UNIVERSITÉ DE TECHNOLOGIE DE TROYES

École Doctorale

SCIENCES DES SYSTÈMES TECHNOLOGIQUES ET ORGANISATIONNELS

Année : 2006

Thèse de doctorat de l'Université de Technologie de Troyes

Spécialité : Optimisation et sûreté des systèmes

Présentée et soutenue publiquement par

Caroline PRODHON

LE PROBLÈME DE LOCALISATION-ROUTAGE

Jury

Rapporteurs :	
Michel GENDREAU	*Professeur, Centre de recherche sur les transports, Université de Montréal, Canada*
Frédéric SEMET	*Professeur, LAMIH, Université de Valenciennes et du Hainaut-Cambrésis*
Examinateurs :	
Alexandre DOLGUI	*Professeur, G2I, École des Mines de Saint Étienne*
Aristide MINGOZZI	*Professeur, PSDC, Université de Bologne, Italie*
Christian PRINS	*Professeur, ICD, Université de Technologie de Troyes*
Marc SEVAUX	*Professeur, LESTER, Université de Bretagne-Sud*
Roberto WOLFLER-CALVO	*Maître de conférence, ICD, Université de Technologie de Troyes*

Directeur de thèse : Christian PRINS
Co-directeur : Roberto WOLFLER-CALVO
Laboratoire : Institut Charles Daulaunay - Équipe Optimisation des Systèmes Industriels

Résumé

Cette thèse porte sur le problème de localisation-routage (*Location-Routing Problem* - *LRP*). Il s'agit d'un problème de logistique du transport qui implique deux niveaux de décision : la localisation de dépôts (niveau stratégique) et l'élaboration de tournées de véhicules (niveau tactique ou opérationnel). Ces deux niveaux sont souvent inter-dépendants, mais pour des raisons de simplification, ils sont généralement traités séparément. Or, de récentes recherches ont montré que la prise en compte des futures tournées dans la résolution d'un problème de localisation permettait des gains significatifs sur les coûts totaux. Il existe de nombreuses applications du *LRP* comme la distribution du courrier, la livraison de colis ou la collecte de déchets. La motivation des travaux développés ici est d'aborder un problème difficile et dans une version encore peu étudiée dans la littérature, comportant des capacités limitées à la fois pour les dépôts à ouvrir et pour les véhicules réalisant les tournées. De plus, des problèmes de taille réaliste sont visés, c'est-à-dire avec un nombre de clients à servir allant jusqu'à 200. Nous proposons des techniques de résolutions prenant en considération l'intégralité du problème, sans décomposition hiérarchique en deux phases, par le biais de méthodes de type heuristique, mais également avec une approche exacte basée sur de nouveaux modèles mathématiques. Tous les algorithmes développés ont été testés et validés sur des jeux d'essais nouveaux ou provenant de la littérature.

Mots clés : Optimisation combinatoire, Recherche opérationnelle, Logistique (organisation), Problèmes de transport (programmation), Problèmes de localisation (programmation), Heuristique.

Remerciements

Remerciements ... partie très importante puisqu'elle est dédiée à toutes les personnes qui m'ont permis d'en arriver où je suis et parce qu'elle symbolise également la fin du travail de thèse.

Je ne vais quand même pas repartir de ma petite enfance, même si j'ai une pensée à tous ceux qui m'ont donné le goût d'apprendre. Quoi? j'ai toujours aimé ça? Bon, peut-être ... mais il n'empêche que j'ai de très bons souvenirs de l'école, alors je pense que cela aide.

Pour ne remonter qu'à des temps raisonnables, un très grand merci à mes deux directeurs de thèse, Christian Prins et Roberto Wolfler-Calvo, pour m'avoir fortement encouragée à poursuivre dans cette voie et choisie pour ce sujet de thèse. Petite pensée également à Anne Barros qui m'a confortée dans cette voie.

Comme mes directeurs de thèse ont une grande place dans mon cœur, mais je tiens également à leur exprimer toute ma gratitude pour leur aide, conseils, soutien... et ceci aussi bien sur le plan professionnel que personnel. Grâce à eux, j'ai aussi pu rencontrer et travailler avec des chercheurs de différents pays, ce qui est un atout et une chance exceptionnelle, et je leur en suis particulièrement reconnaissante.

Merci à toutes les personnes avec qui j'ai travaillé avec un immense plaisir, Patrick Soriano, Angel Ruiz, Enrique Benavent, Jose-Manuel Belenguer, et tous ceux avec qui j'ai pu être en contact dans les laboratoires.

Je tiens bien évidemment à exprimer ma reconnaissance aux membres du jury : Michel Gendreau et Frédéric Semet, mes rapporteurs, pour avoir pris le temps de lire ce mémoire dans les moindres détails et juger ce travail, Marc Sevaux, président du jury, pour sa confiance et son estime, et enfin Alexandre Dolgui, Aristide Mingozzi et le jury dans son ensemble pour leur présence, leurs encouragements et conseils qui m'ont permis de tout finaliser, et leur sympathie.

Sans argent, pas de thèse, alors je donne ma complaisance à la région Champagne-Ardenne pour ses financements.

C'est d'une manière indubitable et profondément sincère que je dis un grand merci à mes parents et ma famille pour leur soutien, la fierté et la confiance qu'ils me portent, même si la recherche n'est pas leur milieu et qu'ils ne comprennent pas toujours tout de ce que je fais de mes journées.

Petite dédicace aux thésards qui ont partagé mon quotidien ou de simples moments : mes amis Manu, Julien, Mathieu et Yann, l'équipe Ellidoc, bien évidemment Mourad avec qui j'ai partagé mon bureau et les différentes phases de la thèse, et tous les autres. Grâce à eux, des fous-rires et des soirées sympathiques ont égayé mes trois années, même dans les moments les moins roses.

Enfin, merci à Gaëlle pour son écoute et son soutien, à mes amis hors ou ex-UTT pour les moments de détentes partagés et à tous ceux que j'aurai pu oublié.

Caroline PRODHON.

Table des matières

Notations et Abréviations

Notations

V : ensemble des nœuds d'un graphe

I : sous-ensemble des nœuds d'un graphe représentant m sites possible des dépôts.

$J = V \backslash I$: sous-ensemble de V représentant n nœuds-clients.

c_{ij} : coût de transport entre deux nœuds i et j.

W_i : capacité associée au dépôt $i \in I$.

O_i : coût fixe d'ouverture associé au dépôt $i \in I$.

d_j : demande du client $j \in J$.

K : ensemble de véhicules identiques.

Q : capacité d'un véhicule.

F : coût fixe d'utilisation d'un véhicule.

$\forall H \subseteq E$, $x(H)$: nombre d'arêtes utilisées dans un ensemble H.

$\forall S \subseteq J$, $D(S) = \sum_{j \in S} d_j$: demande totale des clients de l'ensemble S.

$\forall S \subseteq V$, $\delta(S)$: ensemble des arêtes ayant une extrémité dans S et l'autre dans $V \backslash S$.

$\forall S \subseteq V$, $\gamma(S)$: ensemble des arêtes ayant leurs deux extrémités dans S.

$\forall S \subseteq V$ et $\forall S' \subseteq V \backslash S$, $E(S : S')$: ensemble des arêtes avec une extrémité dans S et l'autre dans S'

Abréviations

AC (Ant Colony) : algorithme à colonie de fourmis.

ECWA (Extended Clarke and Wright Algorithm) : algorithme généralisé de Clarke et Wright.

GA (Genetic Algorithm) : algorithme génétique.

GRASP (Greedy Randomized Adaptive Search Procedure) : procédure de recherche adaptative gloutonne et randomisée.

GTS (Granular Tabu Search) : algorithme de recherche taboue granulaire.

LRP (Location-Routing Problem) : problème de localisation-routage.

LS (Local Search) : recherche locale.

MA|PM (Memetic Algorithm with Population Management) : algorithme mémétique avec gestion de la population.

MDVRP (Multi-Depot Vehicle Routing Problem) : problème de tournées de véhicules multi-dépôt.

