.93	23 481.28	6.90 %	5 952	91 673
	G-H-03	27 224.82	25 177.92	8.12 %	9 078	119 962
moy.				7.51 %	7 515	105 817
IX	G-H-08	29 629.83	27 334.84	8.40 %	5 855	186 252
	G-H-04	37 756.58	34 991.21	7.98 %	18 845	40 791
moy.				8.19 %	12 350	113 522
X	G-H-16	4 090.37	3 621.85	12.94 %	1 666	62 174
	G-H-12	2 739.94	2 482.92	10.35 %	4 474	42 240
moy.				11.65%	3 070	52 207

Table 5.31: Comparaison avec les résultats de Potvin et Naud

Du tableau 5.31, on observe que le taux d'éloignement moyen de l'algorithme ALNS par rapport à TS+ est de 1.97 %, 1.59 %, 5.96 %, 4.92 %, 9.86 %, 9.27 %, 2.98 %, 7.51%, 8.19 %, 11.65 % respectivement pour les classes I, II, III, IV, V, VI, VII, VIII, IX et X. Il n'y a que dans les classes I, II et VII où la dominance de TS+ (en termes de la qualité des solutions) par rapport à ALNS est relativement faible. Cependant, au niveau du temps d'exécution, ALNS est plus efficace que TS+ pour toutes les classes.

Classe	Instance	ALNS	GA	gapSol_{ga}	tpsALNS	tpsGA
I	G-H-05	15 480.43	16 254.20	-4.76 %	3 436.163	889.646
	G-H-06	20 555.90	20 717.86	-0.76 %	3 076.908	5 350.663
moy.				-2.76 %	3 256.536	3 120.155
II	G-H-01	14 421.07	14 812.40	-2.64 %	2 611.325	661.852
III	G-H-17	1 763.41	1 932.18	-8.74 %	1 183.666	379.060
IV	G-H-13	2 351.28	2 330.81	0.88 %	635.657	1 256.318
	G-H-09	1 392.35	1 386.03	0.46 %	1 983.914	3 259.721
moy.				0.67 %	1 309.786	2 258.020
	G-H-18	2 913.72	3 062.08	-4.85 %	879.061	729.860
V	G-H-19	3 753.50	3 892.96	-3.58 %	761.630	1 004.026
	G-H-20	4 875.45	4 865.32	0.21 %	928.389	1 450.434
moy.				-2.83 %	856.360	1 061.440
	G-H-14	2 906.10	2 809.86	3.43 %	1 360.141	2 687.650
VI	G-H-10	1 679.99	1 622.14	3.57 %	2 102.409	8 750.368
	G-H-15	3 471.83	3 285.70	5.67 %	1 840.507	5 963.082
	G-H-11	2 412.30	2 266.04	6.45 %	4 805.699	14 759.343
moy.				4.78 %	2 527.189	8 040.111
VII	G-H-02	19 090.38	19 395.20	-1.57 %	6 641.278	4757.013
VIII	G-H-07	25 101.93	24 727.21	1.52 %	5 952.840	10 955.261
	G-H-03	27 224.82	26 523.43	2.64 %	9 078.449	14 040.338
moy.				2.08 %	7 515.645	12 497.800
IX	G-H-08	29 629.82	28 605.47	3.58 %	5 855.142	23 568.062
	G-H-04	37 756.58	36 261.53	4.12 %	18 845.730	23 744.678
moy.				3.85 %	12 350.436	23 656.370
X	G-H-16	4 090.37	3 780.43	8.20 %	1 666.354	10 786.368
	G-H-12	2 739.94	2 580.32	6.19 %	4 474.308	32 527.158
moy.				7.20 %	3 070.331	21 656.763

Table 5.32: Comparaison avec les résultats de Katrika et al.

Du tableau 5.32, on observe que le taux d'éloignement moyen de l'algorithme ALNS par rapport à GA est de -2.76 %, -2.64 %, -8.74 %, 0.67 %, -2.83 %, 4.78 %, -1.57 %, 2.08 %, 3.85 %, 7.20 % respectivement pour les classes I, II, III, IV, V, VI, VII, VIII, IX et X. On constate donc que ALNS performe mieux que GA pour les classes I, II, III, V et VII. Pour l'ensemble des 20 instances avec flottes hétérogènes de Golden, ce taux d'éloignement moyen de ALNS par rapport à GA est de 1.00 %. Cet écart demeure encore fort acceptable par rapport à GA. Au niveau du temps d'exécution, ALNS est plus rapide que GA pour les classes IV, V, VI, VIII, IX et X.

