UNIVERSITÉ DU QUÉBEC À MONTRÉAL

ALGORITHME DE SÉPARATION LOCALE POUR LE PROBLÈME DE TOURNÉES DE VÉHICULES AVEC DEMANDES STOCHASTIQUES

MÉMOIRE

PRÉSENTÉ

COMME EXIGENCE PARTIELLE

DE LA MAÎTRISE EN INFORMATIQUE DE GESTION

Par

SAHBI KHTATFA

JANVIER 2014

UNIVERSITÉ DU QUÉBEC À MONTRÉAL
Service des bibliothèques

Avertissement

La diffusion de ce mémoire se fait dans le respect des droits de son auteur, qui a signé le formulaire *Autorisation de reproduire et de diffuser un travail de recherche de cycles supérieurs* (SDU-522 – Rév.01-2006). Cette autorisation stipule que «conformément à l'article 11 du Règlement no 8 des études de cycles supérieurs, [l'auteur] concède à l'Université du Québec à Montréal une licence non exclusive d'utilisation et de publication de la totalité ou d'une partie importante de [son] travail de recherche pour des fins pédagogiques et non commerciales. Plus précisément, [l'auteur] autorise l'Université du Québec à Montréal à reproduire, diffuser, prêter, distribuer ou vendre des copies de [son] travail de recherche à des fins non commerciales sur quelque support que ce soit, y compris l'Internet. Cette licence et cette autorisation n'entraînent pas une renonciation de [la] part [de l'auteur] à [ses] droits moraux ni à [ses] droits de propriété intellectuelle. Sauf entente contraire, [l'auteur] conserve la liberté de diffuser et de commercialiser ou non ce travail dont [il] possède un exemplaire.»

REMERCIEMENTS

Le présent travail est le fruit des efforts consentis par différentes personnes. Sincèrement, il m'est impossible d'exprimer à sa juste valeur ma reconnaissance et mes remerciements à toute personne qui a apporté sa contribution et sa collaboration dans la réalisation de ce travail.

Je tiens d'abord à remercier mes directeurs de recherche, M. Walter Rei et Mme. Ola Jabali de m'avoir encadré pour réaliser ce travail et qui m'ont fait profiter de leur vaste culture, de leurs conseils et de leurs directives qui m'ont servi pour mener à bien ce projet.

Je remercie également tous les membres du centre de recherche CIRRELT où j'ai implémenté le code, et précisément M. Serge Bisaillon pour son aide.

Enfin je tiens à dire combien les soutiens quotidiens de ma famille et mes amis ont été importants tout au long de ces quelques années, je leur dois beaucoup.

TABLE DE MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

Tableau Page

LISTE DES SIGLES ET ABRÉVATIONS

Abréviation	Signification en français	Signification en anglais
AI	Approche Itérative	Iterative approch
AG	Algorithme génétique	Genetic algorithm
BC	Séparation et ajout de plans coupants	Branch and cut
BP	Génération de colonnes	Branch and Price
CNS	Composante non structurée	Unstructured vertex sets
GM	Gap moyen	Average gap
MP^g	Gap moyen pondéré	Weighted average gap
MP^t	Temps moyen pondéré	Weighted average time
PCCS	Problème du plus court chemin stochastique	Stochastic shortest-path problem
PC	Problème courant	Current problem
PDM	Processus décisionnel Markovien	Markov decision process
PLNE	Programme linéaire en nombres entiers	Linear integer program
PM	Problème maître	Master problem
PMR	Problème maître restreint	Restricted master problem
PTVDS	Problème de tournées de véhicules avec demandes stochastiques	Vehicle routing problem with stochastic demands
PTV	Problème de tournée de véhicule	Vehicle routing problem
RT	Recherche tabu	Tabu search
SL	Séparation locale	Local Branching
SP	Sous-problème	Subproblem
TM	Temps moyen	Average time

RÉSUMÉ

Le problème de tournées de véhicules avec demandes stochastiques (PTVDS) consiste à trouver des routes optimales à assigner aux véhicules et permettant de répondre à des demandes inconnues des clients dispersés géographiquement. En effectuant une route, la présence de l'incertitude au niveau des demandes des clients engendre des problèmes de rupture de stock pour les véhicules. Lorsqu'une telle situation survient, une action corrective doit-être appliquée (i.e., le véhicule fait un retour au dépôt pour recharger la capacité avant de reprendre la route). De telles actions entrainent des coûts supplémentaires et l'introduction de ces coûts dans les modèles d'optimisation (PTVDS) rendent ceux-ci beaucoup plus complexes à résoudre que les problèmes déterministes. L'objectif principal de cette recherche est de produire une heuristique efficace pour résoudre le PTVDS pour des instances de grandes tailles. L'heuristique proposée permettra de trouver des routes à coût minimum et de satisfaire toutes les contraintes exigées par le modèle.

Pour résoudre ce problème, nous utiliserons la méthode de séparation locale. Cette méthode se base sur une formulation mathématique combinatoire, dont le principe est de décomposer le problème initial en sous-problèmes par l'ajout de contraintes de séparation locale. Chaque sous-problème est par la suite résolu à l'aide d'un solveur spécialisé. Pour le cas du PTVDS, nous utiliserons comme solveur l'algorithme Integer *L*-Shaped proposé par (Jabali et *al.*, 2012). L'analyse numérique que nous avons mené illustre empiriquement que l'heuristique est en mesure d'obtenir des solutions de qualité équivalente à l'approche exacte (i.e.; des solutions optimales ou quasi-optimales) en réduisant considérablement les temps de calculs. Ainsi, notre algorithme peut résoudre plus facilement les instances qui sont difficilement résolvables par l'algorithme Integer *L*-Shaped.

MOTS CLÉS : problème des tournées de véhicules avec demandes stochastiques, heuristique, séparation locale, algorithme Integer *L*-Shaped

CHAPITRE I

INTRODUCTION

1.1 Introduction

Dans la chaine logistique, le transport est un élément indispensable du fait que les marchandises ne peuvent évidemment pas arriver au client sans avoir subi de déplacements. Sur le plan économique, la minimisation des coûts de transport, tout en respectant les délais de livraison, est un facteur qui influe sur la compétitivité des entreprises. Selon Statistique Canada en 2011, le secteur du transport représente 4,2% du produit intérieur brut, soit 53 milliards $, dont la grande partie est associée au transport par camion avec 32 %. Au centre des activités économiques de transport, nous trouvons le problème de tournées de véhicules (PTV) auquel les entreprises et les organisations sont confrontées lorsque celles-ci doivent effectuer des opérations de distribution auprès de leur clientèle. Dans sa forme la plus simple, le PTV se définit comme suit : considérant une flotte de véhicules localisée à un dépôt et un ensemble de clients à visiter dans la journée, il faut trouver un ensemble de routes à assigner aux véhicules qui permettent d'effectuer les visites nécessaires aux clients (i.e. la distribution). Ces routes débutent et se terminent au dépôt, chaque client est visité une seule fois et la demande totale moyenne des clients associés à une route ne dépasse pas la capacité d'un véhicule, tout en minimisant les coûts de transport. Des études ont démontré que lorsque ces problèmes sont négligés, cela entraine des coûts supplémentaires importants pour les entreprises et les organisations. Tel que décrit dans (Barker, 2002), plusieurs études de cas ont montré que l'application des algorithmes pour résoudre des PTV a conduit à des économie substantielles.

Dans un contexte pratique, le PTV implique bien souvent de l'incertitude au niveau des paramètres considérés. Cette incertitude provient particulièrement des demandes des clients qui doivent-être servis. Ainsi, une entreprise devant résoudre un PTV pour ses opérations de logistique, ne connaît pas toujours à l'avance les demandes de ses clients. Par conséquent, les versions stochastiques du PTV, le PTV avec demandes stochastiques (PTVDS) par exemple, apparaissent comme des problèmes intéressants à étudier dans un contexte d'applications plus réalistes. Les problèmes de remplissage d'argent aux guichets automatiques, la collecte des ordures et la distribution de bière sont des exemples représentatifs de PTVDS. Bien que l'intérêt pour le PTVDS soit indéniable, peu d'études ont portées sur le sujet. Cet état de la littérature s'explique par le fait que ces problèmes sont plus complexes à résoudre que les versions déterministes.

Depuis quelque années, on observe que la gestion du transport peut être améliorée ou facilitée par l'informatique et les solutions mathématiques. Effectivement, Toth et Vigo (2002) mentionnent que l'utilisation de méthodes informatisées dans les processus de distribution entraîne des économies allant de 5% à 20% au des coûts de transport. Par conséquent, le présent mémoire servira à étudier comment il est possible de résoudre le PTVDS à l'aide d'une méthode informatisée efficace.

1.2 Objectif de recherche

L'objectif principal de cette recherche est de produire une heuristique efficace pour résoudre le PTVDS avec plusieurs véhicules pour des instances de grandes tailles. L'heuristique proposée permettra de trouver des routes à coût minimum et de traiter l'ensemble des caractéristiques du modèle, comme l'incertitude des demandes des clients, les contraintes exigées pour les routes et le nombre limité des ressources. Il est à noter qu'il existe des méthodes exactes pour résoudre le problème de façon optimale. Par contre, ces méthodes ne sont pas efficaces que pour résoudre des

instances de tailles moyennes, ce qui fait que leur applicabilité à des situations réelles est limitée. Nous proposons donc une méthode de résolution heuristique pour résoudre des instances de grande taille. Cette méthode est basée sur l'utilisation de la stratégie de séparation locale. Pour réaliser notre objectif, nous nous inspirerons de l'algorithme développé par (Rei *et al.*, 2010) pour le cas du PTVDS avec un seul véhicule. Nous adapterons celui-ci au cas général du PTVDS pour une flotte de plusieurs véhicules. Pour ce faire, nous aurons recours à l'algorithme Integer *L*-Shaped proposé par (Jabali *et al.*, 2012) pour résoudre les sous-problèmes définis par la méthode de séparation locale.

1.3 Organisation de la recherche

Le reste de ce mémoire se divise de la façon suivante. Dans le deuxième chapitre, nous ferons la revue de littérature sur le PTVDS, en présentant les approches de modélisation et les méthodes de résolution développées qui y sont associées. Nous présenterons en détails les deux principales stratégies de résolution exacte ayant été développées pour le PTVDS : l'algorithme Integer *L*-Shaped, qui est une approche de type « Branch and cut » (BC), et l'algorithme de génération de colonnes, ou « Branch and Price » (BP), proposé par (Christiansen et Lysgaard, 2007). Nous aborderons également les heuristiques pertinentes ayant été proposées. La méthodologie employée pour atteindre nos objectifs sera expliquée dans le troisième chapitre de ce mémoire. Le quatrième chapitre inclura une analyse numérique visant à évaluer l'efficacité de la méthode de résolution proposée. Pour ce faire, une comparaison sera effectuée avec l'algorithme Integer *L*-Shaped (Jabali *et al.*, 2012). Finalement, le cinquième chapitre contient la conclusion en présente les limites et les contributions du présent mémoire.

CHAPITRE II

REVUE DE LITTÉRATURE

Le PTV est l'un des problèmes le plus étudié dans le domaine de la recherche opérationnelle. La première étude réalisée sur le sujet est due à (Dantzig et Ramser, 1959). Depuis, le problème a attiré l'attention de beaucoup de chercheurs dans la littérature académique pour deux raisons fondamentales. Dans un premier temps, le problème apparaît dans un grand nombre de situations pratiques. Dans un deuxième lieu, le problème est théoriquement intéressant et pose des défis importants pour ce qui est de sa résolution. Au cours des années, avec l'évolution des méthodes et des algorithmes d'optimisation développés dans le domaine de la recherche opérationnelle, plusieurs études sur le PTV ont porté sur le développement de méthodes de résolutions exactes. Par contre, (Vidal *et al.*, 2011) ont mentionné que pour l'instant les meilleures méthodes exactes pour le PTV sont toujours cantonnées à des problèmes de taille relativement restreinte et ils ont ajouté que les méthodes heuristiques constituent, de ce fait, un domaine de recherche très actif.

Dans ce chapitre, nous aborderons la littérature pertinente au PTVDS. Pour ce faire, nous débuterons en présentant à la section 2.1 les principales approches de modélisation pour le problème. La section 2.2 s'attardera sur les méthodes de résolutions principales qui ont été développées au fil des années. Nous présenterons dans cette section les deux principaux algorithmes exacts qui ont été développés, à savoir l'algorithme Integer *L*-Shaped «Branch and Cut» (BC) et l'algorithme de génération de colonne «*Branch and Price*» (BP). Finalement, nous présentons les heuristiques pertinentes pour le problème à la section 2.3.