PPV : heuristique du Plus Proche Voisin.

PR (Path Relinking) : algorithme de Path Relinking.

SA (Simulated Annealing) : algorithme de type recuit simulé.

SS (Scatter Search) : algorithme de type recherche dispersée.

TS (Tabu Search) : algorithme de recherche taboue.

TSP (Traveling Salesman Problem) : problème du voyageur de commerce.

VRP (Vehicle Routing Problem) : problème de tournées de véhicules.

Chapitre 1

Introduction générale

Le problème étudié dans cette thèse se nomme le problème de localisation-routage (*Location-Routing Problem - LRP*). Avant de l'aborder plus spécifiquement, ce chapitre d'introduction rappelle dans la section 1.1 l'intérêt de l'activité économique auquel il appartient, la logistique du transport. Les sections suivantes présentent des classifications possibles pour un problème de logistique, à savoir selon les niveaux de décision (section 1.2) et selon diverses caractéristiques pouvant être considérées (section 1.3). Pour finir, nous situons la problématique étudiée dans cette thèse par rapport aux classifications proposées et énonçons le plan de ce mémoire en section 1.4.

1.1 Logistique du transport

Avec notamment la mondialisation, les délocalisations des centres de production et l'expansion des marchés, le transport revêt une importance capitale en assurant les liaisons entre ces derniers. La logistique qui y est liée soulève un grand nombre de problèmes, très souvent difficiles à résoudre de manière optimale. Le *LRP* est un de ces problèmes qui consiste à localiser un ensemble de dépôts à ouvrir et à élaborer des tournées de véhicules associées. D'une manière plus générale, en logistique du transport, le but est en principe de visiter un ensemble de lieux appelés *clients* (pouvant être considérés ponctuels comme un site bien précis, ou longiligne comme une rue) à un coût minimum.

L'intérêt pour ce genre d'activité est grandissant dans la vie actuelle, puisqu'elle peut s'appliquer à de nombreux domaines, aussi bien le transfert de produits physiques avec la distribution du courrier, la livraison de colis, la collecte de lait, que l'agencement de lignes de transports en commun, le salage du réseau routier, le ramassage des ordures, etc.

De plus, la mondialisation implique des coûts logistiques de plus en plus importants pour les entreprises. C'est pourquoi ces dernières ont tout intérêt à s'intégrer dans des réseaux d'approvisionnement et de distribution afin de minimiser les coûts engendrés et assurer leur compétitivité et leur pérennité. Il ne faut pas non plus négliger un autre aspect prenant une dimension d'un intérêt grandissant : la prise de conscience environnementale qui incite au développement durable en visant par exemple une réduction des émissions des gaz à effet de

serre, dont le transport est en partie responsable.

C'est pour toutes ces raisons que des efforts sont nécessaires afin de développer des systèmes de transport flexibles et efficaces répondant aux préoccupations actuelles relatives à l'économie et à notre qualité de vie. La Recherche Opérationnelle sur ce type de problèmes s'avère donc essentielle.

Mais pour les aborder, il est important de savoir quel type de décisions doivent être considérées. La section suivante propose une classification des niveaux de décisions.

1.2 Niveaux de décision

Les problèmes de logistique du transport soulèvent souvent différentes questions auxquelles il faut répondre, l'idée générale étant de satisfaire les clients, sous diverses contraintes, au moindre coût. Les grandes catégories de décisions à prendre peuvent se répartir selon une classification popularisée par la gestion de production :

- *Stratégique* : ce sont des décisions à long terme (un an ou plus). A ce niveau, c'est plutôt l'aspect localisation des sites de production et de transit qui est considéré (usines, dépôts, plate-formes...)
- *Tactique* : ce sont des décisions à moyen terme (quelques mois à un an). Il s'agit par exemple de décider de l'achat de nouveaux véhicules ou de la fréquence de visite des clients dans le cas de tournées périodiques.
- *Opérationnel* : ce sont des décisions à court terme (de quelques jours à un mois). Le but est d'élaborer les tournées qui seront effectivement réalisées par exemple.
- *Temps Réel* : ce sont les décisions à très court terme (de l'ordre de quelques heures à quelques minutes). Ces décisions interviennent lors d'aléas de dernière minute qui perturbent les décisions planifiées.

L'approche de résolution fréquemment appliquée est la décomposition en deux ou trois phases selon ces niveaux de décision, en commençant par les aspects stratégiques. L'avantage est la simplification apportée.

En effet, pour des raisons temporelles logiques, il est souvent nécessaire de répondre aux questions d'ordre supérieur avant d'aborder les autres niveaux. Par exemple, il est impossible d'élaborer des plans de distribution (niveau tactique ou opérationnel) pour un ensemble d'usines sans savoir où ces dernières seront construites (niveau stratégique).

Un autre motif touche la fiabilité des données concernant les questions d'ordre inférieur au niveau considéré. Un tel cas est manifeste pour la constitution d'un ordonnancement d'atelier (niveau opérationnel) lors du calcul d'un plan de production (niveau tactique). Ce dernier se fait généralement sur des estimations de futurs besoins qui ont de grandes chances d'évoluer d'ici la période de fabrication à proprement parler. Les données au moment de l'élaboration de ce plan ne sont donc pas suffisamment fidèles pour construire un ordonnancement adéquat.

Une considération également prise en compte concerne les montants financiers mis en jeu dans les décisions de niveau supérieur, qui sont en principe les plus importants. La respon-

sabilité et la qualification des personnels gérant de telles phases sont donc plus élevées. Par exemple, les pertes occasionnées par des mauvais choix lors d'une tournée de distribution sont a priori moins pénalisantes que celles découlant de décisions de construction d'usines à des emplacements peu judicieux.

Enfin, une explication se rapporte à la simplification découlant de la décomposition. Les problèmes obtenus sont plus petits et nécessitent ainsi des méthodes de résolution moins complexes.

Cependant, malgré les avantages et les justifications d'une décomposition en phases selon les niveaux, cette manière de procéder ne prend pas en compte l'intégralité du problème considéré. En effet, chaque phase ignore ou gère grossièrement les données disponibles concernant les décisions de niveaux inférieurs, et les choix sur des phases déjà résolues sont souvent irrévocables ou difficilement modifiables. Les solutions globalement optimales risquent d'être écartées par cette démarche. Il est donc intéressant de s'attarder sur une vision plus globale du problème, d'autant plus que les progrès de la recherche opérationnelle et de l'informatique ouvrent la porte à des résolutions de problèmes de plus en plus complexes.

Le problème de localisation-routage étudié dans cette thèse combine des décisions classiquement considérées de niveaux différents : stratégique (localisation) et tactique ou opérationnel (élaboration de tournées de véhicules).

1.3 Différents types de problèmes de distribution

Outre la détermination des niveaux de décisions visés, il est important d'analyser les caractéristiques du problème logistique traité avant de chercher à le résoudre. Pour cela, différents aspects sont à considérer. Sans chercher à être exhaustifs, les paragraphes suivants donnent des idées sur les aspects à prendre en compte en phase préliminaire de modélisation d'un problème.

1.3.1 Les contraintes

Différents types de contraintes peuvent entrer en jeu dans un problème de logistique du transport. Par exemple, il peut y avoir des contraintes de capacités sur les véhicules et/ou sur les dépôts. Pour illustrer simplement, si le but est de servir un ensemble de clients au moindre coût, sans considérer l'aspect localisation et sans aucune capacité le problème de base obtenu est appelé problème du voyageur de commerce ou *TSP* (*Traveling Salesman Problem*). Ce dernier est déjà *NP*-difficile au sens fort. Mais si tous les clients ne peuvent être livrés par un même véhicule pour cause de contraintes de capacité, on obtient le problème de tournées de véhicules, *VRP* (*Vehicle Routing Problem*). Lorsque de plus, l'ensemble des clients ne peut être servi par un unique dépôt ou que plusieurs sites peuvent les approvisionner par exemple, le problème de tournées de véhicules devient multi-dépôt, *MDVRP* (*Multi-Depot Vehicle Routing Problem*). En complexifiant encore, si l'aspect localisation fait partie des décisions à prendre, alors le problème à résoudre est celui de localisation-routage, *LRP* (*Location Routing Problem*).