Notre étude montre que les taux d'amélioration moyens de la solution initiale par notre algorithme ALNS sur l'ensemble des 14 instances avec flottes hétérogènes de Christofides et Eilon (23.16 %), et les 20 instances avec flottes hétérogènes de Golden (24.24 %) sont considérables. L'algorithme ALNS performe bien pour la quasi-totalité des instances de la première famille (CE-H) mais il est dominé par les algorithmes TS+, IDEA et AVNS-Slow sur l'ensemble des instances de la deuxième famille (G-H). On ne peut toutefois dire avec certitude si la supériorité de ces algorithmes (en terme de qualité de la solution) pour les instances G-H est due à leur force intrinsèque ou à l'obtention préalable d'une solution initiale de meilleure qualité que la nôtre.

Chapitre 6

Conclusion

Ce travail où nous avons appliqué la métaheuristique ALNS au VRPPC sur les 14 instances hétérogènes de Christofides et Eilon, et les 20 instances hétérogènes de Golden nous permet de faire trois remarques:

- Tous les véhicules de la flotte privée sont utilisés pour servir les clients dans chaque instance, excepté le cas de la plus grande instance de Golden. Pour cette instance sur tous les tests réalisés, il y a un seul cas où un des véhicules de la flotte privée n'est pas utilisé. Ceci confirme encore la difficulté de la réduction du nombre de véhicules qui a motivé l'ajout de l'opérateur de destruction orienté véhicule.

- L'algorithme ALNS performe très bien pour les instances de Christofides et Eilon à l'exception des deux instances les plus grandes.

- En ce qui concerne les instances de Golden, dans l'ensemble nous nous éloignons des meilleurs résultats connus. La performance de l'algorithme ALNS est bonne pour les instances de taille petite, modeste pour les instances de taille moyenne et plutôt médiocre pour des instances de très grande taille.

On a pu constater qu'en faisant la post-optimisation des routes après l'application de l'algorithme ALNS, cela entraîne une amélioration modeste voir inexistante de la solution finale. Comme perspective ultérieure, nous pensons que la recherche d'opérateurs de destruction plus performants permettrait d'améliorer les résultats sur les instances de Golden.

Bibliographie

[1] C. Archetti, D. Feillet, A. Hertz, and M. G. Speranza. The capacitated team orienteering and profitable tour problems. *Journal of the Operational Research Society*, 60(6):831–842, January 2009.

[2] E. Balas. The price collecting traveling salesman problem. *Networks*, 19:621–636, 1989.

[3] M.O. Ball, B. L. Golden, and A. A. Assadand L. D. Bodin. Planning for truck fleet size in the presence of a common-carrier option. *Decision Sciences*, 14(1):103–120, January 1983.

[4] D. Bienstock, M. X. Goemans, D. Simchi-Levi, and D. Williamson. A note on the price collecting traveling salesman problem. *Mathematical Programming*, 59:413–420, 1993.

[5] M.-C. Bolduc, J. Renaud, and F. Boctor. A heuristic for the routing and carrier selection problem. *European Journal of Operational Research*, 183(2):926–932, 2007.

[6] M.-C. Bolduc, J. Renaud, F. Boctor, and G. Laporte. A pertubation metaheuristic for the vehicle routing problem with private fleet and common carriers. *Journal of the Operational Research Society*, 59:776–787, 2007.

[7] I. Chao, B. Golden, and E. A. Wasil. The team orienteering routing problem. *European Journal of Operational Research*, 88(3):464–474, 1996.

[8] N. Christofides. Worst-case analysis of a new heuristic for the traveling salesman problem. Technical Report 388, Carnegie-Mellon University (Pittsburgh, PA,, Graduate School of Industrial Administration, 1976.

[9] N. Christofides and S. Eilon. An algorithm for the vehicle-dispatching problem. *Operations Research*, 20(3):309–318, September 1969.

[10] N. Christofides, A. Mingozzi, and P. Toth. The vehicle routing problem. In *Combinatorial Optimization*, pages 315–338. Wiley, Chichester, Edited by N. Christofides, A. Mingozzi, P. Toth, C. Sandi, 1979.

[11] C.-W. Chu. A heuristic algorithm for the truckload and less-than-truckload problem. *European Journal of Operational Research*, 165:657–667, 2005.

[12] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12:568–581, 1964.