2.1 Notation et approches de modélisation

Le problème peut être formulé sur un graphe $G(V, A)$, où $V = \{1,2 \ldots n\}$ est un ensemble de nœuds. Le nœud 1 représente le dépôt où est situé une flotte de m véhicules identiques de capacité D et $A = \{(v_i, v_j): v_i, v_j \in V, i < j\}$ est l'ensemble des arcs reliant l'ensemble des nœuds de V. Chaque arc (v_i, v_j) est associé à un coût c_{ij} et on suppose que ces coûts satisfont l'inégalité du triangle, i.e., $c_{ij} + c_{jk} \geq c_{ik}$ pour tous les nœuds $v_i, v_j, v_k \in V$. Les demandes des clients sont définies par des variables aléatoires non négatives $\xi_i, \forall\, i \in V\backslash\{v_1\}$. Les demandes sont indépendantes et nous considérons que les distributions de celles-ci sont connues. Elles sont observées quand le véhicule arrive chez le client. Par conséquent, le véhicule peut arriver chez un client avec une capacité insuffisante. Dans ces cas le véhicule retourne au dépôt pour effectuer un rechargement afin de continuer la route (action de recours). La fonction objective vise à minimiser la somme du coût des routes planifiées et du coût total espéré de recours pour les rechargements. La route d'un véhicule débute et finit au dépôt, chaque client est servi une seule fois par un seul véhicule. Nous assumons que si une rupture a lieu à un client donné. le véhicule doit compléter le service à celui-ci avant de poursuivre la route. La somme des demandes moyennes prévue d'une route ne dépasse pas la capacité du véhicule.

Les modèles sont différenciés en fonction de comment les décisions liées à la construction des routes et aux rechargements (retour au dépôt en cas de rupture) sont prises. Ainsi, les routes peuvent être construites de façon statique (i.e. les routes sont construites *a priori*) ou dynamique (i.e., les routes sont construites au fur et à mesure que les demandes sont observées). Pour ce qui est des décisions de rechargement, celles-ci peuvent être également statiques (i.e., prise de façon réactive lorsqu'une rupture survient ou alors selon une procédure fixe établie au préalable) ou dynamiques (i.e., prises en fonction des demandes observées aux clients). Le tableau

2.1 (Secomandi et Margot, 2009) représente toutes les principales approches de modélisation qui ont été développées.

Tableau 2.1
Classification des approches de modélisation proposées (Secomandi et Margot, 2009)

	Rechargement	
Route	Statique	Dynamique
Statique	A priori	Restockage
Dynamique	Ø	Ré-optimisation

2.1.1 L'approche a priori et restockage

Cette stratégie consiste à modéliser le problème en deux étapes. La première étape correspond à construire des routes qui visitent tous les clients une seul fois. La deuxième étape consiste à suivre les routes établies en prenant des décisions de rechargement lorsque celles-ci sont nécessaires. À ce niveau, nous distinguons deux types de rechargement, soit la stratégie de recours classique et la stratégie de restockage. Dans la stratégie de recours classique, les rechargements ne sont appliqués que lorsqu'une situation de rupture survient. Par contre, la stratégie de restockage permet de prendre des décisions de rechargement de façon préventive.

Nous utilisons le modèle de (Laporte *et al.*, 2002) pour représenter le modèle *a priori* avec recours classique. Les variables de décision sont définis de la façon suivante : x_{ij} prend la valeur 1 si l'arc (v_i, v_j) apparaît dans la solution, 0 dans le cas contraire. Nous notons par $Q(x)$ le coût total espéré de recours. Le modèle est représenté de la façon suivante :

Modèle 1 :

$$\min \sum_{i<j} c_{ij} x_{ij} + Q(x), \tag{2.1}$$

Sous les contraintes :

$$\sum_{j=2}^{n} x_{1j} = 2m, \tag{2.2}$$

$$\sum_{i<k} x_{ij} + \sum_{j>k} x_{jk} = 2, \qquad (k = 2, \dots, n), \tag{2.3}$$

$$\sum_{v_i, v_j \in S} x_{ij} \leq |S| - \left\lceil \sum_{v_i \in S} \xi_i / D \right\rceil, \qquad (S \subset V \setminus \{v_1\}; 2 \leq |S| \leq n-2), \tag{2.4}$$

$$0 \leq x_{ij} \leq 1 \qquad (2 \leq i < j < n). \tag{2.5}$$

$$0 \leq x_{1j} \leq 2 \qquad (j = 2, \dots, n), \tag{2.6}$$

$$x = (x_{ij}) \text{ Entier} \quad (1 \leq i < j \leq n). \tag{2.7}$$

La fonction objective (2.1) représente le coût total à minimiser qui est la somme des coûts associés aux routes et le coût de recours $Q(x)$. La contrainte (2.2) spécifie que chaque route commence et finit au dépôt et les contraintes (2.3) exigent que chaque client soit visité une seule fois. Les contraintes (2.4) éliminent les sous-tours et s'assurent que la demande totale moyenne des clients associés à chaque route ne dépasse pas la capacité du véhicule. Les bornes associées aux variables de décision sont imposées dans (2.5) et (2.6). Finalement, les contraintes d'intégrité associées aux variables de décision sont inclues dans (2.7).

Il est à noter que la valeur de la fonction de recours $Q(x)$ varie en fonction de l'orientation de la route. Ainsi, il faut décider *a priori* l'orientation à utiliser pour chaque route. La fonction de recours se définie donc comme suit :

$$Q(x) = \sum_{k=1}^{m} \min\{Q^{k,1}, Q^{k,2}\} \tag{2.8}$$

Nous notons par $Q^{k,\delta}$ le coût prévu de recours correspondant à la route k et l'orientation $\delta = 1$ ou 2. Pour une solution donnée $V_r = (v_{i_1} = v_1, v_{i_2}, \ldots, v_{i_{n+1}} = v_1)$, Laporte *et al.*, (2002) définissent le coût de recours de la première orientation comme suit :

$$Q^{k,1} = 2\sum_{j=2}^{n}\sum_{l=1}^{j-1} P\left(\sum_{s=2}^{j-1}\xi_{i_s} \leq lD < \sum_{s=2}^{j}\xi_{i_s}\right) c_{1i_j} \tag{2.9}$$

Le terme de probabilité est associé à l'événement aléatoire qui consiste à observer le $l^{ème}$ échec de la route au client v_{i_j}. Pour calculer $Q^{r,2}$, nous utilisons la même fonction en inversant l'orientation de la route k.

Yang *et al.* (2000) ont proposé une définition alternative du recours, soit la stratégie de restockage. Selon cette stratégie, les rechargements peuvent-être faits de façon préventive si cela entraine une réduction des coûts. Par exemple, une décision de rechargement peut-être bénéfique lorsqu'un véhicule se trouve à proximité du dépôt et que la capacité de celui-ci est presque vide. Comme dans l'approche *a priori* avec recours classique, la première étape consiste à construire des routes qui visitent l'ensemble des clients une fois. Par contre, dans la deuxième étape, ils appliquent chaque route en prenant les décisions de rechargement selon une politique définie sur des seuils spécifiques reliés à la capacité résiduelle du véhicule lorsque celui-ci quitte les clients prévus dans la route. Ainsi, les auteurs ont démontré que pour tout client j prévu dans une route, il existe une quantité h_j telle que la décision optimale de rechargement en quittant j (une fois le service complété), en supposant que la capacité résiduelle du véhicule est q. Le véhicule peut servir le prochain client $j+1$ prévu dans la route lorsque $q \geq h_j$ et il effectue un rechargement au dépôt lorsque $q < h_j$. Pour une route *a priori* donnée, les valeurs h_j définissant une

politique préventive de rechargement, sont obtenues par la programmation dynamique.

2.1.2 L'approche par ré-optimisation

Contrairement à l'approche *a priori*, l'approche par ré-optimisation considère que les routes des véhicules sont construites de façon dynamique au fur et à mesure que les demandes des clients sont observées. Cette approche repose donc sur le principe que les chauffeurs des véhicules, après avoir servi un client, doivent décider en temps réel quelle sera la prochaine visite à effectuer : un client non visitée, un client dont la demande n'a pas été complètement servie ou le dépôt pour effectuer un rechargement.

Dror *et al.* (1989) ont présenté la première formulation par ré-optimisation du PTVDS. Le problème est formulé comme un processus décisionnel Markovien (PDM). Considérant la complexité du modèle proposé, aucun résultat numérique n'est présenté dans cet article. Les auteurs se sont plutôt concentrés sur les formulations possibles pour le problème. À ce titre, cette étude a servi de point de départ pour plusieurs travaux visant à trouver des stratégies permettant l'utilisation de ces modèles en pratique.

En utilisant un seul véhicule, Secomandi (2001) formule le problème comme un problème de plus court chemin stochastique (PCCS). Nous utilisons la notation proposée par (Secomandi, 2001) pour modéliser le problème. Nous notons par ξ_i la variable aléatoire qui représente la demande du client i. Nous notons par Q la capacité du véhicule. Les distributions des probabilités de ces demandes sont discrète et connues: $\forall\, i \in V\,, p_i(k) = \Pr\{\,\xi_i = k\}$, $k = 0,1, \ldots K \leq Q$. Nous supposons que le stock de départ au dépôt est d'au moins nK unités. Soit le vecteur $x_l = (l, q_l, j_1 \ldots j_n)$ où $l \in \{0,1, \ldots, Q\}$ qui définit l'état du véhicule, $q_l \in (0,1 \ldots Q)$ est la capacité du véhicule après avoir servi le client l et j_i est la valeur associée à la quantité de la demande qui doit encore être livré au client i. La quantité exacte demandée par un

client est connue à l'arrivée du véhicule à l'emplacement pour la première fois. Seule la probabilité de distribution associée à ξ_i est connu dès le commencement. Chaque j_i peut prendre les valeurs suivantes $\{?,0,1,\dots K\}$, le cas où $j_i =?$ représente une demande inconnue (i.e., non-observée). La cardinalité de l'espace des états est représentée par $O(nQK^n)$, dont l'état de départ est $(0, D, ?, ?, \dots, ?)$ et l'état de fin est $(0, D, 0,0, \dots, ?)$. Pour chaque état x du système, il est associé un ensemble de contrôles $U(x) = \{\{m \in \{1,2,\dots,n\} \mid j_m \neq 0\} \cup \{0\}\} \times \{a | a \in \{0,1\}\}$. Le contrôle $u \in U(x)$ est défini par deux variables, soit la variable m qui définit le prochain client à visiter et la variable a. Si $a = 0$, le client m est visité directement. Si $a = 1$ il faut aller au dépôt. Le coût de transition de x vers x' sous le contrôle u est $g(x, u, x')$;

$$g(x,u,x') = \begin{cases} c_{lm} & si\ u = (m,0) \\ c_{l0} + c_{0m} & si\ u = (m,1) \end{cases}$$

La capacité de véhicule dans le nouvel état est q_m;

$$q_m = \begin{cases} \max(0, q_l - j_l) & si\ u = (m,0) \\ q_l + D - j_l & si\ u = (m,1) \end{cases}$$

Si $j_m =?$ dans l'état x, j'_m est définit comme étant une réalisation de variable aléatoire ξ_m, autrement $j'_m = j_m$. Alors les probabilités de transition sont définies de la façon suivante par :

$$p_{xx'}(u) = \begin{cases} 1 & si\ j_m\ est\ connu \\ Pr\{\xi_m = j'_m\} & si\ non \end{cases}$$

Soit la variable aléatoire N qui représente le nombre de transitions effectuées par le système, cette variable dépend des demandes aléatoires et la séquence des contrôles utilisés. Soit x_k l'état où se trouve le système à l'étape k et x_N l'état terminal. Nous définissons également une politique $p = \{u_0, u_1, \dots, u_{N-1}\}$ comme étant une séquence de contrôles appliqués aux étapes par lesquelles le système passe. Pour chaque état x_k

il existe une fonction u_k permettant d'associer un contrôle particulier à l'état considéré $u_k(x_k) \in U_k(x_k)$. La fonction objectif peut être définie comme suit :

$$J_N^p(x) = E\left[\sum_{k=0}^{N-1} g(x_k, u_k(x_k), x_{k+1}) | x_0 = x\right],$$

Pour tout état $x \in S$, où S définit l'espace des états possibles du système. L'espérance est formulée sur la base de distributions des probabilités de la chaîne markovien $\{x_0, x_1, \ldots, x_N\}$, qui à son tour dépend de l'état initial et de la politique p. Le PTVDS formulé sous l'approche par ré-optimisation est donc défini de la façon suivante :

$$J_N^*(x) = \min_p J_N^p(x).$$

Étant donné que l'égalité précédente satisfait l'équation de Bellman, nous pouvons noter :

$$J_N^*(x) = \min_{u_k \in U_k(x_k)} \sum_{x_{k+1} \in S} p_{x_k x_{k+1}}(u_k)\{g(x_k, u_k x_{k+1}) + J_N^*(x_{k+1}) | x_k = x\},$$

pour tous les états $x \in S$. Si nous connaissons $J_N^*(x)$ pour tous les états, alors un contrôle optimal à chaque état x est obtenu en effectuant la minimisation suivante :

$$u_N^*(x) = arg \min_{u_k \in U_k(x_k)} \sum_{x_{k+1} \in S} p_{x_k x_{k+1}}(u_k)\{g(x_k, u_k, x_{k+1}) + J_N^*(x_{k+1}) | x_k = x\},$$

Le PCCS est considéré comme un problème difficile à résoudre étant donné la taille de l'ensemble S. La programmation neuro-dynamique est une méthode qui a fait ses preuves pour résoudre le PCCS. Cette méthode repose sur l'utilisation de fonctions approximatives pour calculer la fonction objectif. Ces fonctions sont obtenues en utilisant des techniques de simulation et des stratégies d'approximation paramétrique.