D'autres contraintes peuvent également s'ajouter. On peut citer celles visant à éviter le fait d'avoir une tournée géante et une multitude de petites, si ce type de résultat n'est pas en accord avec la politique de travail de l'entreprise. Un autre exemple est l'intégration de fenêtres de temps. Il s'agit alors de visiter les clients pendant un intervalle de temps prédéfini. Ce dernier type de contraintes est souvent utilisé pour des retraits ou livraisons chez des clients non disponibles continuellement par exemple.

Enfin, on peut citer l'extension consistant à prendre en compte des contraintes de stocks, d'où résulte l'*inventory routing problem* ou *IRP*. L'*IRP* est un problème multi-période où en plus de livrer les clients, l'entreprise aura en charge la gestion de leurs stocks, ce qui lui permet d'élaborer des tournées optimisées en combinant les deux dimensions (stocks et routage).

1.3.2 La nature des données

Une autre considération à prendre en compte concerne la nature des données. En général, celles-ci comportent une part d'incertitude. Cela est dû au fait qu'il est quasiment impossible de prévoir de manière sure les demandes à venir et qu'il est souvent indispensable de prendre des décisions stratégiques, tactiques ou même opérationnelles avant même d'avoir reçu des commandes fermes. C'est pourquoi, la plupart du temps, les études sont faites avec des données moyennes ou des estimations. Ces estimations pourront alors être traitées comme des commandes fermes. Cependant, parfois, quand les aléas sont importants, une représentation déterministe est trop faible et il peut être nécessaire d'avoir recours à des modèles stochastiques afin de mieux représenter le problème réel.

1.3.3 La nature du problème

Il s'agit ici du caractère statique, dynamique ou périodique du problème. Selon l'horizon considéré et indépendamment de l'incertitude sur les données, ces dernières peuvent être constantes (cas statique) ou varier dans le temps. Il peut s'agir de perturbations (comme des pannes) ou d'apparitions/disparitions de demandes (cas dynamique), ou alors de fluctuations saisonnières ou d'évolutions de tendance de la demande (cas périodique) par exemple. L'approche la plus répandue est l'approche statique, considérant des informations constantes sur la période de l'étude. Il est cependant parfois nécessaire d'étudier les cas dynamiques et périodiques.

1.3.4 Les objectifs

Le dernier aspect abordé dans cette section est relatif aux objectifs visés. Si ceux-ci correspondent à un ou plusieurs termes non contradictoires à minimiser par exemple, alors il est possible de les combiner afin de formuler une unique fonction-objectif.

Par contre, comme c'est parfois le cas dans des problèmes réels, différents objectifs antagonistes peuvent être à optimiser. Par exemple, pour un cas de ramassage scolaire, le but peut être la minimisation du nombre de tournées et le temps d'attente des enfants. Deux grandes approches sont alors envisageables. La première consiste à pondérer chaque objectif

et à les sommer. Cette approche nous ramène au cas précédent. La seconde est de séparer les objectifs. Ce dernier cas est en général mieux adapté et le problème est dit multi-objectif. Il faut alors effectuer un choix parmi une liste de solutions efficaces. En effet, il est impossible de définir ce qu'est *la* valeur optimale d'un problème d'optimisation multi-objectif en toute généralité puisque les fonctions à optimiser sont contradictoires. D'une manière générale, il existe un ensemble de solutions, formant une frontière de *Pareto*.

Dans un autre style, la fonction-objectif peut être linéaire ou non. Des modèles complètement linéaires de grande taille peuvent être actuellement résolus par des solveurs du commerce. Il est par contre difficile de déterminer un optimum global pour un modèle non-linéaire.

1.4 But de la thèse

La thèse étudiée ici s'inscrit dans le domaine du transport. Elle traite du problème de localisation-routage ou *LRP* (*Location-Routing Problem*). Ce dernier combine deux niveaux de décision. Un choix doit se faire sur un sous-ensemble de dépôts à ouvrir, niveau de décision stratégique, et des tournées de véhicules doivent être élaborées, niveau tactique ou opérationnel, dans le but de minimiser les coûts totaux, considérés linéaires. Il appartient au champ de l'Optimisation Combinatoire.

L'association des deux niveaux de décision rend la résolution particulièrement complexe, même en considérant un modèle de type statique et déterministe, comme c'est le cas dans cette thèse. De plus, des contraintes de capacité sur chacun des niveaux sont prises en compte et l'ambition est de traiter des instances de grande taille, allant jusqu'à 200 clients à servir et 20 sites possibles pour les dépôts.

Afin de parvenir au but visé, il est important de connaître les méthodes déjà testées sur différentes versions du *LRP* et sur les sous-problèmes classiques de logistique du transport qui le composent. C'est pourquoi un état de l'art sur la localisation, les tournées de véhicules et le *LRP* est présenté au chapitre 2. Une définition plus précise du problème de localisation-routage étudié durant cette thèse est exposé au chapitre 3 avec différentes propositions de modélisations possibles.

L'objectif étant essentiellement de résoudre des problèmes de taille réaliste, le chapitre 4 est consacré à des heuristiques relativement rapides par rapport aux métaheuristiques : des méthodes constructives et de fusions, et des recherches locales.

Dans le chapitre 5, une première métaheuristique est exposée. Il s'agit d'une procédure de recherche adaptative gloutonne et randomisée (appelée *GRASP*). Afin de la rendre plus pertinente, la méthode de base a été ici renforcée par une technique d'apprentissage et par une post-optimisation par Path Relinking (*PR*). Une seconde métaheuristique est présentée dans le chapitre 6. L'approche est de type évolutif. Plus précisément, il s'agit d'un algorithme mémétique avec gestion de la population.

Ensuite, une nouvelle approche dite *coopérative* est présentée au chapitre 7. Cette méthode est basée sur un rapport non hiérarchique entre des phases de localisation et de constitution de tournées. Ainsi, le choix des dépôts est basé sur une formulation simplifiée du problème de départ, permettant d'obtenir un problème de localisation résolu à l'aide d'une relaxation

lagrangienne avec une optimisation par sous-gradients et une heuristique lagrangienne. La partie sur les tournées est améliorée par une heuristique de type tabou granulaire.

Une méthode exacte et des propositions de bornes inférieures sont développées dans le chapitre 8. Il s'agit de techniques basées sur un algorithme de coupes. Dans chaque chapitre, la nouvelle méthode présentée étant comparée avec l'approche précédente ayant fournit les meilleurs résultats, une synthèse sur les méthodes développées en comparaison avec des résultats de la littérature est également réalisée dans ce chapitre 8. Pour terminer, une conclusion sur les travaux de cette thèse ainsi que sur les perspectives de recherche dans le domaine sont données au chapitre 9.

Chapitre 2

État de l'art

2.1 Introduction

Avant de débuter le développement de modèles et de méthodes de résolution, il est nécessaire de chercher dans la littérature ce qui existe comme travaux en rapport avec le problème traité dans cette thèse. Le *LRP* étant une combinaison des aspects localisation et routage, il semble judicieux de s'attarder également sur ces deux sous-problèmes.

Dans un premier temps, des concepts relatifs aux problèmes de logistique du transport sont introduits dans la section 2.2. Ensuite, les sous-problèmes du *LRP* sont développés, avec quelques références, non-exhaustives, concernant les diverses méthodologies de résolution qui leur sont consacrées. Plus précisément, les sections 2.3 et 2.4 donnent un aperçu des travaux concernant respectivement des problèmes de localisation et de tournées de véhicules en relation avec le problème considéré dans cette thèse. La section 2.5 est consacrée aux travaux existants sur le *LRP*. La section 2.6 clôt le chapitre.

2.2 Concepts de base

2.2.1 Représentation des problèmes de logistique du transport

De nombreux problèmes relatifs au transport sont modélisés par un graphe. Cet représentation sert, de manière générale, à décrire la structure d'un ensemble complexe dont les éléments sont en relation. Il est donc parfaitement adapté pour définir un réseau quelconque, comme un réseau routier ou électrique par exemple.