[13] T. H. Cormen, C.E. Leiseron, R. L. Rivest, and C. Stein. *Introduction à l'agorithmique*. Dunod Paris, Deuxième édition, 2004. Cours et exercices.

[14] J.-F. Côté and J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with private fleet and common carrier. *European Journal of Operational Research*, 198:464–469, 2009.

[15] M. Diaby and R. Ramesh. The distribution problem with carrier service: a dual based penalty approach. *ORSA Journal on Computing*, 7(1):24–35, Winter 1995.

[16] J. Euchi, H. Chabchoub, and A. Yassine. New evolutionary algorithm based on 2-opt local search to solve the vehicle routing problem with private fleet and common carrier. *International Journal of Applied Metaheuristic Computing*, 2(1):58–82, January-March 2011.

[17] M. M. Flood. The traveling-salesman problem. *Operations Research*, 4(1):61–75, February 1956.

[18] M. Gendreau, F. Guertin, J.-Y. Potvin, and R. Seguin. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Transportation Research Part C*, 14:157–174, 2006.

[19] M. Gendreau, G. Laporte, and J.-Y. Potvin. Metaheuristic for the capacity VRP. In *The Vehicle Routing Problem*, pages 129–154. SIAM, Edited by Paolo Toth, Daniele Vigo, 2002. Monographs on Discrete Mathematics and Applications.

[20] M. Gendreau, G. Laporte, and F. Semet. A tabu search heuristic for the undirected selective travelling salesman problem. *European Journal of Operational Research*, 106:539–545, April 1998.

[21] M. Gendreau and J.-Y. Potvin. Tabu search VRP. In *Handbook of Maetaheuristic*, pages 41–59. Springer, Edited by M. Gendreau , J.-Y. Potvin, 2010.

[22] F. Glover. Ejection chains, reference structures and alternating path methods for the traveling salesman problems. *Discrete Applied Mathematics*, 65:223–253, 1996.

[23] B. L. Golden and W. R. Stewart. Empirical analysis of heuristics. In *The traveling salesman problem: A Guide Tour of Combinatorial Optimization*, pages 207–249. Wiley, Edited by E. W. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys, 1985.

[24] B. L. Golden, E. A. Wasil, J.P. Kelly, and I.-M Chao. The impact of metaheuristics on solving the vehicle routing problem: Algorithms, Problems tests and Computational results. In *Fleet Management and Logistics*, pages 35–36. Springer US, Edited by T. G. Grainic, G. Laporte, 1998.

[25] K. Huang and C. Hsu. A Lagrangian heuristic for the vehicle routing problem with private fleet and the common carrier. *Journal of the Eastern Asia Society for Transportation Studies*, 9:644–658, 2011.

[26] S. Huijink, G. Kant, and M. Peeters. Analysis of construction methods for the vehicle routing problem with private fleet and common carrier. In *Proceedings of the ILS 2014 conference*, pages 1–8, Breda, 2014.

[27] S. Huijink, G. Kant, and R. Peeters. An adaptable variable neighborhood search for the vehicle routing problem with order outsourcing. Technical Report Vol 2014-062, Tilburg University, Tilburg: Operations Research, M.J.P (2014). CentER Discussion Paper.

[28] T. Ibaraki. Algorithms for obtaining shortest paths visiting specified nodes. *SIAM Review*, 15(2):309–317, April 1973.

[29] S. Irnich and G. Desaulniers. Shortest path problems with resource constraints. In *Column Generation*, pages 33–65. Springer, New York, Edited by G. Desaulniers, J. Desrosiers and M. M. Solomon, 2005.

[30] R. Jonker and A. Volgenant. Transforming asymmetric into symmetric travelling salesman problems. *Operations Research Letters*, 2:161–163, 1983.

[31] I. Kara, G. Laporte, and T. Bektas. A note on the lifted Miller-Tucker-Zemlin subtour elimination constraints for the capacitated vehicle routing problem. *European Journal of Operational Research*, 158:793–795, 2004.

[32] J. Katrica, T. Kostic, D. Tosic, D. Dugosija, and V. Filipovic. A genetic algorithm for the routing and carrier selection problem. *Computer Science and Information Systems*, 9:49–62, January 2012.

[33] B. W. Kernighan and D. M. Ritchie. *Le langage C Norme ANSI*. Dunod, Janvier 2000. 2^e édition.

[34] J. G. Klincewicz, H. Luss, and M. Plicher. Fleet size planning when outside carrier services are available. *Transportation Science*, 24(3):169–182, July 1990.