L'approche itérative (AI) est une stratégie neuro-dynamique pour résoudre le PCCS de type heuristique.

Beaucoup d'approches de résolution se basant sur l'AI ont été proposées pour résoudre le PCCS. Par exemple : l'approche de *Rollout* proposé par (Secomandi, 2001), et l'approche par ré-optimisation partielle par (Secomandi et Margot, 2009).

En ce qui concerne l'application de cette approche, Secomandi et Margot (2009) ont signalé que les nouvelles technologies (i.e., les technologies de géo-localisation et de communication) jouent un rôle très important dans la mise-en-œuvre de modèle de type ré-optimisation. Ces technologies facilitent la communication en temps réel et permettent une mise à jour dynamique des données. C'est donc grâce à ces technologies qu'un modèle dynamique du type PTVDS par ré-optimisation peut être appliqué en pratique. En effet, ces technologies permettent à un conducteur, ayant complété le service à un client, de communiquer en temps réel avec un centre de répartition en indiquant sa position et la capacité résiduelle du véhicule. À son tour, un répartiteur pourrait lui indiquer la prochaine visite à effectuer.

Finalement, tel qu'illustré numériquement dans (Secomandi, 2000), cette approche permet de réduire considérablement les distances parcourues par les véhicules. Par contre, considérant la complexité du modèle, un effort numérique important est à prévoir pour la résolution du problème.

2.2 Les approches de résolution

La littérature propose une grande variété de stratégies pour la résolution du PTVDS. Dans cette section nous présenterons les différentes approches de résolution qui ont été développées. Nous pouvons classifier celles-ci en deux grandes catégories, soit les méthodes exactes et les heuristiques. Les méthodes exactes permettent d'obtenir la solution optimale au problème considéré au prix d'effort de calcul plus élevé. À

l'opposé, les méthodes heuristiques fournissent des solutions réalisables au problème sans garantir l'optimalité de celles-ci. Ainsi, puisque ce mémoire porte sur le modèle *a priori*, nous passerons en revue dans cette sous-section les principales approches proposées pour ce type de modèle.

2.2.1 Les méthodes exactes

Deux stratégies générales de résolution exacte ont été proposées pour le PTVDS : la stratégie de séparation et ajout de plans coupants, "*branch and cut*" et la stratégie de génération de colonnes, "*branch and price*".

L'algorithme de séparation et ajout de plans coupants, appelé la méthode Integer *L*-Shaped, a été proposé à l'origine par Laporte et Louveaux (1993) pour le PTVDS. Cette méthode est basée sur une approche de résolution par relaxation. Le modèle initial étant très complexe à résoudre, une simplification de celui-ci est obtenue en relaxant premièrement les contraintes (2.4) et (2.7) du Modèle 1 . Les contraintes d'intégrité (2.7) sont réintroduites de façon dynamique à travers le processus de séparation de l'algorithme, ce qui permet une exploration exhaustive du domaine admissible défini par les contraintes du modèle. Cette exploration prend la forme d'une liste de sous-problèmes obtenus en bornant les valeurs associées aux variables de décision. Ces sous-problèmes sont par la suite évalués de façon à obtenir des bornes (inférieures et supérieures) sur la valeur de la solution au PTVDS. L'obtention des bornes inférieures implique l'ajout de plans coupants (contraintes violées par les solutions obtenues pour les sous-problèmes) qui permettent d'améliorer la qualité de la borne obtenue. C'est à cette étape que les contraintes (2.4) sont ajoutées au modèle relaxé. La relaxation du modèle original (2.1)-(2.7) est également obtenue en remplaçant la fonction de recours par une variable θ définissant une borne inférieure sur $Q(x)$. Ainsi, la valeur de θ est définie par des inégalités valides bornant inférieurement la valeur du recours associée aux solutions obtenues pour les sous-

problèmes. Ces inégalités valides sont également ajoutées sous forme de plans coupants au sein de l'algorithme Integer L-Shaped.

Tel que présenté par (Laporte *et al.*, 2002), en supposant que L définit une borne inférieure générale sur la valeur du recours pour le PTVDS. Cette borne est calculée sur la base de la probabilité d'échec associé à chaque route prés séparément. L'algorithme Integer L-Shape peut être définit de la façon suivante :

Étape 0 : Calculer L et initialiser le compteur $t = 0$. Définir le problème courant (PC) en relaxant le modèle de PTVDS par l'élimination des contraintes (2.4) et (2.7). Remplacer la fonction $Q(x)$ par θ et introduire la contrainte $\theta \geq L$ dans le modèle relaxé. Initialiser la valeur de la meilleure solution connue $\overline{z} = \infty$. À ce niveau, le seul nœud en attente est le PC initial.

Étape 1 : Sélectionner un nœud en attente dans la liste courant, s'il n'en existe pas arrêter.

Étape 2 : Fixer $t = t + 1$ et résoudre le sous-problème, soit (x^t, θ^t) la solution optimale.

Étape 3 : Vérifier si des contraintes (2.4) sont violées et les générer en conséquence. À cette étape des inégalités valides ou des inégalités valides de type LBF pour traiter les routes partielles (nous définissons cette notion plus tard dans cette section) peuvent être générées. Si des contraintes violées sont trouvées, les ajouter au PC et retourner à l'**Étape 2.** Si $cx^t + \theta^t \geq \bar{z}$, éliminer le nœud courant et retourner à l'**Étape 1.**

Étape 4 : Si la solution n'est pas entière, choisir une variable fractionnelle et séparer le domaine réalisable de sous-problème en utilisant cette variable (de nouveaux sous-problèmes sont ainsi créés). Ajouter les nouveaux sous problèmes à la liste courante et retourner à l'**étape 1.**

Étape 5 : Calculer $Q(x^t)$ et $z^t := cx^t + Q(x^t)$ si $z^t < \bar{z}$, fixé $\bar{z} = z^t$.

Étape 6 : Si $\theta^t \geq Q(x^t)$ éliminer le nœud courant et retourner à l'**Ètape 1.** Sinon, imposer la contrainte suivante au modèle relaxé :

$$\sum_{\substack{1<i<j \\ x_{ij}^t=1}} x_{ij} \leq \sum_{1<i<j} x_{ij}^t - 1$$

Cette contrainte permet d'éliminer la solution courante (x^v, θ^v). Retourner à l'**Ètape 2.**

C'est à l'Étape 3 de l'algorithme Integer L-Shaped que les plans coupants sont ajoutés à la relaxation de telle sorte à améliorer la qualité de la borne inférieure associée au sous-problème courant traité. Tel qu'indiqué précédemment, deux types de contraintes sont ajoutées à cette étape, les contraintes de capacité et des inégalités valides bornant inférieurement la valeur de θ. L'ajout de contraintes (2.4) suit les méthodes développées dans le cadre de la résolution de modèle déterministe (Lysgaard *et al.*, 2004). La particularité dans l'application de ces méthodes pour le PTVDS est que celles-ci sont trouvées en considérant les demandes moyennes associées aux clients.

Pour ce qui est des inégalités valides qui sont ajoutées pour améliorer la borne inférieure obtenue associée à θ. Ces inégalités sont définies comme des fonctions bornant inférieurement la valeur du recours pour des routes partielles.

Le concept d'inégalités valides définies sur des routes partielles a été proposé originalement (Hjorring et Holt, 1999) pour le cas d'un seul véhicule. À partir d'une solution non-entière obtenue à l'Ètape 2 de l'algorithme Integer L-Shaped, considérant le sous-graphe induit par les valeurs positives de cette solution, une route partielle h est un ensemble de clients assignés à un véhicule pour lesquels la route de ceux-ci est partiellement définie. Tel qu'illustré à la Figure 2.2, une route partielle

débute et se termine au dépôt et elle est constituée d'une suite de composantes connexes de deux types : des chaînes ou des composantes non-structurés (CNS). Les chaînes représentent des séquences de visites alors que les CNS sont des ensembles de clients dont les séquences de visites n'ont pas encore été fixées. Finalement, les composantes d'une route partielle sont reliées par des nœuds d'articulation.

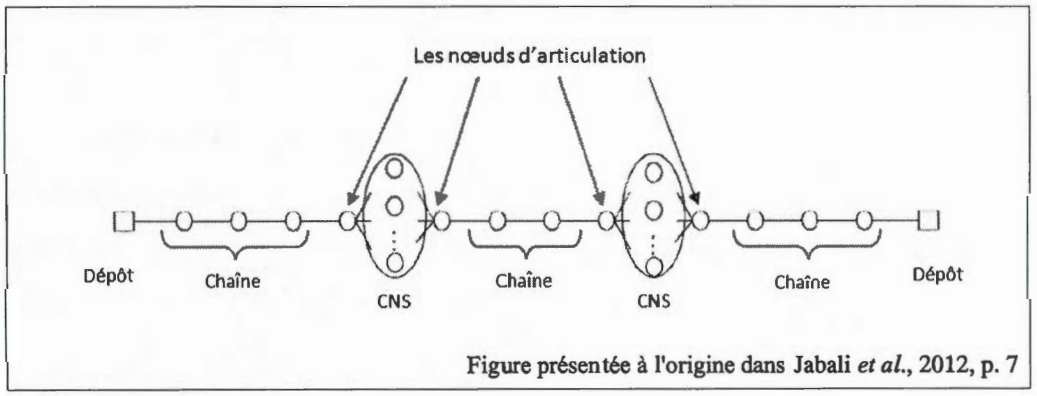


Figure présentée à l'origine dans Jabali *et al.*, 2012, p. 7

Figure 2.1 Une route partielle

Pour obtenir des inégalités valides basées sur des routes partielles, il fut proposé dans (Hjorring et Holt, 1999) de borner le recours associé à ce type de routes en considérant que celles-ci comprennent toujours trois composantes : une chaine au début de la route, une chaîne à la fin de la route et un CNS entre les deux. Il fut démontré par les auteurs qu'en agrégeant les clients inclus dans le CNS, il était possible d'obtenir une borne valide sur le recours en évaluant simplement $Q(x)$ pour la route agrégée. Une inégalité valide a été développée pour borner inférieurement toute solution dans laquelle la route partielle se retrouve dans le sous-graphe induit. Les auteurs ont proposé ces inégalités dans le cas du PTVDS à un seul véhicule et ils ont démontré numériquement la grande efficacité de celles-ci pour résoudre le problème.

Par la suite, ces inégalités ont été appliquées au cas du PTVDS à plusieurs véhicules par (Laporte et *al.*, 2002). Les auteurs ont également proposé un algorithme heuristique permettant de trouver des inégalités violées à partir de solutions non-entières. Encore une fois, l'analyse numérique a démontrée l'efficacité de ces inégalités pour résoudre le problème.

Récemment, une généralisation de ce type d'inégalité a été proposée par par Jabali *et al.*, (2012). Les auteurs définissent un ensemble d'inégalités fondées sur des typologies différentes pour les tournées partielles. Ces typologies utilisent les structures particulières des composantes d'une tournée partielle pour obtenir des bornes inférieures différentes pour les solutions ayant la tournée partielle. Des inégalités valides ont été par la suite définies en fonction des différentes bornes obtenues. L'analyse numérique menée par les auteurs a démontré qu'en diversifiant les inégalités valides ajoutées il est possible de résoudre des instances du PTVDS de plus grande taille. À ce jour, l'implantation la plus efficace de l'algorithme Integer *L*-Shaped pour le PTVDS est celle développée dans (Jabali *et al.*, 2012).