Souvent, les relations entre les éléments sont orientées : une relation d'un élément à un autre n'est pas forcément réciproque. On parle alors de *graphe orienté*, défini par un couple $G = (V,A)$ avec :

- V un ensemble de sommets ou *nœuds*.
- A une famille de couples ordonnés de sommets appelés *arcs*.

En pratique un nœud i peut correspondre à un dépôt, un client, mais aussi un carrefour et un arc (i,j) symbolise un tronçon de route ou une rue entre deux carrefours, avec un sens de circulation. Si toutes les rues sont à double sens, l'existence d'un arc (i,j) de i vers j implique celle de l'arc opposé (j,i), et le graphe est dit symétrique. Dans ce cas, on peut utiliser un modèle de graphe non-orienté $G = (V,E)$, dans lequel l'ensemble A est remplacé par un ensemble E d'arêtes (liaisons non-orientées). Un graphe urbain est par contre rarement symétrique, à cause des sens uniques.

Les relations entre les éléments peuvent être pondérées, ou valuées, c'est-à-dire que chaque arc du graphe est muni d'un poids, ou *coût*, défini par une application C de A dans $\mathbb{R}$. Le graphe se note alors $G = (V,A,C)$. Concrètement, dans un réseau routier par exemple, les valuations peuvent correspondre à des temps de parcours, des distances, des capacités en nombre de véhicules/heure, une limitation de vitesse, etc.

S'il existe un chemin menant d'un nœud i à un nœud j dans un graphe valué, il est possible de définir son coût, égal en général à la somme des valuations des arcs ou arêtes traversées pour joindre les deux nœuds. En reprenant l'exemple précédent, les éléments d'un réseau routier peuvent être disséminés dans un graphe, avec une majorité de nœuds-carrefours. Dans ce cas, il est possible de former un nouveau graphe *complet* $G' = (S,U,D)$ dans lequel l'ensemble de sommets S représente les nœuds auxquels on s'intéresse particulièrement (par exemple les clients et les dépôts, mais pas les carrefours). Chaque arc $(i,j) \in S \times S$ de G' modélise un plus court chemin de i à j, valué par le coût c_{ij} de ce chemin.

Pour calculer les chemins de coût minimal ou *plus courts chemins* d'un nœud de départ s vers tous les autres nœuds, il est possible d'utiliser des algorithmes dits à *étiquettes*. A chaque sommet i est attribuée une étiquette, valeur des plus courts chemins de s à i. Deux grandes familles de tels algorithmes existent : ceux dits à *fixation d'étiquettes* comme l'algorithme de Dijkstra (1959), et ceux dits à *correction d'étiquettes* comme l'algorithme de Bellman (Gondran et Minoux, 1995).

Le *LRP* peut être défini par rapport à un graphe $G = (V,E,C)$, souvent complet et symétrique. V est alors un ensemble de nœuds, composé de deux sous-ensembles : un sous-ensemble I de m dépôts et un sous-ensemble $J = V \backslash I$ de n clients. E représente les arêtes mettant en relation les nœuds du graphe. C est l'application qui définit le coût d'utilisation des arêtes.

Les ouvrages traitant des graphes sont nombreux. Concernant la théorie qui leur est liée, on peut citer par exemple Yellen et Gross (1998). D'un point de vue plus algorithmique, on peut nommer le livre de Gibbons (1985), ou l'ouvrage plus récent en français de Lacomme *et al.* (2003).

2.2.2 Optimisation combinatoire

Les décisions à prendre dans le *LRP* concernent l'ouverture de dépôts et la constitution de tournées, ainsi que le choix des arcs pour construire les tournées. Les variables de décision nécessaires sont de type entier et le but est de minimiser les coûts en respectant un certain nombre de contraintes. Le problème appartient au champ de l'Optimisation Combinatoire (*Combinatorial Optimization - CO*).

De manière générale, un problème d'optimisation combinatoire consiste à rechercher un extremum (minimum ou maximum) d'une fonction, souvent appelée fonction-coût ou *fonction-objectif*, sur un ensemble discret. Cette solution est alors dite *optimale*. De tels problèmes ne sont pas abordables par les outils de l'analyse mathématique, à moins de les approximer en utilisant des variables à valeurs réelles. Malheureusement, la solution optimale entière peut être très éloignée de la relaxation continue. Théoriquement, il faudrait donc énumérer toutes les solutions possibles et ne garder que la meilleure. Ce procédé est une trivialité d'un point de vue mathématique mais d'un point de vue informatique et pratique, le temps de recherche d'une solution optimale d'un problème de *CO*

ces contraintes forment un polyèdre (ensemble de solutions d'un système fini d'inégalités linéaires).

La relaxation continue d'un *PLNE* peut se résoudre par l'algorithme du simplexe par exemple. Mais en nombres entiers, il faut avoir recours à des méthodes telles que les algorithmes de *séparation et évaluation* ou les *méthodes de coupes*. Ces deux approches sont succinctement abordées ici.

Algorithmes de séparation et évaluation

Un algorithme par séparation et évaluation, également appelé selon le terme anglo-saxon *branch and bound*, est une méthode générique de résolution de problèmes d'optimisation combinatoire, apparue au milieu du XX^{e} siècle. Elle énumère de manière intelligente l'ensemble des solutions. Pour cela, elle décompose l'espace des solutions en sous-ensembles de plus en plus petits, dont une bonne partie est éliminée à l'aide de bornes. Ce type d'énumération peut donc fournir une solution optimale en un temps réduit par rapport à une énumération complète. Cependant, pour les instances de grande taille des problèmes *NP*-difficiles, leur durée d'exécution est encore trop importante pour pouvoir être utilisable dans des applications réelles et il faut alors se tourner vers les approches heuristiques.

Les composantes principales des méthodes arborescentes sont :

- une règle de séparation des solutions (*branching rule*);
- une fonction d'évaluation des solutions (*evaluation function*);
- une stratégie d'exploration (*search strategy*).

La méthode part d'un nœud-racine N_0 qui représente implicitement l'ensemble des solutions. La séparation consiste à partitionner cet ensemble en applicant une décision. Par exemple, dans un programme linéaire en 0-1. on peut choisir une variable x_j et séparer N_0 en deux nœuds N_1 et N_2 : les solutions avec $x_j = 0$ et celles avec $x_j = 1$. En répétant ce processus sur les nœuds-fils, on construit de proche en proche un arbre dont les feuilles correspondent aux solutions.

Le but de la méthode étant de réduire l'arbre de recherche, chaque nœud N subit une évaluation. Pour cela, une estimation du coût de la meilleure solution en ce nœud est calculée à l'aide d'une borne inférieure pour un problème de minimisation (ou supérieure en maximisation). Si cette estimation est supérieure (inférieure dans le cas de maximisation) à la meilleure solution trouvée jusqu'à maintenant, il ne sert à rien de séparer le nœud. Ce dernier est dit *tué* ou *élagué* (*pruned*). On échappe ainsi au développement de tout sous-arbre de racine N.

La stratégie d'exploration quant à elle désigne la façon d'avancer dans l'arbre : soit en profondeur, soit en progressif. L'exploration en profondeur (*depth-first search*) est la plus économique en mémoire. On sépare d'abord le dernier nœud construit et l'exploration se fait en profondeur jusqu'à une feuille ou jusqu'à ce qu'un nœud soit tué. Arrivé à ce stade, il faut remonter au dernier nœud non exploré et reprendre en profondeur, et ainsi de suite. La stratégie n'explore qu'une branche à la fois, qu'on stocke en pratique dans une pile. En progressif (*frontier search*), le nœud séparé est celui de plus faible évaluation. L'avantage est de souvent améliorer plus rapidement la meilleure solution provisoire et donc d'accélérer la recherche puisqu'il sera plus facilement possible de tuer des nœuds. En revanche, la méthode consomme plus de mémoire. Dans tous les cas, afin d'accélérer au maximum la méthode, il

est primordial d'avoir les meilleures bornes possibles.