[35] R. V. Kulkarni and P. R. Bhave. Integer programming formulations of vehicle routing problems. *European Journal of Operational Research*, 20:58–67, 1985.

[36] G. Laporte. The vehicle routing problem: an overview of exact and approximate algorithms. *European Journal of Operational Research*, 59:345–358, 1992.

[37] G. Laporte and F. Semet. Classical heuristic for the capacitated VRP. In *The Vehicle Routing Problem*, pages 109–128. SIAM, Edited by P. Toth, D. Vigo, 2002. Monographs on Discrete Mathematics and Applications.

[38] A. M. Law. *Simulation Modeling and Analysis*. The McGraw-Hill Companies, 2007. Fourth Edition.

[39] S. Lin. Computer solutions of the traveling salesman problem. *Bell Systems Technical Journal*, 44:2245–2269, 1965.

[40] N. Mladenovic and P. Hansen. Variable neighborhood search. *Computer and Operations Research*, 24(11):1097–1100, 1997.

[41] I. Or. Traveling salesman-type combinatorial problems and their relation to the logistics of regional blood banking. In *Ph. D. Thesis*. Department of Industrial Engineering and Management Sciences, Northwestern University, Evanston, IL, 1976.

[42] I. H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41:421–451, 1993.

[43] D. Pisinger and S. Ropke. A general heuristic for the vehicle routing problems. *Computers and Operations Research*, 34:2403 –2435, 2007.

[44] D. Pisinger and S. Ropke. Large neighborhood search. In *Handbook of metaheuristics*, pages 399–419. Springer, 2010.

[45] J.-Y. Potvin and M.-A. Naud. Tabu search with ejection chains for the vehicle routing problem with private fleet and common carrier. *Journal of the Operational Research Society*, 62:326–336, 2011.

[46] J.-Y. Potvin and J.-M. Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66:331–340, 1993.

[47] J. Renaud, F. F. Boctor, and G. Laporte. A fast composite heuristic for the symmetric traveling salesman problem. *INFORMS Journal on computing*, 8(2):134–143, Spring 1996.

[48] S. Ropke and D. Pisinger. An adaptative large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, 2006.

[49] S. Ropke and D. Pisinger. A unified heuristic for the large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, 171(3):750–775, 2006.

[50] L.-M. Rousseau, M. Gendreau, and G. Pesant. Using constraint-based operators to solve the vehicle routing problem with time windows. *Journal of Heuristics*, 8:43–58, 2002.

[51] M. Sakarovitch. *Optimisation combinatoire*. Hermann Enseignement des Sciences, 1984. Méthodes mathématiques et algorithmiques. Programmation Discrète.

[52] P. Shaw. A new local search algorithm providing high quality solutions to solve vehicle routing problems. In *Technical report*. Department of Computer Science, University of Strathclyde, Scotland, 1997.

[53] P. Shaw. Using constraint programming and loacal search methods to solve vehicle routing problems. In *LNCS 1520*, pages 417–431. Springer-Verlag Berlin Heidelberg 1998, Edited by M. Maher and J.-F Puget, 1998. CP'98.

[54] A. Stenger, D. Vigo, S. Enz, and M. Schwind. An adaptative variable neighborhood search algorithm for a vehicle routing problem arising in small packages shipping codes. *Transportation Science*, 47(1):64–80, February 2013.

[55] W. R. Stewart. A computationnally efficient heuristic for the traveling salesman problem. In *Proceedings of the 13 th Annual Meeting of Southeaster TIMS*, pages 75–85. 1977. Myrtle Beach, SC, USA.

[56] E. Taillard. Parallel iterative search methods for vehicle routing problems. *Networks*, 23:661–673, 1993.

[57] P. M. Thompson and J. B. Orlin. Theory of cyclic transferts. Technical report, MIT, Cambridge, MA, Operations Research Center, 1989. Working paper.

[58] F. Tilman and T. Cain. An upper bounding algorithm for the single and multiple terminal delivery vehicle problem. *Management Science*, 18:664–682, 1972.

[59] T. Vidal, N. Maculan, L. S. Ochi, and P.H. V. Penna. Large neighbohoods with implicit customer selection for vehicle routing problems with profits. *Transportation Science*, To appear.

[60] T. Volgenant and R. Jonker. On some generalizations of the travelling-salesman problem. *Journal of the Operational Research Society*, 38(11):1073–1079, 1987.