Christiansen et Lysgaard (2007) ont proposé un algorithme de type génération de colonnes (BP) pour le PTVDS. Pour ce faire, ils ont modélisé le problème à l'aide d'une formulation route .Soit le paramètre α_{ir} prenant la valeur 1 si la route r visite le client i, 0 sinon. Considérons $\Re_e$ comme étant l'ensemble des routes élémentaires. Pour $r \in \Re_e$, soit x_r une variable de décision prenant la valeur 1 si la route r est choisie et 0 sinon. Soit le nœud 0 représente le dépôt et l'ensemble de nœuds $V_c = \{1, \dots, n\}$ représente les clients. Les auteurs définissent donc le problème (P_{org}) de la façon suivante :

P_{org} :

$$\min \sum_{r \in \Re_e} c_r x_r \tag{2.10}$$

$$s.l.c. \qquad \sum_{\Re_e} \alpha_{ir} x_r = 1 \quad \forall i \in V_c, \tag{2.11}$$

$$x_r \in \{0,1\} \forall r \in \Re_e \tag{2.12}$$

La fonction objectif (2.10) permet de minimiser le cout de distribution. Les contraintes (2.11) exigent que chaque client soit visité par une seule route et les contrainte (2.12) garantissent que les variables décisionnelles x_r soient binaires.

Le BP se base sur la décomposition de Dantzing et Wolf (1960). En appliquant cette stratégie, le problème original est décomposé en deux modèles : le problème maître (PM) et le sous-problème (SP). Le PM est défini comme une restriction du problème original (i.e., un modèle considérant un sous-ensemble des variables de décision) alors que le SP a pour but de trouver, à partir de solution au problème maître, des variables à inclure dans le problème maître qui permettent d'améliorer la solution de celui-ci. Nous décrirons dans ce qui suit les grandes lignes de l'algorithme proposé par Christiansen et Lysgaard (2007).

Le problème original contient un très grand nombre de variables décisionnelles. L'énumération complète de ces variables *a priori* n'est pas envisageable du point de vue computationnel. Ainsi, pour résoudre le problème, les auteurs restreignent l'ensemble des variables de décision en utilisant un sous-ensemble de celles-ci. Les contraintes d'intégrité (2.12) sont également relaxées de façon à obtenir un problème plus simple à résoudre. Celles-ci seront réintroduites de façon dynamique à travers un

processus de séparation (i.e., génération d'un arbre de branchement) permettant l'énumération des solutions entières. Un problème maître restreint (PMR) relaxé est donc obtenu :

PMR :

$$\min \sum_{r \in \Re} c_r x_r \tag{2.13}$$

$$s.l.c. \qquad \sum_{\Re} a_{ir} x_r = 1 \quad \forall i \in V_c, \tag{2.14}$$

$$x_r \geq 0 \quad \forall r \in \Re. \tag{2.15}$$

Il est à noter que les contraintes (2.12) du modèle précédent sont remplacées par les contraintes (2.15), où $\Re \subset \Re_e$.

Le SP utilisé par l'algorithme sert quant à lui à générer des routes n'étant pas présentes dans la formulation du PMR, mais permettant d'améliorer la solution courante de celui-ci (i.e., une route dont le coût réduit est strictement négatif). Ainsi, en utilisant les variables duales associées à la solution optimale au modèle (2.13)-(2.15), qui est défini à un nœud de l'arbre de branchement, le modèle d'optimisation définissant le sous-problème est initialisé. Dans le cas du PTVDS, ce modèle prend la forme d'un modèle de plus court chemin avec consommation de ressources. Tel que développé par Christiansen et Lysgaard (2007), la particularité dans le présent cas est que le coût espéré de recours encouru à un client visité dans une route peut être traité comme une ressource consommée le long du chemin menant au client considéré. Par conséquent, les algorithmes de programmation dynamique traditionnellement utilisés pour résoudre ce type de problème de plus court chemin peuvent être appliqués dans le cas du PTVDS. Pour plus de détails, le lecteur est référé à l'article de Christiansen et Lysgaard (2007).

Ci-dessous nous présentons le pseudo-code de l'algorithme BP :

Étape 1 : Construire le problème maître restreint sous le sous-ensemble $\mathfrak{R}^t \subseteq \mathfrak{R}$, où $\mathfrak{R}$ est l'ensemble des routes réalisables et $\mathfrak{R}^t$ un sous-ensemble des routes réalisables dans l'itération t).

Étape 2 : Résoudre le problème maître restreint relaxé.

Étape 3 : Résoudre le SP à l'itération t en utilisant les valeurs de variable duales de PMR^t.

Étape 4 : Si le SP génère des colonnes avec coût réduit positif, alors ajouter les colonnes à $\mathfrak{R}^t$ et incrémenter le compteur des itérations $t = t + 1$ et retourner à l'Étape 2.

Étape 5 : Vérifier si $x^t \in \mathbb{N}$. Si oui, alors arrêter et $z^* = z^t$. Sinon brancher de nouveau en ajoutant les contraintes violées et retourner à l'Étape 2.

L'analyse numérique menée par Christiansen et Lysgaard (2007) a démontré l'efficacité de la stratégie BP pour résoudre des instances impliquant un nombre élevé de véhicules. Par contre, cette méthode atteint ses limites lorsque la taille des routes est élevée. Ceci s'explique par le fait que les sous-problèmes deviennent de plus en plus difficiles à résoudre au fur et à mesure que le nombre de clients dans les routes augmente. Ainsi, dans ce cas, l'algorithme Integer L-Shaped est plus approprié, voir Jabali *et al.*, (2012).

2.2.2 Les méthodes heuristiques

Des heuristiques ont également été développées pour résoudre des instances du PTVDS impliquant un grand nombre de clients. Ces méthodes visent à obtenir des solutions de bonne qualité dans un temps raisonnable. Contrairement aux algorithmes exacts, les heuristiques ne garantissent pas l'optimalité.

Gendreau *et al.*, (1996) ont proposé l'algorithme TABUSTOCH qui se base sur la méthode de recherche avec tabou (RT). L'algorithme traite le problème de tournées

de véhicules avec clients et demandes stochastiques. À partir d'une solution initiale, l'algorithme cherche dans son voisinage une solution différente de meilleure qualité. Le voisinage considéré est défini sur la base de mouvements de clients dans les routes associées à la solution courante. L'originalité de l'algorithme réside dans l'utilisation d'une fonction alternative permettant de mesurer la variation entre le coût espéré du recours de la solution courante et celui associé à une solution voisine. L'analyse numérique effectuée par les auteurs a démontré que sur une liste de 825 instances, l'heuristique obtenait des solutions dont la valeur de la fonction se trouvait en moyenne à 0,38% de l'optimum.

Yang *et al.*, (2000) ont proposé deux heuristiques pour résoudre le PTVDS avec la stratégie de restockage. La première heuristique débute en construisant une route visitant l'ensemble des clients. Cette route est par la suite divisée de telle sorte à produire des sous-routes réalisables pour les véhicules disponibles. La deuxième heuristique débute plutôt en créant des sous-ensembles de clients qui seront assignés aux véhicules. Par la suite, une route est construite pour chaque sous-ensemble obtenu. Avec ces heuristiques, les auteurs résolvent des instances dont la taille varie entre 10 et 60 clients.

Chepuri et Homem-De-Mello (2005) ont proposé des heuristiques probabilistes pour résoudre le PTVDS. Cependant, il est important de noter que ces auteurs traitent un modèle légèrement différent des précédents. Ainsi, considérant une route *a priori*, il est possible de laisser tomber un client en contrepartie d'un dédommagement. Une pénalité supplémentaire est donc ajoutée à la fonction de recours pour tenir compte des clients n'ayant pas été servis.

Ismail et Irhmah (2008a) ont proposé un algorithme génétique (AG) pour le PTVDS à plusieurs véhicules. Les algorithmes AG sont des procédures évolutionnistes dont l'objectif est d'améliorer de façon itérative une population de solutions au problème considéré. L'algorithme proposé par les auteurs comprend trois opérateurs

principaux : la sélection, servant à identifier les solutions à croiser (i.e., les solutions parents); le croisement, servant à produire une solution enfant à partir de deux solutions parents; et la mutation, servant à appliquer des modifications aléatoires aux solutions enfants obtenues. Dans ce cas, l'opérateur de sélection est basé sur l'application d'une roulette aléatoire où la probabilité de sélection de chaque solution est proportionnelle au coût de celle-ci. L'opérateur de croisement construit les solutions enfants en échangeant des chaines tirées des routes associées aux solutions parents. Finalement, l'opérateur de mutation applique des modifications aléatoires aux solutions enfants de façon à diversifier la population courante. Cet opérateur est modulé en fonction du niveau de diversité associé aux solutions dans la population courante (i.e., la mutation augmente lorsque la diversité diminue), voir Ismail et Irhmah (2008a). Avec cette méthode, les auteurs ont résolu des instances allant de 10 à 50 clients.

Ces auteurs ont aussi proposé une version améliorée de leur algorithme, voir Ismail et Irhmah (2008b). Cette nouvelle version, de type hybride, combine le AG avec une méthode de type RT. L'algorithme développé peut-être considéré comme une stratégie de type mémétique, où la procédure RT remplace l'opérateur de mutation. L'analyse numérique menée par les auteurs démontre que ce nouvel algorithme performe mieux que le AG.

Finalement, Rei *et al.*, (2010) ont proposé un algorithme combinant la méthode de séparation locale (SL) et la méthode de simulation Monte Carlo. Cet algorithme utilise à la fois la simulation de Monte Carlo pour réduire la complexité associée à l'évaluation de la fonction de recours et la recherche de type SL pour explorer de façon efficace le domaine réalisable du problème. Cette stratégie a fait ses preuves sur le cas du PTVDS à un seul véhicule. Au prochain chapitre, nous décrirons comment appliquer cette méthode au cas général du PTVDS à plusieurs véhicules.

CHAPITRE III

MÉTHODE DE RÉSOLUTION

L'objectif de ce mémoire est de proposer un algorithme efficace pour résoudre le PVTDS pour le cas où plusieurs véhicules sont utilisés. Pour résoudre ce problème, nous généralisons l'algorithme développé par Rei *et al.*, (2010). Tel qu'indiqué précédemment, cet algorithme est basé sur la méthode SL. Cette méthode a été proposée par Fischetti et Lodi (2003) pour résoudre des problèmes d'optimisation combinatoire pouvant être formulés comme un programme linéaire en nombres entiers (PLNE). En partant de la formulation mathématique du problème, l'algorithme décompose celui-ci en sous-problèmes en ajoutant des contraintes de type SL. Chaque sous-problème est par la suite résolu à l'aide d'un solveur adapté au PLNE considéré. Dans le cas présent, puisque le problème à résoudre est un PTVDS à plusieurs véhicules, le solveur qui est utilisé est l'algorithme Integer *L*-Shaped développé par Jabali *et al.*, (2012). Dans les prochaines sous-sections, nous illustrerons comment ce solveur est utilisé dans le cadre de la méthode SL proposée.

La méthode de résolution que nous généralisons est basée sur une structure multi-descentes de type SL. Pour présenter clairement celle-ci, nous aborderons dans les deux prochaines sections de ce chapitre les principales stratégies sur lesquelles repose l'algorithme. Ainsi, nous présenterons premièrement la stratégie d'intensification définissant la façon dont chaque descente de l'algorithme est effectuée. Par la suite, nous présenterons la stratégie de diversification déterminant le processus itératif par lequel l'algorithme passe d'une descente à une autre. Finalement, nous conclurons ce chapitre en décrivant le pseudo-code de la méthode de résolution développée.