Des détails sur cette approche peuvent se trouver dans la plupart des livres traitant de problèmes d'optimisation combinatoire, comme Balas et Toth (1985) ou Lacomme *et al.* (2003), et elle se retrouve dans de nombreux articles comme ceux de Carpaneto et Toth (1980); Fischetti *et al.* (1994) ou Toth et Vigo (2002).

Algorithme de coupes

Les méthodes de coupes sont basées sur la relaxation de certaines contraintes du problème initial. Le but est de se ramener à une formulation simplifiée. Une solution optimale du problème relaxé peut être obtenue par un solveur commercial. Souvent, une telle résolution ne conduit pas à une borne très serrée du problème d'origine. Il faut alors trouver un ensemble de contraintes du problème initial qui est violé dans la solution optimale du problème relaxé, et l'introduire dans le programme linéaire. L'algorithme répète ce processus, alternant résolution d'un programme linéaire, identification de contraintes violées et introduction de ces contraintes dans la formulation. Il s'arrête soit quand il obtient une solution réalisable pour le problème d'origine ou une solution dont la fonction-objectif est égale à celle d'une solution heuristique (puisque dans ces deux cas, aucune meilleure solution ne peut être trouvée), soit quand plus aucune contrainte violée n'est identifiée par les procédures dites de *séparation* utilisées. Dans ce dernier cas, la méthode fournit quand même une borne au problème (inférieure pour un problème de minimisation, supérieure sinon).

L'ajout d'une contrainte violée s'appelle une *coupe* car cela supprime une partie de l'espace des solutions afin de se rapprocher du polyèdre de solutions réalisables pour le problème d'origine. Cette approche de résolution est intéressante car elle permet de partir d'une formulation de problème simplifiée et est bien plus rapide à résoudre qu'une procédure de séparation et évaluation appliquée directement sur le modèle du problème initial. Ensuite, des contraintes sont introduites progressivement au fil des itérations. Ces contraintes sont si possible celles induisant des facettes du polyèdre de solutions réalisables car elles permettent alors d'éliminer un plus grand ensemble de solutions non-réalisables du problème d'origine.

Un algorithme de branchement et coupes (*branch and cut*) peut être également réalisé. Le principe est en fait le suivant : à chaque nœud de l'arbre de branchement, une borne inférieure est calculée par un algorithme de coupes. Dans cette version, les solutions obtenues seront alors souvent de meilleure qualité, mais une telle approche requiert des temps de calcul plus importants.

Des détails sur cette approche de type *polyédrale* peuvent se trouver dans la plupart des livres d'optimisation combinatoire Wolsey (1998), et des articles en relation avec les méthodes de coupes pour les problèmes de tournées sont, par exemple, Padberg et Rinaldi (1991); Naddef et Thienel (2002a,b) ou Lysgaard *et al.* (2004).

2.2.3.2 Heuristiques simples et recherches locales

Heuristiques simples

Les heuristiques sont des méthodes approchées, souvent basées sur le bon sens ou sur des observations empiriques, qui permettent d'obtenir des solutions réalisables.

La construction d'une telle solution résulte en général de décisions élémentaires consécutives, chaque élément ajouté étant sélectionné de manière gloutonne. Cela signifie que l'attribut choisi parmi les différentes possibilités pour étendre la solution partielle est celui qui optimise un certain critère. Par exemple, l'heuristique du Plus Proche Voisin (PPV) pour le TSP construit une chaîne hamiltonienne en ajoutant un par un les nœuds en bout de séquence. Une décision élémentaire consiste alors à choisir le nœud selon un critère glouton. Ce dernier est la distance entre le dernier nœud de la chaîne courante et les nœuds encore libres.

Recherches locales

Les solutions obtenues peuvent être améliorées par des recherches locales. Il s'agit alors de regarder dans un voisinage de la solution courante s'il n'est pas possible d'améliorer cette dernière. Pour ne pas tomber dans l'énumération complète des solutions possibles, le voisinage doit être de taille relativement restreinte. Le voisinage d'une solution courante S est le plus souvent défini implicitement comme l'ensemble des solutions qu'on peut obtenir en appliquant à S une transformation simple appelée mouvement. La solution obtenue après de telles améliorations successives sera alors un minimum local.

Une recherche locale classique en problème de tournées est le k-opt (Lin et Kernighan, 1973). Un mouvement dans cette recherche consiste à enlever k arêtes d'un cycle et à reconnecter les chaînes obtenues avec k autres arêtes. A chaque itération, le procédé commence par tester tous les échanges possibles et en estimant leur coût, puis le meilleur est effectué (celui permettant de réduire le plus possible le coût). La recherche stoppe quand plus aucun échange améliorant n'est trouvé. Généralement, on prend $k = 2$ ou 3, car l'exploration est en $O(n^k)$. Avec $k = 2$ par exemple, cela revient à croiser 2 arêtes, et donc à inverser le sens de traversée sur certains arcs. Dans le même genre, Or (1976) a proposé le Or-opt qui consiste à déplacer 1, 2 ou 3 nœuds consécutifs, tout en conservant le sens de circulation. D'autres améliorations ont ensuite été développées dans le même esprit comme par exemple les échanges de chaînes (extension du Or-opt) (Fahrion et Wrede, 1990), les échanges 2-opt* (Potvin et Rousseau, 1995), et 4-opt* (Renaud *et al.*, 1996a). On peut citer également des recherches locales plus sophistiquées : la méthode $GENIUS$ (Gendreau *et al.*, 1992), les λ-interchanges (Osman, 1993), les transferts cycliques (Thompson et Psaraftis, 1993), les chaînes d'éjections (Rego et Roucairol, 1996; Rego, 1998), et la recherche sur très grands voisinages (*very large scale neighborhoods search*), basée sur l'exploration implicite d'un graphe auxiliaire (Ergun *et al.*, 2002).

2.2.3.3 Métaheuristiques

Souvent plus performantes que des heuristiques couplées avec recherche locale classique, les métaheuristiques emploient des méthodes visant à éviter les minimums locaux. Il en existe deux grandes familles.

Les métaheuristiques les plus classiques sont celles fondées sur l'exploration d'un voisinage, et forment la première famille. L'algorithme part d'une solution et la fait évoluer à chaque itération sur l'espace de recherche. Les méthodes les plus connues dans cette catégorie sont la méthode $GRASP$ (*Greedy Randomized Adaptive Search Procedure*), le recuit simulé (*Simulated Annealing, SA*) ou encore la recherche tabou (*Tabu Search, TS*).

L'autre approche possible est basée sur l'utilisation d'une population de solutions. Ce type d'approche constitue la seconde famille. A chaque itération, la métaheuristique fait évoluer un ensemble de solutions en parallèle. Les algorithmes génétiques (*Genetic Algorithms, GA*), de recherche dispersée (*Scatter Search, SS*) ou les algorithmes de colonies de fourmis (*Ant Colony, AC*) sont des exemples courants de ce type de méthode.

Le principe de base de ces six grands types de métaheuristiques est rappelé dans la suivante.

GRASP

Le *GRASP* (*Greedy Randomized Adaptive Procedure*) a été introduit pour la première fois par Feo et Resende (1989). Le principe est de créer une nouvelle solution à chaque itération, indépendante des précédentes. Pour cela, deux phases sont nécessaires : une phase de construction utilisant un algorithme glouton randomisé et une phase de recherche locale. La meilleure solution trouvée lors des différentes itérations est restituée comme résultat.

Des techniques plus élaborées peuvent être introduites dans le but d'améliorer les performances de la version de base (Feo et Resende, 1995; Laguna et Marti, 1995; Fleurent et Glover, 1999; Prais et Ribeiro, 2000).

Parmi les succès du *GRASP* sur les problèmes combinatoires, on peut citer les problèmes de tournées (Kontoravdis et Bard, 1995; Carreto et Baker, 2002), de localisation (Delmaire *et al.*, 1999; Klincewicz, 1992), de transport (Feo et González-Velarde, 1995).