3.1 La stratégie d'intensification

La stratégie d'intensification est illustrée par la Figure 3.1, le triangle marqué par la lettre T représente le solveur PLNE, soit l'algorithme Integer L-Shaped de Jabali *et al.* (2012), résolvant un sous-problème à l'optimalité. Afin de simplifier la description de la méthode, nous remplaçons notre modèle (2.1)-(2.7) par le modèle suivant, dans lequel x représente un vecteur de taille n_1, où $n_1 = |A|$:

$$min\, c^T x + Q(x) \tag{3.1}$$

$$x \in X. \tag{3.2}$$

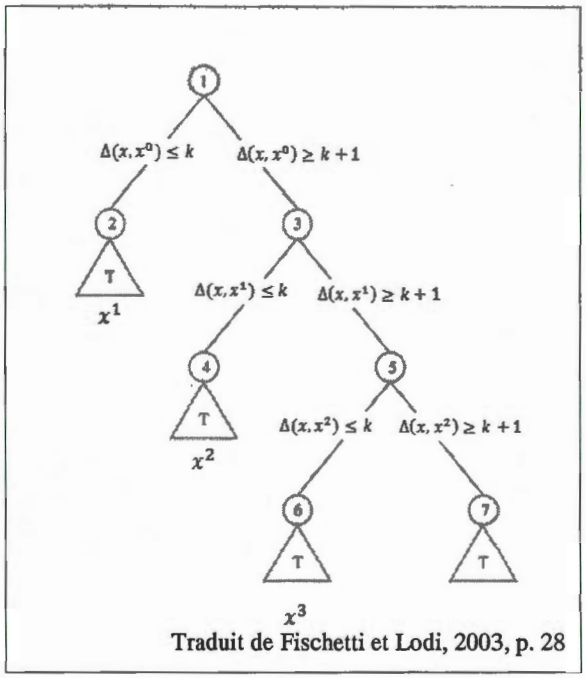
Soit $x^0 \in X$ une solution réalisable, tel que $S_0 = \{j \in N_1 | x^0 = 1\}$ et $N_1 = \{1, \dots, n_1\}$. Nous définissons également la distance de Hamming entre x et x^0 par la fonction (3.3).

$$\Delta(x, x^0) = \sum_{j \in S_0} (1 - x_j) + \sum_{j \in N_0 \setminus S_0} x_j \tag{3.3}$$

Considérant deux vecteurs de même taille, la distance de Hamming permet d'obtenir le nombre des composantes où ces deux vecteurs diffèrent. Soit κ une valeur entière, la fonction $\Delta(x, x^0)$ permet une séparation locale de l'espace des solutions au problème (3.1)-(3.2) en considérant la disjonction suivante :

$\Delta(x, x^0) \leq \kappa$ (Séparation à gauche) $\quad \Delta(x, x^0) \geq \kappa + 1$ (Séparation à droite)

En imposant la séparation à gauche, le solveur est restreint à explorer un voisinage de la solution de départ x^0 (i.e., toute solution se trouvant à une distance de Hamming d'au plus κ de la solution x^0). Pour forcer le solveur à trouver une solution différente de x^0, nous ajoutons également la contrainte $\Delta(x, x^0) \geq 1$ au modèle.



Traduit de Fischetti et Lodi, 2003, p. 28

Figure 3.1 Le schéma de base représentant la séparation locale

Lorsque la résolution du sous-problème défini par la séparation à gauche est complété, l'algorithme poursuit la recherche à partir de la zone admissible inexplorée. Cette zone inexplorée est définie par la séparation à droite. La séparation à droite regroupe les solutions réalisables se trouvant à une distance de Hamming d'au moins κ +1 de la solution x^0, comme le montre la Figure 3.2. Le grand cercle représente le domaine réalisable associé au problème original alors que le petit cercle est le domaine réalisable définit par l'ajout des contraintes de séparation à gauche $\Delta(x, x^0) \leq \kappa$ et $\Delta(x, x^0) \geq 1$. Le domaine réalisable restant est définit par l'ajout de la contrainte de séparation à droite $\Delta(x, x^0) \geq \kappa + 1$.

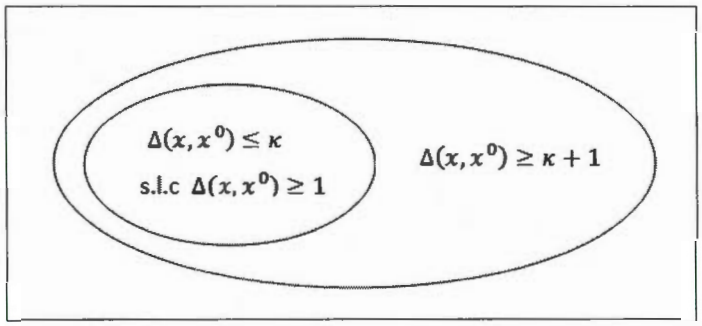


Figure 3.2 Séparation locale (première niveau)

En supposant que x^1 est la solution optimale obtenue pour le sous-problème défini par la première séparation à gauche.

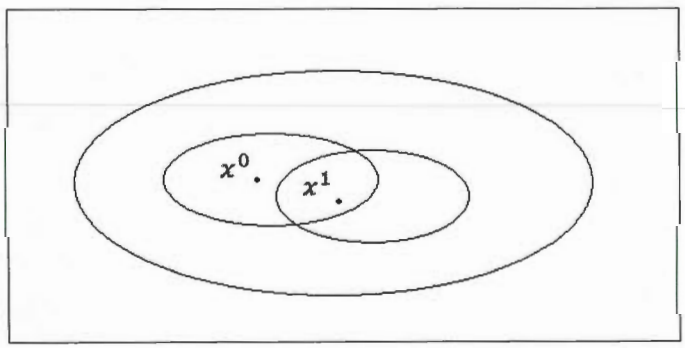


Figure 3.3 Séparation locale (deuxième niveau)

Dans le deuxième niveau, le solveur explore le voisinage de la nouvelle solution x^1 avec l'ajout de la nouvelle contrainte de séparation à gauche $\Delta(x, x^1) \leq \kappa$, comme le montre la Figure 3.3. Ce processus sera répété pour un nombre prédéfini de niveaux, soit p la profondeur de l'arbre de séparation. La descente s'arrête lorsque le nombre de niveaux explorés atteint p. Par exemple, p=3 dans la Figure 3.1.

Le modèle (3.4)-(3.8) représente le sous-problème résolu par le solveur à chaque niveau t. L'objectif (3.4) et les contraintes (3.8) représentent le problème stochastique original. Les contraintes (3.5), (3.6) et (3.7) représentent les contraintes de SL, où I^t représente l'ensemble des solutions trouvées dans un niveau t (i.e. dans la figure (3.1) au deuxième niveau $I^2 = \{x^1, x^2\}$).

$$SP_t : \quad \min c^T x + Q(x), \tag{3.4}$$

$$\text{s.l.c} \quad \Delta(x, x^i) \geq \kappa + 1, \quad \forall x^i \in I^t, \tag{3.5}$$

$$\Delta(x, x^t) \leq \kappa, \tag{3.6}$$

$$\Delta(x, x^t) \geq 1, \tag{3.7}$$

$$x \in X \tag{3.8}$$

3.2 La stratégie de diversification

Nous réutilisons la même stratégie de diversification que celle proposée par Rei *et al.* (2010). Cette stratégie définie le processus itératif par lequel la méthode passe d'une descente à une autre. L'objectif général de cette stratégie est d'explorer au maximum l'espace réalisable du problème. À chaque fois qu'une descente débute, une solution de départ est requise. Par conséquent, pour obtenir cette solution, le modèle suivant est résolu :

$$\min c^T x, \tag{3.9}$$

$$\text{s.l.c} \quad \Delta\left(x, x^{l^i}\right) \geq \kappa + 1$$

$$i = 1, \dots, p-1; l = 1, \dots, d \tag{3.10}$$

$$x \in X \tag{3.11}$$

Nous notons par d le nombre de descentes ayant été effectuées, (3.9) correspond à la fonction objectif déterministe et (3.10) correspondent aux séparations à droites ayant été définies pour les d premières descentes effectuées par l'algorithme. La solution optimale au problème (3.9)-(3.11) est donc la meilleure en considérant uniquement le coût déterministe. Dans le cas présent, cela revient à résoudre le PTV déterministe. Ainsi, une descente débute à partir d'une solution dont les coûts des tournées *a priori* sont optimaux. Le coût du recours est par la suite considéré dans le cadre de la descente. En réalisant une descente, il est donc possible d'améliorer localement les solutions de départ.

Nous terminerons ce chapitre en présentant le pseudo-code de l'algorithme, originalement développé par Rei *et al.* (2010) pour le PTVDS à un seul véhicule. L'algorithme met en œuvre les stratégies d'intensification et de diversification décrites dans ce chapitre.

Notation

l : le compteur de nombre de descentes

i : le compteur de nombre des profondeurs

$\overline{z}$: la valeur de la solution final

$\hat{x}$: une solution réalisable

$\tilde{x}^l$: la meilleure solution dans une descente l

$\overline{x}$: la solution finale

X^l: le domaine des solutions réalisables du problème déterministe à la descente l

$\hat{X}$: le domaine des solutions réalisables du problème stochastique

1. **Initialisation**
2. Soit $l = 0, X^0 = \hat{X} = X$, et $\bar{z} = +\infty$ la valeur de la fonction objectif.
3. **Répéter**
4. Résoudre
5. $\min C^T x$
6. $x \in X^l$
7. Soit $\hat{x}$ la route optimale obtenue.
8. **Si** $c^T\hat{x} + Q(\hat{x}) < \bar{z}$, **alors**
9. $\bar{z} = c^T\hat{x} + Q(\hat{x})$
10. Fixer $\bar{x} = \hat{x}$.
11. **Fin si**
12. Effectuer la séparation locale à partir de $\hat{x}$ et utiliser $\hat{X}$ comme le premier domaine des
13. solutions réalisables en utilisant les paramètres k, p et d.
14. Fixer $\tilde{x}^l = \arg min_{l=1,\dots,p}\{c^T x^{l^l} + Q(x^{l^l})\}$
15. **Si** $c^T\tilde{x}^l + Q(\tilde{x}^l) < \bar{z}$, **alors**
16. Fixer $\bar{z} = c^T\tilde{x}^l + Q(\tilde{x}^l)$
17. Fixer $\bar{x} = \tilde{x}^l$
18. **Fin Si**
19. Fixer $X^{l+1} = X^l \cap \left\{x \middle| \Delta(x, \hat{x}) \geq k+1, \Delta\left(x, x^{l^l}\right) \geq k+1, i = 1, \dots, p-1\right\}$.
20. Fixer $\hat{X} = \hat{X} \cap \{x | \Delta(x, \tilde{x}^l) \geq 1\}$.
21. Fixer $l = l + 1$
22. **Jusqu'à** $l < d$
23. **Retourner** $\bar{x}$ **et** $\bar{z}$

L'algorithme est formé principalement d'une boucle dont la condition d'arrêt est la fin du nombre de descentes. Dans la première partie (les lignes 4-7), le problème déterministe est résolu pour trouver une solution de départ $\hat{x}$ pour chaque nouvelle descente l. L'ensemble X^l est le domaine des solutions réalisables du problème déterministe. Une fois le problème déterministe résolu, la deuxième partie effectue la séparation locale à partir de $\hat{x}$ en utilisant les paramètres p et κ. Soit $\tilde{x}^l$ la meilleure solution trouvée dans la descente courante, cette solution est trouvée à la ligne 14. La troisième partie (les lignes 19-21) est dédié à mettre à jour les domaines des solutions réalisables du problème déterministe et du problème stochastique. La ligne 19 permet d'ajouter toutes les contraintes de séparation à droite au problème déterministe et la ligne 20 force le programme stochastique à ignorer toutes les solutions stochastiques déjà trouvées. À la ligne 21, il s'agit d'incrémenter le compteur l. L'algorithme retourne une solution $\bar{x}$ et sa valeur $\bar{z}$.

CHAPITRE IV

ANALYSE NUMÉRIQUE DES RÉSULTATS

Le présent chapitre a comme objectif de présenter les résultats de nos tests numériques. Ainsi, les résultats obtenus par l'algorithme SL seront comparés aux résultats obtenus par l'algorithme Integer *L*-Shaped de Jabali *et al.*, (2012) sur un ensemble d'instances du PTVDS jugées difficiles. Ces analyses nous permettrons d'identifier le degré de réussite de l'algorithme proposé en fonction des objectifs de recherche établis. Ce chapitre se divise da la façon suivante. En premier lieu, nous expliquons les détails des implémentations des algorithmes testés, tout en incluant une description des instances utilisées pour l'ensemble des tests numériques. Dans la deuxième partie, nous présentons les résultats et les analyses. Nous analysons finalement l'impact des deux stratégies appliquées dans l'algorithme SL, à savoir les stratégies d'intensification et de diversification.

4.1 Implémentations et instances

L'algorithme SL a été codé en utilisant le langage de programmation C++ et en utilisant le logiciel CPLEX 12.3. Les résultats numériques ont été obtenus sur un ensemble de 15 ordinateurs, chaque ordinateur ayant deux processeurs Opteron 275 AMD Dual-Core 2.2 GHZ 64-bits et un RAM de 7.7 GHZ. Le système d'exploitation utilisé par ces ordinateurs est Linux SuSe11.3. Pour ce qui est des algorithmes de séparation, nous utilisons les coupes **comb** et **cap** de la librairie CPTVSEP développée par Lysgaard *et al.*, (2004) afin d'identifier les contraintes de capacité violées par les solutions fractionnaires. Ces contraintes sont celles permettant

également l'élimination des sous-tours, voir les inéquations (2.4) définies au Chapitre 2.