Recuit simulé

Le recuit simulé (*Simulated Annealing, SA*) a été introduit par Kirkpatrick *et al.* (1983) et est inspirée d'une formule de physique concernant le comportement des atomes (énergie E) selon des variations de température (T) : $e^{-E/kT}$. Ainsi, un métal refroidi trop vite présente des imperfections que les auteurs comparent à un minimum local. Par contre, en refroidissant plus lentement, les atomes ont le temps de s'arranger dans une forme cristalline parfaite équivalente au minimum global. Dans un problème combinatoire, la valeur E/k (énergie sur la constante de Boltzmann) est représentée par la variation de la fonction-objectif après l'exécution d'un mouvement de recherche locale à une certaine température T, cette dernière étant un réel. Plus concrètement, il faut tirer au sort une transformation à effectuer sur la solution courante. Si celle-ci engendre une variation de coût Δf négative (amélioration), elle est alors acceptée et réalisée. Sinon, la transformation est acceptée avec une probabilité $e^{-\Delta f/T}$. La température T est diminuée au fur et à mesure des itérations (par exemple $T = 0.999 \times T$). La méthode s'arrête quand T a atteint une certaine valeur proche de 0, ou après un certain nombre d'itérations. La meilleure solution trouvée est alors renvoyée. Le réglage des paramètres de cette méthode est assez délicat.

Des explications et applications de la méthode pour des problèmes de tournées peuvent être trouvées dans les articles de Kirkpatrick *et al.* (1983); Osman (1993) ou Van Breedam (1996).

Recherche de type tabou

Les recherches taboues (*Tabu Search, TS*) nous viennent de Glover (1986, 1989, 1990). Contrairement au *GRASP* et au recuit simulé, il s'agit d'une méthode déterministe, connue pour donner de meilleurs résultats en général. Le principe consiste à balayer entièrement un

voisinage de la solution courante et à effectuer la meilleure transformation possible, même si cette dernière détériore la fonction-objectif. Le retour en arrière sur une solution venant d'être visitée est interdit en gardant dans une liste taboue, de longueur limitée, un historique de la recherche. La meilleure solution trouvée est mémorisée et restituée en fin d'algorithme. La méthode se termine généralement quand un certain nombre d'itérations est atteint.

Le stockage de solutions complètes dans la liste taboue (*tabu list*) consomme souvent beaucoup de mémoire. Une idée est donc de ne conserver que quelques attributs, comme les mouvements ayant permis de passer d'une solution à l'autre ou, plus simplement, la valeur de la fonction-objectif. La liste taboue implémente une mémoire à court terme. Une seconde mémoire, à long terme cette fois-ci, peut être ajoutée pour diversifier la recherche.

Il est possible d'accélérer la méthode en retirant du voisinage visité les solutions faisant intervenir des attributs ayant très peu de chance d'appartenir à la solution optimale. Cette technique de réduction de l'espace proposée par Toth et Vigo (2003) donne le *Granular Tabu Search* (*GTS*).

Des détails sur la méthode peuvent être trouvés dans le livre de Glover et Laguna (1997) et des applications aux problèmes de tournées peuvent être trouvés dans les articles de Gendreau *et al.* (1994); Renaud *et al.* (1996b); Cordeau *et al.* (2001); Cordeau et Laporte (2002) par exemple.

Algorithmes génétiques

Les algorithmes génétiques (*Genetic Algorithms, GA*) sont des méthodes à population proposées par Holland (1975). Ce sont des approches de recherche globale qui imitent les processus naturels observés en génétique. Elles consistent à faire évoluer une *population* d'individus (solutions aléatoires) par des phénomènes de reproduction et de mutation. Mais l'analogie avec la génétique s'arrête là. Un algorithme génétique pur n'utilise que très peu d'informations spécifiques au domaine du problème traité.

Le principe de base est de partir d'une population de solutions, représentées sous forme de *chromosomes*. Ensuite, à chaque itération, des *croisements* de deux *individus-parents* issus de la population sont effectués, en favorisant la *sélection* des plus prometteurs (possédant ce qu'on appelle un bon *fitness*). Le croisement (*crossover*) de deux individus combine des caractéristiques de ces derniers pour générer un nouvel individu appelé *enfant*. Des *mutations* peuvent intervenir lors de la création d'une progéniture, ceci permet une diversification évitant une convergence prématurée. Ensuite, deux modes de gestion de la population peuvent être utilisés. Soit chaque itération crée un nombre d'enfants égal à la taille de la population initiale, et cette dernière est remplacée par la population-enfant lors de l'itération suivante (gestion *générationnelle*). Soit chaque itération ne combine que deux parents, leurs enfants étant alors directement intégrés en remplaçant d'autres individus dans la population courante (gestion *incrémentale*).

Le livre de Reeves (2003) contient des chapitres intéressants sur les algorithmes génétiques. Des versions plus récentes des algorithmes génétiques introduisent des compléments dans la méthode visant à la rendre plus performante. C'est le cas de la version hybride introduisant une recherche locale (algorithme mémétique) (Moscato, 1989; Moscato et Cotta, 2003), ou de la forme plus évoluée utilisant une mesure de distance afin d'apporter une certaine diversité dans les chromosomes-parents avec l'algorithme génétique avec gestion de la population (*Memetic*

Algorithm with Management of the Population, MA|PM) (Sörensen et Sevaux, 2003). Ces versions hybrides permettent de combler l'écart entre les performances des *GA* et des *TS*.

Les algorithmes de type génétique ont été utilisés pour des problèmes de d'affectation (Drezner, 2003) et des problèmes de tournées. Par exemple, pour le *TSP*, Potvin (1996) propose un *GA* de base. Pour *VRP*, plusieurs *MA* ont été développés (Berger et Barkaoui (2003); Lima *et al.* (2004) ou Prins (2004)). Enfin, Prins *et al.* (2004b) nous donne une application d'un *MA|PM* pour le problème de tournées sur arcs (*Capacitated Arc Routing Problem - CARP*).

Recherche dispersée

La recherche dispersée (*Scatter Search, SS*) est également une méthode à population. Le but est d'exploiter l'information contenue dans un ensemble de solutions, et d'en tirer parti pour en construire de nouvelles (Glover *et al.*, 2003). Ses solutions de départ doivent répondre à un critère de diversification basée sur une mesure de distance. Elles peuvent être construites aléatoirement ou par une technique de combinaison par exemple, puis améliorées par recherche locale. En pratique, un grand nombre de solutions sont générées et seul un petit sous-ensemble de référence (souvent appelé *RefSet*) est gardé. Il est composé des solutions répondant à la fois à des critères de performance (coût de la solution) et de diversification. Tous les regroupements possibles de k solutions contenues dans *RefSet* sont réalisés (souvent $k = 2$). Une génération de nouvelles solutions est alors effectuée dans chaque regroupement en réalisant des combinaisons des éléments le constituant. Les meilleures solutions obtenues (au sens performance et diversification) remplacent les moins bonnes dans *RefSet*. Si aucune nouvelle solution n'entre dans *RefSet*, il est possible d'y introduire de nouvelles, selon les mêmes critères que lors de l'initialisation. La procédure s'arrête après un certain nombre d'itérations.

Des détails sur cette méthodes peuvent être trouvées dans les articles de Glover *et al.* (2000, 2003). Une application pour le *VRP* se trouve dans l'article de Russell et Chiang (2006).