Afin de comparer adéquatement la performance de notre algorithme et l'algorithme Integer L-Shaped, nous utilisons les mêmes instances générées par Jabali *et al.* (2012) pour effectuer les tests. Les instances sont générées de la façon suivante. D'abord, n nœuds ont été générés dans un carré $[0,100]^2$ selon une distribution uniforme continue. Cinq obstacles rectangulaires dont les dimensions égales à $[20,80]^2$ sont aussi générés, chacun à une base de 4 et une hauteur de 25, couvrant 5% de toute la surface. Les demandes des clients ξ_i sont considérées indépendantes et ayant des distributions normales $\mathcal{N}(\mu_i, \sigma_i)$ tronquées à zéro. Le coefficient de variation utilisé pour les distributions des demandes est égal à 30%. Soit $\bar{f} = \sum_{i=2}^{n} \mu_i / mD$ le coefficient de remplissage moyen de la flotte composée de m véhicules. Plus le coefficient $\bar{f}$ est grand, plus l'instance devient difficile à résoudre. Le nombre de clients est représenté par le paramètre n. Les valeurs de ces paramètres sont combinées pour générer neuf types d'instances. Pour chaque type, 10 instances sont générées. Le Tableau 4.1 montre les combinaisons employées pour générer les instances (90 au total).

Tableau 4.1
Les combinaisons des paramètres pour générer les instances

m=2	$n \in \{60, 70, 80\}$	$\bar{f}$=0.93
m=3	$n \in \{50, 60, 70\}$	$\bar{f}$=0.88
m=4	$n \in \{40, 50, 60\}$	$\bar{f}$=0.82

Les paramètres de l'algorithme SL sont fixés à différentes valeurs. La taille des voisinages pour les sous-problèmes est fixée aux valeurs suivantes : $\kappa = 4, 6$ et 8. La

profondeur des descentes est fixée à $p = 1, 3$ et 6. et le nombre des descentes au niveau de la stratégie de diversification est fixé à $d = 1, 2, 4$ et 6 . Chaque instance est résolue par l'algorithme SL pour chaque combinaison des paramètres κ, p et d. Nous considérons la même limite de temps de calcul que celle utilisée pour tester l'algorithme Integer L-Shaped (Jabali *et al.*, 2012) soit 10 heures (ou 36 000 secondes).

Afin d'identifier le degré de réussite de notre heuristique et de déterminer l'efficacité de la méthode SL sur les instances classées difficiles à résoudre pour l'algorithme Integer L-Shaped, il est primordial de comparer les résultats de nos tests avec celles de l'algorithme Intger L-Shaped. D'abord, deux éléments seront observés pour effectuer les analyses, soit le gap moyen et le temps d'exécution moyen. Ensuite, nous présenterons le nombre de fois où l'heuristique est en mesure d'améliorer les valeurs des solutions obtenues par rapport l'algorithme Integer L-Shaped. Finalement nous allons classer le pourcentage des instances résolues des deux algorithmes pour différentes gammes d'instances.

4.2 Analyse des résultats

Afin de fixer les valeurs de κ et p, nous avons effectué les tests sur deux phases. Dans la section 4.2.1, nous avons fixé le nombre de descentes $d = 1$ et nous avons varié p et κ afin de tester la stratégie d'intensification. Dans la section 4.2.2, nous avons fixé κ et p sur la base des observations de la première phase et nous avons augmenté le nombre d afin de tester la stratégie de diversification.

Nous divisons les instances en deux ensembles E_1 et E_2. Ces derniers dépendent des résultats obtenus par l'algorithme Integer L-Shaped. Le premier ensemble E_1 comporte les instances résolues à l'optimalité par l'algorithme Integer L-Shaped dans le temps qui lui est alloué. Le deuxième ensemble E_2 comporte les instances non résolues à l'optimalité dans le temps alloué par l'algorithme Integer L-Shaped. Selon

cette classification, nous obtenons 21 instances dans le premier ensemble, dont aucune instance résolue pour les deux combinaisons suivantes : 1) n=60, m=3 et $\bar{f}$=0.88 et 2) n=70, m=3 et $\bar{f}$=0.88). Quant au deuxième, celui-ci comporte 69 instances. Nous procédons à cette classification afin d'observer l'impact de la méthode SL sur les instances difficiles et faciles à résoudre par l'algorithme Integer L-Shaped.

4.2.1 L'évaluation de la stratégie d'intensification

Afin d'observer l'impact de la stratégie d'intensification sur le processus général de résolution, nous avons fixé le paramètre $d = 1$, et nous avons varié les valeurs du paramètre p = 1, 3 et 6. Pour ce qui est du paramètre κ, nous le fixons aux valeurs suivantes : κ = 4, 6 et 8. Le choix de ces dernières valeurs est motivé par les observations qui ont été faites par Rei *et al.*, (2010), à savoir que ces valeurs sont adéquates pour la résolution du PTVDS. Il est à noter que dans les tableaux de résultats présentés, les informations suivantes sont indiquées au niveau des colonnes :

- **Instance** : Le nom de l'instance. Le format pour présenter une instance est le suivant : $n\_m\_\bar{f}$.
- **Opt** : Le nombre des instances résolues à l'optimalité par l'algorithme Integer L-Shaped dans le temps maximum alloué.
- **Pas.Opt** : le nombre des instances qui n'étaient pas résolues à l'optimalité par l'algorithme Integer L-Shaped dans le temps maximum alloué.
- ***L*-Shaped** : Les résultats de l'algorithme Integer L-Shaped.

Nous calculons le gap en se basant sur la borne inférieure de l'algorithme Integer L-Shaped.

$$\text{gap} = \frac{\text{Borne supérieur (SL)} - \text{Borne inférieure } (L - \text{Shaped})}{\text{Borne inférieure } (L - \text{Shaped})}$$

Étant donné le grand nombre de résultats pour les instances considérées, nous rapporterons dans les tableaux les gaps moyens (GMs) et les temps d'exécution moyens (TMs) sur le nombre des instances (Opt et Pas.Opt).

Pour calculer les gaps moyens pondérés (MP^gs) et les temps moyens pondérés (MP^ts) pour chaque application de SL en combinant les paramètres d, p et κ, nous utilisons le même principe que la moyenne pondérée.

Dans le cas de l'algorithme SL, le temps de résolution disponible (i.e., 36 000 secondes) est distribué de façon uniforme sur l'ensemble des sous-problèmes à résoudre. Par conséquent, nous définissons par $t' = 36000/(dp)$ le temps maximum initial attribué à chaque profondeur. Par la suite, lorsque l'algorithme SL débute, pour tout $p > 1$, la valeur t' est modifiée dynamiquement en ajoutant de façon automatique le temps non utilisé dans une profondeur au temps total disponible pour la résolution des profondeurs restantes.

Les Tableaux 4.2 et 4.3 présentent les GMs et les TMs pour l'ensemble des instances E_1 lorsque l'algorithme SL est appliqué pour $d = 1$. Nous remarquons, qu'à chaque fois que les valeurs de p et κ augmentent, le MP^g diminue. Par exemple, lorsque $p = 1$ et $\kappa = 4$ le MP^g est de 0,28%, lorsque $p = 6$ et $\kappa = 8$, le MP^g diminue à 0,08%. De plus, pour les instances de 80 clients, l'heuristique obtient les solutions optimales pour toutes les combinaisons de κ et p, à l'exception de $\kappa = 4$.

Tableau 4.2

Les gaps moyens de E_1 pour $d = 1$

Instance	Opt	d=1									L-Shaped
		p=1			p=3			p=6			
		κ=4	κ=6	κ=8	κ=4	κ=6	κ=8	κ=4	κ=6	κ=8	
60_2_0.93	9	0,08%	0,02%	0,08%	0,50%	0,22%	0,22%	0,08%	0,02%	0,07%	0%
70_2_0.93	2	0,00%	0,27%	0,07%	0,80%	0,64%	0,61%	0,00%	0,27%	0,07%	0%
80_2_0.93	3	0,05%	0,00%	0,00%	0,04%	0,00%	0,00%	0,04%	0,00%	0,00%	0%
50_3_0.88	2	0,45%	0,01%	0,01%	0,45%	0,01%	0,01%	0,45%	0,01%	0,01%	0%
40_4_0.82	3	0,20%	0,21%	0,16%	0,15%	0,16%	0,16%	0,15%	0,16%	0,16%	0%
50_4_0.82	1	2,61%	1,06%	1,06%	2,61%	0,75%	0,48%	2,61%	0,48%	0,48%	0%
60_4_0.82	1	0,89%	0,86%	0,00%	0,89%	0,00%	0,00%	0,33%	0,00%	0,00%	0%
MP^g		0,28%	0,16%	0,12%	0,24%	0,09%	0,05%	0,25%	0,08%	0,08%	0%
Totale	21										

Tableau 4.3

Les temps d'exécution moyens de $\boldsymbol{E_1}$ pour $\boldsymbol{d = 1}$

Instance	Opt	d=1									L-Shaped
		p=1			p=3			p=6			
		κ=4	κ=6	κ=8	κ=4	κ=6	κ=8	κ=4	κ=6	κ=8	
60_2_0.93	9	56	109	508	224	613	1907	315	1815	5092	443
70_2_0.93	2	17	65	33	59	107	2888	47	176	10102	159
80_2_0.93	3	46	193	280	404	4809	1021	771	4522	6826	837
50_3_0.88	2	46	149	339	192	711	1768	356	2025	5484	563
40_4_0.82	3	6	24	48	31	185	319	63	446	1481	565
50_4_0.82	1	115	2650	8133	368	3601	7449	717	4674	11452	2135
60_4_0.82	1	285	12024	555	1248	12496	1234	1644	6777	5227	3026
MP^t		57	797	714	259	1821	1866	405	2242	5648	704
Totale	21										

Si nous comparons la valeur moyenne des solutions trouvées par le SL pour $p = 6$ et $\kappa = 4$ et la valeur moyenne des solutions obtenues par l'algorithme Integer L-Shaped, nous remarquons clairement que l'algorithme SL peut produire un MP^g proche de l'optimalité en moins de temps (405 secondes comparativement à 704 secondes).

Nous constatons également qu'à chaque augmentation de la valeur de κ, le temps de calcul augmente. Ceci s'explique par le fait qu'une augmentation de la valeur de ce

paramètre engendre une augmentation de la taille du domaine réalisable associé aux sous-problèmes résolus. Ainsi, le MP^t observé pour l'algorithme SL dépasse le MP^t associé à l'algorithme Integer L-Shaped lorsque $k = 6$ et 8.

Précédemment, nous avons analysé les résultats obtenus pour l'ensemble E_1 en regroupant les instances que l'algorithme Integer L-Shaped résout à l'optimalité. En second lieu, nous analyserons les résultats obtenus pour l'ensemble E_2 correspond aux instances que l'algorithme Integer L-Shaped n'arrive pas à résoudre. Il est à noter que pour cet ensemble, l'algorithme Integer L-Shaped obtient un MP^g de 1,38%.

Les Tableaux 4.4 et 4.5 présentent les GMs et les TMs pour l'ensemble des instances E_2 lorsque l'algorithme SL est appliqué pour $d = 1$. Le MP^g obtenu par l'heuristique lorsque $p = 1$ et $k = 8$ (1,39%) est équivalant au MP^g produit par l'algorithme L-Shaped. En revanche, l'algorithme SL obtient ces résultats en moins de temps, soit 16 367 secondes comparativement à 36 000 secondes nécessaires pour l'algorithme L-Shaped (le temps maximum alloué pour résoudre une instance). Pour toutes les combinaisons de $p = 1$ et 3 et $\kappa = 6$ et 8, à l'exception de $p = 1$ et $\kappa = 6$, l'algorithme SL obtient de meilleurs GMs, et ce, plus rapidement que l'algorithme Integer L-Shaped (ex. pour la combinaison $p = 3$ et $\kappa = 8$, le MP^g est 1,22% pour un MP^t de 20 784 secondes).

Tableau 4.4

Les gaps moyens de E_2 pour $d = 1$

Instance	Pas.Opt	d=1									L-Shaped
		p=1			p=3			p=6			
		κ=4	κ=6	κ=8	κ=4	κ=6	κ=8	κ=4	κ=6	κ=8	
60_2_0.93	1	1,00%	0,44%	0,44%	1,00%	0,44%	0,44%	1,00%	0,44%	0,44%	0,44%
70_2_0.93	8	0,99%	0,80%	0,70%	0,96%	0,71%	0,70%	0,96%	0,67%	0,67%	0,33%
80_2_0.93	7	0,90%	0,71%	0,66%	0,88%	0,70%	0,65%	0,88%	0,66%	0,65%	0,68%
50_3_0.88	8	1,92%	1,41%	1,36%	1,92%	1,06%	0,98%	1,92%	0,99%	0,93%	0,61%
60_3_0.88	10	1,31%	1,19%	1,06%	1,27%	1,15%	0,97%	1,20%	1,07%	0,97%	0,96%
70_3_0.88	10	1,58%	1,32%	1,36%	1,54%	1,21%	1,32%	1,43%	1,17%	1,30%	1,48%
40_4_0.82	7	3,53%	3,12%	2,55%	2,64%	2,40%	1,87%	2,37%	2,17%	1,81%	2,85%
50_4_0.82	9	2,30%	1,72%	1,67%	2,14%	1,60%	1,55%	1,73%	1,60%	1,59%	2,21%
60_4_0.82	9	2,18%	1,94%	1,92%	2,01%	1,60%	1,78%	1,79%	1,49%	1,71%	2,01%
MP^g		1,80%	1,49%	1,39%	1,65%	1,29%	1,22%	1,52%	1,21%	1,20%	1,38%
Totale	69										

Tableau 4.5

Les temps d'exécution moyens de E_2 pour $d = 1$

Instance	Pas.Opt	d=1									L-Shaped
		p=1			p=3			p=6			
		κ=4	κ=6	κ=8	κ=4	κ=6	κ=8	κ=4	κ=6	κ=8	
60_2_0.93	1	113	997	3037	486	1259	4007	963	3764	9733	36000
70_2_0.93	8	212	6867	15268	734	12491	16644	1359	17632	25324	36000
80_2_0.93	7	737	4400	8178	1669	12436	18698	2612	19327	28029	36000
50_3_0.88	8	132	4241	13741	598	10188	16909	1179	17699	29876	36000
60_3_0.88	10	371	9077	13684	1196	15889	20634	1992	20885	29474	36000
70_3_0.88	10	632	21787	22837	2657	22146	23558	4487	25156	27616	36000
40_4_0.82	7	65	6259	9687	141	8698	15555	255	17598	18661	36000
50_4_0.82	9	133	8402	19869	488	19736	26134	957	25939	31218	36000
60_4_0.82	9	531	16670	25016	2114	22614	27195	3214	25684	33143	36000
MP^t		355	10127	16367	1243	15828	20784	2082	21303	27946	36000
Totale	69										

4.2.2 L'évaluation de la stratégie de diversification

Pour la deuxième phase des tests numériques, nous analyserons l'impact qu'une augmentation du nombre de descentes effectuées (le paramètre d) peut avoir sur les résultats. Pour ce faire, nous fixons p à 1 et 3 étant donné la qualité des GMs par l'heuristique. Il est à noter qu'en fixant p à 6, l'algorithme SL obtient la même qualité de solution, mais nécessite en moyenne plus de temps.

Les Tableaux 4.6 et 4.7 présentent les GMs et les TMs pour l'ensemble des instances E_1 lorsque l'algorithme SL est appliqué pour $d = 1$. Nous observons une amélioration au niveau de la qualité de solution avec une augmentation au niveau du temps d'exécution. En effet, ceci est dû au fait que l'effort ait été doublé.

Tableau 4.6
Les gaps moyens de E_1 pour $d = 2$

Instance	d=2						L-Shaped
	p=1			p=3			
	κ=4	κ=6	κ=8	κ=4	κ=6	κ=8	
60_2_0.93	0,07%	0,02%	0,08%	0,07%	0,02%	0,07%	0%
70_2_0.93	0,00%	0,00%	0,07%	0,00%	0,00%	0,07%	0%
80_2_0.93	0,01%	0,00%	0,00%	0,00%	0,00%	0,00%	0%
50_3_0.88	0,30%	0,01%	0,01%	0,30%	0,01%	0,01%	0%
40_4_0.82	0,15%	0,16%	0,16%	0,15%	0,16%	0,16%	0%
50_4_0.82	1,81%	1,06%	1,06%	1,81%	0,75%	0,48%	0%
60_4_0.82	0,13%	0,13%	0,00%	0,13%	0,00%	0,00%	0%
MP^g	0,18%	0,09%	0,12%	0,17%	0,07%	0,08%	0%

Tableau 4.7
Les temps d'exécution moyens de E_1 pour $d = 2$

Instance	d=2						L-Shaped
	p=1			p=3			
	κ=4	κ=6	κ=8	κ=4	κ=6	κ=8	
60_2_0.93	118	239	1027	315	1194	4370	443
70_2_0.93	36	88	83	64	149	6552	159
80_2_0.93	103	454	565	514	8304	9719	837
50_3_0.88	92	278	865	352	1682	4581	563
40_4_0.82	14	54	126	57	317	677	565
50_4_0.82	202	14763	13898	763	14822	15166	2135
60_4_0.82	442	12154	613	1320	6481	2115	3026
MP^t	110	1492	1320	355	2932	5241	704

Les Tableaux 4.8 et 4.9 présentent les GMs et les TMs pour l'ensemble des instances E_2 lorsque l'algorithme SL est appliqué pour $d = 2$. La combinaison p= 3 et $\kappa = 6$ produit un MP^g de 1,36% versus 1,38% pour l'algorithme Integer L-Shaped. Par contre, les résultats de l'algorithme SL sont obtenus en moyenne 12 fois plus rapidement que l'algorithme Integer L-Shaped. Ainsi, lorsque $\kappa = 4$, 6 et 8 en fixant p à 3, l'heuristique obtient des GMs meilleurs que ceux obtenus par l'algorithme Integer L-Shaped.

Tableau 4.8
Les gaps moyens de E_2 pour $d = 2$

Instance	d=2						L-Shaped
	p=1			p=3			
	κ=4	κ=6	κ=8	κ=4	κ=6	κ=8	
60_2_0.93	0,44%	0,44%	0,85%	1,00%	0,44%	0,44%	0,44%
70_2_0.93	0,79%	0,71%	0,98%	0,71%	0,68%	0,67%	0,33%
80_2_0.93	0,67%	0,65%	0,85%	0,72%	0,67%	0,65%	0,68%
50_3_0.88	1,06%	1,01%	1,16%	1,06%	0,71%	0,63%	0,61%
60_3_0.88	1,19%	1,06%	1,31%	1,09%	0,97%	0,97%	0,96%
70_3_0.88	1,33%	1,36%	1,58%	1,51%	1,19%	1,29%	1,48%
40_4_0.82	2,82%	2,26%	2,91%	2,46%	2,22%	1,80%	2,85%
50_4_0.82	1,59%	1,58%	1,98%	1,49%	1,54%	1,57%	2,21%
60_4_0.82	1,96%	1,83%	2,15%	1,87%	1,60%	1,55%	2,01%
MP^g	1,40%	1,30%	1,60%	1,36%	1,18%	1,14%	1,38%

Tableau 4.9
Les temps d'exécution moyens de E_2 pour $d = 2$

Instance	d=2						L-Shaped
	p=1			p=3			
	κ=4	κ=6	κ=8	κ=4	κ=6	κ=8	
60_2_0.93	249	2223	5840	776	2407	8494	36000
70_2_0.93	498	12503	18298	1310	17195	23790	36000
80_2_0.93	1697	9246	15143	2888	16098	27065	36000
50_3_0.88	248	8828	14994	915	12867	22894	36000
60_3_0.88	831	13235	19047	2009	20772	27462	36000
70_3_0.88	1255	22948	24467	4109	23929	26724	36000
40_4_0.82	123	8238	15036	269	13779	20467	36000
50_4_0.82	297	13688	24972	850	23948	31767	36000
60_4_0.82	1096	23399	25900	3360	27426	31230	36000
MP^t	759	14360	19948	2025	19731	26428	36000

Les Tableaux 4.10-4.17 présentent les GMs et les TMs pour l'ensemble des instances E_1 et E_2 lorsque l'algorithme SL est appliqué pour $d = 4$ et $d = 6$. Nous constatons qu'en augmentant le nombre des descentes, l'algorithme SL obtient des solutions pour lesquelles le MP^g se rapproche de l'optimalité (Pour $d = 4$, $p = 1$ et $\kappa = 4$ le

MP^g est à 1,49%, comparativement à 1,08% lorsque $d = 6$, $p = 3$ et $\kappa = 8$. Par contre, ces améliorations s'accompagnent de MP^t plus élevés. Pour $d = 4$ $p = 1$ et $\kappa = 4$ le MP^t est à 202 secondes, comparativement à 32 514 secondes lorsque $d = 6, p = 3$ et $\kappa = 8$.

Tableau 4.10

Les gaps moyens de E_1 pour $d = 4$

Instance	d=4						*L*-Shaped
	p=1			p=3			
	κ=4	κ=6	κ=8	κ=4	κ=6	κ=8	
60_2_0.93	0,07%	0,02%	0,08%	0,07%	0,02%	0,07%	0%
70_2_0.93	0,00%	0,00%	0,07%	0,00%	0,00%	0,07%	0%
80_2_0.93	0,01%	0,00%	0,00%	0,00%	0,00%	0,00%	0%
50_3_0.88	0,30%	0,01%	0,01%	0,30%	0,01%	0,01%	0%
40_4_0.82	0,15%	0,16%	0,16%	0,15%	0,16%	0,16%	0%
50_4_0.82	1,55%	1,06%	1,06%	1,81%	0,75%	0,48%	0%
60_4_0.82	0,13%	0,13%	0,00%	0,13%	0,00%	0,00%	0%
MP^g	0,16%	0,09%	0,12%	0,17%	0,07%	0,08%	0%

Tableau 4.11

Les temps d'exécution moyens de E_1 pour $d = 4$

Instance	d=4						*L*-Shaped
	p=1			p=3			
	κ=4	κ=6	κ=8	κ=4	κ=6	κ=8	
60_2_0.93	199	454	2165	547	3958	8421	443
70_2_0.93	69	137	158	142	303	14136	159
80_2_0.93	234	806	1208	1093	3736	10357	837
50_3_0.88	190	589	1211	665	3560	9858	563
40_4_0.82	32	116	266	119	653	1264	565
50_4_0.82	353	35766	27568	1176	21119	20973	2135
60_4_0.82	777	9554	1203	1567	4104	5718	3026
MP^t	202	2554	2639	615	3892	8825	704

Tableau 4.12

Les gaps moyens de E_2 pour $d = 4$

Instance	d=4						L-Shaped
	p=1			p=3			
	κ=4	κ=6	κ=8	κ=4	κ=6	κ=8	
60_2_0.93	0,85%	0,44%	0,44%	0,85%	0,44%	0,44%	0,44%
70_2_0.93	0,73%	0,79%	0,65%	0,70%	0,72%	0,57%	0,33%
80_2_0.93	0,84%	0,67%	0,65%	0,66%	0,66%	0,67%	0,68%
50_3_0.88	1,16%	0,99%	0,99%	1,05%	0,71%	0,63%	0,61%
60_3_0.88	1,13%	1,02%	1,01%	1,09%	0,96%	0,88%	0,96%
70_3_0.88	1,58%	1,33%	1,37%	1,48%	1,25%	1,36%	1,48%
40_4_0.82	2,91%	2,82%	2,28%	2,46%	2,54%	1,72%	2,85%
50_4_0.82	1,65%	1,58%	1,42%	1,46%	1,54%	1,54%	2,21%
60_4_0.82	2,04%	1,96%	1,82%	1,87%	1,51%	1,34%	2,01%
MP^g	1,49%	1,37%	1,26%	1,34%	1,22%	1,09%	1,38%

Tableau 4.13

Les temps d'exécution moyens de E_2 pour $d = 4$

Instance	d=4						L-Shaped
	p=1			p=3			
	κ=4	κ=6	κ=8	κ=4	κ=6	κ=8	
60_2_0.93	529	4642	11894	1611	4754	17497	36000
70_2_0.93	904	21068	20916	2363	21610	27285	36000
80_2_0.93	3745	15390	23724	6269	20872	30141	36000
50_3_0.88	460	12554	19011	1630	18440	26813	36000
60_3_0.88	1991	19420	25337	4113	27886	33794	36000
70_3_0.88	2636	24520	27479	8478	28663	31190	36000
40_4_0.82	229	10206	18227	527	20258	21551	36000
50_4_0.82	576	20631	27000	1575	29751	32709	36000
60_4_0.82	2036	29107	26791	6873	32045	33145	36000
MP^t	1580	19418	23728	4102	25141	29778	36000