Colonies de fourmis

Les méthodes à colonies de fourmis (*Ant Colony, AC*) sont inspirées du comportement des fourmis recherchant de la nourriture. Quand elles en trouvent, elles marquent le chemin y conduisant en laissant des quantités de phéromones relatives la qualité de la source trouvée (éloignement et quantité). Ainsi, les autres fourmis sont averties et attirées dans les directions les plus prometteuses. Avec le temps, les parcours conduisant aux meilleures sources de nourritures sont de plus en plus fréquentés, les taux d'hormones augmentant par le passage de fourmis. Par contre, les chemins peu visités voient leur taux de phéromone s'amoindrir par évaporation. A partir de ces observations, Colorni *et al.* (1991) ont proposé une nouvelle classe de métaheuristiques pour les problèmes combinatoires. Les problèmes de tournées ou de chemins optimaux se prêtent bien à ce type de méthode. En effet, les trajets réalisés par les fourmis correspondent alors à des parcours dans le graphe et la qualité de la source est représentée par la fonction-objectif. L'initialisation du graphe se fait en attribuant des taux de phéronomes nuls sur les arcs. Les fourmis sont représentées par les agents construisant la solution. Ils avancent dans le graphe en effectuant des choix soumis à des probabilités sur les arcs à traverser. En effet, la construction de leur chemin est biaisé en favorisant les arcs fortement marqué de phéromone. Ensuite, selon la valeur de la fonction-objectif obtenue, les

taux de phéromones sont mis à jour. Des méthodes hybrides sont possibles, avec ajout d'une recherche locale par exemple.

Un article détaillant cette méthode est celui de Dorigo *et al.* (1996). Des exemples d'application pour le *VRP* peuvent être trouvés dans l'article de Reimann *et al.* (2004) et Kawamura *et al.* (1998) pour une méthode hybride.

Maintenant que les grands principes de résolution des problèmes d'optimisation combinatoire sont expliqués, dans les sections suivantes, les problèmes liés au *LRP* sont présentés.

2.3 Problèmes de localisation et d'affectation

Il est intéressant de s'attarder sur les sous-problèmes composant le *LRP*, à commencer par celui concernant la localisation des dépôts à ouvrir. En effet, pour concevoir un réseau logistique, la disposition des sites de stockage est une décision très importante. Ces derniers doivent être à la fois bien situés par rapport aux clients, et offrir une capacité suffisante. Les principales questions à se poser en vue de minimiser les coûts résultants sont généralement :

- combien d'installations faut-il ouvrir?
- où doit-on les placer?
- comment y affecter les clients?

Selon les cas, certaines réponses sont déjà fournies et le problème de localisation à résoudre prend alors diverses formes.

Les possibilités de localisation peuvent être considérées comme continues (partout dans le plan) ou discrètes (nombre fini de sites possibles). Le cas discret sera approfondi car le *LRP* relève de cette catégorie.

2.3.1 Problèmes de recouvrement

Le premier problème abordé est celui contenant le moins de type de contraintes. La seule obligation est de couvrir l'ensemble des clients avec les dépôts, chacun ne servant qu'un sous-ensemble pré-défini de clients (souvent les plus proches). Les affectations des clients aux dépôts ne font pas partie des décisions à prendre. Il s'agit d'un problème de recouvrement (*Set Covering Problem - SCP*). L'objectif est de trouver les sites à ouvrir afin de couvrir les clients au moindre coût. En définissant une matrice de recouvrement A, avec chaque élément représentant une donnée binaire $a_{ij} = 1$ si le site i peut couvrir le client j, un coût d'ouverture du dépôt i comme O_i et des variables de décision booléennes telles que x_i vaille 1 si le site i est ouvert, le problème peut se formuler de la manière suivante :

$$\min \sum_{i \in I} O_i x_i \tag{2.1}$$

Sous les contraintes :

$$\sum_{i \in I} a_{ij} x_i \geq 1 \qquad \forall j \in J \tag{2.2}$$

$$x_i \in \{0,1\} \qquad \forall i \in I \tag{2.3}$$

(2.1) est la fonction-objectif. (2.2) assurent que tous les clients sont servis au moins une fois et (2.3) sont les contraintes d'intégrité des variables.

Il est possible de considérer le recouvrement de sous-ensembles de clients non pas par un dépôt mais par une tournée (affectée éventuellement à un unique dépôt ou à des dépôts différents). Pour cela, les lignes de la matrice de recouvrement A représentent les clients, et $a_{ij} = 1$ si une tournée j sert le client i. L'ensemble des tournées possibles est énorme. Une approche appelée *génération de colonnes* permet de n'en générer qu'un sous-ensemble et de résoudre progressivement le programme linéaire.

Si l'inégalité triangulaire est respectée, on obtient une modélisation possible d'un problème de tournées classique décrit plus loin (*Vehicle Routing Problem - VRP*), dans lequel les clients doivent être servis par un véhicule de capacité limitée. Ce problème peut être vu comme un sous-problème du *LRP*. La solution du *SCP* peut se transformer en solution du *VRP* en n'affectant à une seule tournée les clients pour lesquels $\sum_{i \in I} a_{ij} x_i > 1$. Sinon, il faut avoir recours au problème de partitionnement (*Set Partitioning*) pour lequel la contrainte 2.2 est remplacée par $\sum_{i \in I} a_{ij} x_i = 1$.

Les problèmes de recouvrement sont *NP*-difficiles. Des méthodes de coupes pour le *SCP* sont proposées par Balas (1980), Balas et Ho (1980) et Balas et Ng (1989) par exemple. D'autres approches exactes sont aussi possibles, comme celles basées sur méthodes de séparation et évaluation (Harche et Thompson, 1994) ou des relaxations lagrangiennes (Balas et Carrera, 1996).

Les heuristiques les plus fréquentes pour le *SCP* sont celles du type gloutonne, comme l'heuristique de Chvàtal (1979), ou Beasley (1987). Il est ensuite possible d'améliorer la solution de l'heuristique par une recherche locale impliquant des échanges de dépôts.

L'utilisation de métaheuristiques est bien sûr aussi envisageable. Par exemple, Jaramillo *et al.* (2002) proposent l'utilisation d'algorithmes génétiques . Un état de l'art portant sur les algorithmes pour le *SCP* est fourni par Caprara *et al.* (2000). Cependant, les approches heuristiques les plus performantes pour ce type de problèmes sont basées sur une relaxation lagrangienne suivie d'une optimisation par sous-gradients (Ceria *et al.*, 1998; Caprara *et al.*, 1999; Cordone *et al.*, 2001).

2.3.2 Problèmes des p-centres et p-médianes

Si le nombre de dépôts à ouvrir est connu à l'avance et que les véhicules ne servent qu'un seul client à la fois (problème de type camion dédié ou *truckload*), le problème de localisation-routage peut se ramener au problème de type p-médianes dans lequel p dépôts sont à ouvrir de manière à couvrir l'ensemble des clients en minimisant le coût moyen des trajets. Si la fonction-objectif à optimiser est une minimisation du plus long trajet, le problème est du type

des p-centres. Les deux problèmes sont *NP*-difficiles. Une discussion sur la complexité des problèmes de type p-médianes est donnée dans Tamir (1993).

En utilisant les variables de décisions $y_{ij} = 1$ si le client j est affecté au site i, $x_i = 1$ si le site i est ouvert, ainsi qu'une variable z représentant le minimum des distances ou coûts c_{ij} entre les nœuds i et j, le problème des p-centres peut alors s'écrire ainsi :

$$\min z \tag{2.4}$$

Sous les contraintes :

$$\sum_{i \in I} y_{ij} = 1 \qquad \forall j \in J \tag{2.5}$$

$$\sum_{i \in I} x_i = p \tag{2.6}$$

$$y_{ij} \leq x_i \qquad \forall i \in I \quad \forall j \in J \tag{2.7}$$

$$z \geq \sum_{i \in I} c_{ij} y_{ij} \qquad \forall j \in J \tag{2.8}$$

$$x_i \in \{0,1\} \qquad \forall i \in I \tag{2.9}$$

$$y_{ij} \in \{0,1\} \qquad \forall i \in I \quad \forall j \in J \tag{2.10}$$

$$z \geq 0 \tag{2.11}$$

(2.4) est la fonction-objectif, (2.5) imposent que chaque client soit affecté à un site, (2.6) déterminent le nombre de sites à ouvrir, (2.7) indiquent qu'aucun client ne peut être affecté à un site fermé, (2.8) bornent les distances d'intervention, et (2.9), (2.10) et (2.11) sont les contraintes sur les variables.