Tableau 4.14

Les gaps moyens de E_1 pour $d = 6$

Instance	d=6						*L*-Shaped
	p=1			p=3			
	κ=4	κ=6	κ=8	κ=4	κ=6	κ=8	
60_2_0.93	0,07%	0,02%	0,08%	0,07%	0,02%	0,07%	0%
70_2_0.93	0,00%	0,00%	0,07%	0,00%	0,00%	0,07%	0%
80_2_0.93	0,00%	0,00%	0,00%	0,00%	0,00%	0,00%	0%
50_3_0.88	0,30%	0,01%	0,01%	0,30%	0,01%	0,01%	0%
40_4_0.82	0,15%	0,16%	0,16%	0,15%	0,16%	0,16%	0%
50_4_0.82	1,55%	1,06%	1,06%	1,81%	0,75%	0,62%	0%
60_4_0.82	0,13%	0,13%	0,00%	0,13%	0,00%	0,00%	0%
MP^g	0,16%	0,09%	0,12%	0,11%	0,06%	0,05%	0%

Tableau 4.15

Les temps d'exécution moyens de E_1 pour $d = 6$

Instance	d=6						*L*-Shaped
	p=1			p=3			
	κ=4	κ=6	κ=8	κ=4	κ=6	κ=8	
60_2_0.93	290	709	3795	782	5619	11947	443
70_2_0.93	97	192	284	212	355	15591	159
80_2_0.93	376	1126	1741	1707	5099	15221	837
50_3_0.88	326	942	1794	1133	5413	15219	563
40_4_0.82	53	181	408	191	997	1999	565
50_4_0.82	524	36004	36004	1663	23565	24092	2135
60_4_0.82	1005	6968	1552	1899	3885	8739	3026
MP^t	299	2645	3920	904	5135	12078	704

Tableau 4.16
Les gaps moyens de E_2 pour $d = 6$

Instance	d=6						L-Shaped
	p=1			p=3			
	κ=4	κ=6	κ=8	κ=4	κ=6	κ=8	
60_2_0.93	0,85%	0,44%	0,44%	0,85%	0,44%	0,44%	0,44%
70_2_0.93	0,73%	0,79%	0,65%	0,70%	0,57%	0,67%	0,33%
80_2_0.93	0,84%	0,67%	0,66%	0,66%	0,67%	0,67%	0,68%
50_3_0.88	1,16%	0,95%	1,02%	1,05%	0,71%	0,57%	0,61%
60_3_0.88	1,11%	1,02%	1,01%	1,07%	0,86%	0,90%	0,96%
70_3_0.88	1,58%	1,31%	1,35%	1,48%	1,23%	1,28%	1,48%
40_4_0.82	2,91%	2,82%	2,37%	2,46%	2,54%	1,74%	2,85%
50_4_0.82	1,46%	1,57%	1,42%	1,46%	1,54%	1,51%	2,21%
60_4_0.82	2,04%	1,98%	1,86%	1,87%	1,51%	1,34%	2,01%
MP^g	1,46%	1,36%	1,28%	1,33%	1,18%	1,08%	1,38%

Tableau 4.17
Les temps d'exécution moyens de E_2 pour $d = 6$

Instance	d=6						L-Shaped
	p=1			p=3			
	κ=4	κ=6	κ=8	κ=4	κ=6	κ=8	
60_2_0.93	760	5599	17910	2653	6703	23526	36000
70_2_0.93	1316	24364	22669	3440	22774	32363	36000
80_2_0.93	6055	20757	28301	9228	25011	35899	36000
50_3_0.88	677	13944	20677	2321	22354	27914	36000
60_3_0.88	3132	23238	28738	6148	31838	35533	36000
70_3_0.88	3917	25663	28843	11994	31745	35243	36000
40_4_0.82	340	11703	19206	845	21157	23357	36000
50_4_0.82	845	23183	28400	2557	31974	33627	36000
60_4_0.82	2878	29781	28480	8852	33170	34723	36000
MP^t	2398	21811	25869	5846	27725	32514	36000

Lorsque $d > 6$, la qualité moyenne des solutions obtenues stagnent. En effet, ceci s'explique par le fait que le temps maximum alloué à la résolution des sous-

problèmes ne permet pas de résoudre ceux-ci efficacement. Étant donné le temps maximum alloué à la résolution des problèmes, il est préférable de maintenir le nombre de descentes à un maximum de 6.

Le Tableau 4.18 représente le nombre de fois où l'heuristique est en mesure d'améliorer la valeur des solutions obtenues par rapport à l'algorithme Integer L-Shaped. Dans ce tableau nous considérons les résultats obtenus pour $d = 6$. La bonne performance de SL est observée dans les résultats de la combinaison $p = 3$ et $\kappa = 8$, où il y a une amélioration dans 50% des instances.

Tableau 4.18
Amélioration de valeur de solution par rapport L-Shaped

Instance	d=6					
	p=1			p=3		
	κ=4	κ=6	κ=8	κ=4	κ=6	κ=8
60_2_0.93	0	2	2	0	2	2
70_2_0.93	1	1	1	1	2	2
80_2_0.93	2	4	3	3	3	3
50_3_0.88	4	4	4	5	5	6
60_3_0.88	4	5	5	4	5	6
70_3_0.88	7	8	8	7	8	8
40_4_0.82	3	4	4	3	4	5
50_4_0.82	7	7	7	7	7	7
60_4_0.82	6	5	6	6	5	5
Totale	34	40	40	36	41	44

Finalement, le Tableau 4.19 regroupe les pourcentages des instances où des solutions dont le gap d'optimalité est d'au plus 0%, 1%, 2%, 3% et 4% ont été obtenues par les algorithmes testés. Nous observons que 65% des instances sont résolue en bas de 1% de l'optimalité par l'heuristique comparativement à 40% par l'algorithme Integer L-Shaped. Il est également intéressant d'observer que pour 2% des instances,

l'heuristique obtient une solution à plus de 4% de l'optimum comparativement à 7% des instances dans le cas de la méthode exacte.

Tableau 4.19

Classifications des pourcentages des instances résolues par différentes gammes

Gamme	SL	L-Shaped
= 0%	10%	23%
≤ 1%	65%	40%
≤ 2%	92%	82%
≤ 3%	95%	90%
≤ 4%	98%	93%

CHAPITRE V

CONCLUSION

Dans de mémoire, nous avons généralisé l'heuristique proposée à l'origine par Rei et al. (2010) au cas du PTVDS à plusieurs véhicules. L'heuristique se base sur la méthode SL et exploite la performance de l'algorithme Integer *L*-Shaped pour résoudre les sous-problèmes générés par le processus de résolution. L'heuristique met en œuvre une stratégie de recherche multi-descentes dans laquelle la solution de départ de chaque descente est obtenue en solutionnant la version déterministe du problème. La méthode SL est par la suite appliquée pour réaliser chaque descente afin d'améliorer la solution de départ en considérant explicitement la fonction de recours dans les sous-problèmes résolus.

Les résultats de l'algorithme montrent qu'il est possible d'obtenir des solutions de qualité équivalente à celles obtenues par l'algorithme Integer *L*-Shaped, tout en réduisant les temps de calcul. Ainsi, notre algorithme peut résoudre de façon efficace des instances considérées comme étant difficiles lorsque résolues par l'algorithme Integer *L*-Shaped.

En effet, bien que son efficacité sur le problème à un seul véhicule ait été prouvée par (Rei et *al.*, 2010), l'heuristique proposée n'avait pas, à ce jour, été testé sur le cas plus générale où une flotte de m véhicules est utilisée pour résoudre le PTVDS. De plus, considérant que l'algorithme Integer *L*-Shaped ne peut résoudre que des instances de taille plus limitée, le besoin d'une heuristique efficace pour résoudre le problème était amplement justifié. L'heuristique proposée, en appliquant de façon complémentaire la

méthode Integer *L*-Shaped et la méthode SL, a donné de très bons résultats sur le problème considéré.

Suite à la réussite de la méthode et aux résultats obtenus, l'heuristique pourrait être appliquée sur d'autres problèmes d'intérêt tel que les PTV avec clients stochastique, c'est-à-dire lorsqu'il y a de l'incertitude au niveau de la présence des clients. Enfin l'heuristique pourrait aussi être appliquée à des cas réels. Les entreprises pourraient bénéficier de cette recherche en appliquant la méthode de façon à obtenir de meilleures solutions pour leurs activités de distribution.

RÉFÉRENCES

Barker, E. K. (2002). Evolution of Microcomputer-Based Vehicle Routing Software: Case Studies in United States . *In The Vehicle Routing Problem*, P. Toth and D. Vigo (eds), SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 353-362.

Chepuri, K. et Hommem-De-Mello, T. (2005). Solving the vehicle routing problem with stochastic demands using the cross-entropy method. *Annals of Operations Research, 134*, 153-181.

Christiansen, C. H. et Lysgaard, J. (2007). A branch-and-price algorithm for the capacitated vehicle routing problem with stochastic demands. *Operations Research Letters, 35*, 773-781.

Dantzig, G. B. et Ramser, J. H.. (1959). The truck dispatching problem. *Management Science, 6*, 80-91.

Dantzig, G. B. et Wolfe, P. (1960). Decomposition principle for linear programs. *Operations Research, 8*, 101-111.

Dror, M., Laporte, G. et Trudeau, P. (1989). Vehicle routing with stochastic demands : properties and solution frameworks. *Transportation Science, 23*(3), 166-176.

Fischetti, M. et Lodi, A. (2003). Local branching. *Mathematical Programming, 98*, 23-47.

Gendreau, M., Laporte, G. et Séguin, R. (1996). A Tabu Search Heuristic for the vehicle routing problem with stochastic demands and customers. *Operations Research, 44*(3), 469-477.

Hjorring, C. et Holt, J. (1999). New optimality cuts for a single-vehicle stochastic routing problem. *Annals of Operations Research, 86*, 569-584.

Ismail, Z. et Irhmah. (2008). Solving the Vehicle Routing Problem with Stochastic Demands via Hybrid Genetic Algorithm-Tabu Search. *Journal of Mathematics and Stochastic, 4*(3), 161-167.

Ismail, Z. et Irhmah. (2008). Adaptive Permutation-Based Genetic Algorithm for Solving PTV with Stochastic Demands. *Journal of Applied Scientific Information*, *8*(18), 3228-3234.

Jabali, O., Rei, W., Gendreau, M. et Laporte, G.(2012). New Valid Inequalities for the Multi-Vehicle Routing Problem with Stochastic Demands. Publication CIRRELT-2011-58, Centre interuniversitaire de recherché sur les réseaux d'entreprise, la logistique et le transport, Université de Montréal, Montréal, QC. Canada.

Laporte, G. et Louveaux, F. V. (1993). The integer L-Shaped method for stochastic integer programs with complete recourse. Operations Resarch Letters, *13*(3), 133-142.

Laporte, G., Louveaux, F. V. et Van Hamme, L. (2002). The integer *L*-Shaped algorithm for the capacitated routing problem with stochastic demands ». *Operations Research*, *50*(3), 415-423.

Lysgaard, J., Letchford, A. N. et Eglese, R. W. (2004). A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming*, *100*, 423-445.

Rei, W., Gendreau, M. et Soriano, P. (2010). A hybrid Monte Carlo local branching algorithm for the single vehicle routing problem with stochastic demands. *Transportation Science*, *44*(1), 136-146.

Secomandi, N. (2001). A rollout policy for the vehicle routing problem with stochastic demands. *Operations Research*, *49*(5), 796-802.

Secomandi, N. et Margot, F. (2009). Reoptimization approaches for the vehicle-routing problem with stochastic demands. *Operations Research*, *57*(1), 214-230.

Tillman, F. A. (1969). The multiple terminal delivery problem with probabilistic demands. *Transportation Science*, *3*(3), 192-204.

Toth, P. et Vigo, D. (2002). The Vehicle Routing Problem. Philadelphia: Society for Industrial and Applied Mathematics, 367 .

Transport Canada. 2012 (8 juillet). « Les transports et l'économie ». En ligne. <http://www.tc.gc.ca/fra/politique/anre-menu-3016.htm#a4>. Consulté le 13 février 2013.

Vidal, T., Crainic, T. G., Gendreau, M. et Prins, C. (2011). Heuristique pour les problèmes de tournées de véhicules multi-attribus. Publication CIRRELT-2011-12, Centre interuniversitaire de recherché sur les réseaux d'entreprise, la logistique et le transport, Université de Montréal, Montréal, Canada.

Yang, W., Mathur, K. et Ballou, R. H. (2000). Stochastic vehicle routing problem with restocking. *Transportation Science*, *34*(1), 99-112.