Il peut être résolu en le transformant en une suite de problèmes de recouvrement (Handler, 1979). Des approches heuristiques ont été proposée dans la littérature pour ce problème, comme l'utilisation d'une recherche taboue ou d'algorithme de recherche sur voisinage variable (*Variable Neighborhood Search - VNS*) (Mladenović *et al.*, 2003).

Le problème des p-médianes, introduit par Hakimi (1964), peut se formuler ainsi :

$$\min \sum_{i \in I} \sum_{j \in J} c_{ij} y_{ij} \tag{2.12}$$

Sous les contraintes :

$$\sum_{i \in I} y_{ij} = 1 \qquad \forall j \in J \tag{2.13}$$

$$\sum_{i \in I} x_i = p \tag{2.14}$$

$$y_{ij} \leq x_i \qquad \forall i \in I \quad \forall j \in J \tag{2.15}$$

$$x_i \in \{0,1\} \qquad \forall i \in I \tag{2.16}$$

$$y_{ij} \in \{0,1\} \qquad \forall i \in I \quad \forall j \in J \tag{2.17}$$

(2.12) est la fonction-objectif, (2.13) imposent que chaque client soit affecté à un site, (2.14) détermine le nombre de sites à ouvrir, (2.15) indiquent que, si un site est fermé, aucun client ne peut lui être affecté, et (2.16) et (2.17) sont les contraintes d'intégrité des variables.

Parmi les méthodes de résolution du problème des p-médianes, on peut citer celle de Christofides et Beasley (1982). Des détails et des généralisations du problème peuvent être trouvées dans le livre de Mirchandani et Francis (1990). Plus récemment, une méthode par génération de colonnes a été conçue par Lorena et Senne (2004). Le problème-maître est une restriction du problème de départ optimisant un problème de 1-médiane devant couvrir un sous-ensemble de clients en satisfaisant des contraintes de capacité. Les nouvelles colonnes à générer sont apportées par la résolution du sous-problème de sac-à-dos. Une approche de *branch-and-price* (Senne *et al.*, 2005) a été publiée en suite de ces travaux.

Des métaheuristiques sont également développées comme le *TS* proposé par Rolland *et al.* (1997). Baldacci *et al.* (2002) proposent une méthode basée sur le partitionnement. Quant à Resende et Werneck (2004), ils élaborent une heuristique hybride qui combine différents éléments des métaheuristiques traditionnelles, à savoir une approche *multi-start* rappelant un *GRASP*, l'utilisation d'un algorithme de *Path Relinking* (*PR*) comme il est possible d'en voir dans les méthodes taboues et un algorithme de recherche dispersée dont les générations s'apparentent aux algorithmes génétiques. Un état de l'art sur les métaheuristiques pour le problème des p-médianes va paraître prochainement (Mladenović *et al.*, .à paraître).

2.3.3 Problème de localisation d'entrepôts

Dans le cas où les véhicules ne livrent qu'un client à la fois et où ni l'affectation des clients, ni le nombre de dépôts à ouvrir est pré-défini, on obtient le problème de localisation d'entrepôts (*location/allocation problem*). Il représente un cas particulier du *LRP* quand les clients sont servis par camion dédié (pas de tournées).

Le but consiste à minimiser à la fois les coûts d'ouverture des sites et ceux de transport (affectation des clients). Les dépôts peuvent être de capacité limitée ou non. Les clients peuvent être livrés soit par un seul dépôt (*single-source*), soit par plusieurs si des capacités l'imposent. Dans ce dernier cas, le problème déjà *NP*-difficile devient *NP*-difficile au sens fort (Mirchandani et Francis, 1990).

Pour se rapprocher de l'étude faite dans cette thèse, dans le modèle présenté ici, la livraison d'un client i, de demande d_j, doit être effectuée par un seul dépôt i de capacité limitée W_i et de coût d'ouverture O_i. En considérant, les coûts d'affectation c_{ij} d'un client j au dépôt i et les variables de décisions $y_j = 1$ si le site i est ouvert et x_{ij} représentant la quantité livrée par le site i au client j, on obtient le programme suivant :

$$\min \sum_{i \in I} O_i y_i + \sum_{i \in I} \sum_{j \in J} c_{ij} x_{ij} \tag{2.18}$$

Sous les contraintes :

$$\sum_{j \in J} d_j x_{ij} \leq W_i y_i \qquad \forall i \in I \tag{2.19}$$

$$\sum_{i \in I} x_{ij} = 1 \qquad \forall j \in J \tag{2.20}$$

$$y_i \in \{0,1\} \qquad \forall i \in I \tag{2.21}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in I \quad \forall j \in J \tag{2.22}$$

(2.18) est la fonction-objectif, (2.19) imposent que les capacités des dépôts soient respectées, (2.20) correspondent au respect des demandes et assurent que les clients ne soient affectés qu'à un seul dépôt, et (2.21) et (2.22) sont les contraintes d'intégrité des variables.

Des études sur les inégalités valides pour ce problème sont proposées par Leung et Magnanti (1989) et Aardal *et al.* (1995) par exemple. Mais quelque soit l'approche de résolution, exacte ou heuristique, la technique la plus courante est la relaxation lagrangienne de contraintes. Klincewicz et Luss (1986) relaxent les contraintes (2.19) afin d'obtenir un problème de localisation sans capacité et d'utiliser une résolution par l'algorithme de "*dual ascent*" (Erlenkotter, 1978). Cependant, c'est la relaxation de l'affectation des clients qui semble la plus pertinente. Pour chaque dépôt, un problème de sac-à-dos doit alors être résolu (Barcelo et Casanovas, 1984; Pirkul, 1987; Sridharan, 1993). Par exemple, Rönnqvist *et al.* (1999) utilisent cette relaxation en combinaison avec une heuristique de *repeated matching*, Holmberg *et al.* (1999) l'utilisent pour un algorithme de résolution exacte, et Cortinhal et Captivo (2003) s'en servent pour trouver des bornes au problème. Beasley (1993a) quant à lui propose une relaxation lagrangienne des deux contraintes et une résolution avec une méthode d'optimisation par sous-gradients.

Concernant les recherches locales employées, c'est souvent un échange de dépôts qui est utilisé, comme pour le problème des p-médianes. Resende et Werneck (2003) proposent une version rapide de cette méthode. Zhang *et al.* (2004) étendent le principe d'échange de deux dépôts à un échange multiple simultané.

Un récent état de l'art concernant les modèles de localisation est donné par Klose et Drexl (2005).

2.4 Problèmes de tournées

Le deuxième type de sous-problème que l'on peut identifier dans le *LRP* concerne l'élaboration des tournées, les véhicules utilisés pouvant livrer plus d'un client (problèmes aussi appelés *less than truck-load*). Il est possible de distinguer les problèmes à un véhicule (ce dernier servant l'ensemble des clients), et ceux où les tournées sont soumises à des contraintes de capacité imposant l'utilisation de plusieurs véhicules.

Un problème de tournées est souvent modélisé sur un graphe et parfois le nombre de véhicules à disposition au dépôt i est limité à nv_i. Une distinction possible est sur la notion de *client* à visiter. Il peut s'agir soit d'un arc soit d'un nœud. Dans cette partie, les problèmes formulés sur arcs ne seront pas développés puisque le *LRP*, tel que traité dans cette thèse,

est basé sur un ensemble de clients représentés par des nœuds. Le but est de construire des tournées de coût minimal, sachant qu'une tournée correspond à un véhicule partant et revenant au même dépôt, et traitant un certain nombre de clients. Une hypothèse courante est de supposer qu'un client ne doit être visité que par un seul véhicule et une seule fois. Les deux principaux types de problèmes de tournées rencontrés sont alors :

- *TSP ou Traveling Salesman Problem.* C'est le problème du voyageur de commerce, où tous les clients doivent être visités en une seule tournée.
- *VRP ou Vehicle Routing Problem.* C'est le problème classique de tournées de véhicules. Chaque client j a une demande connue d_j. Les véhicules ont une capacité limitées Q qui oblige à faire plusieurs tournées pour satisfaire les demandes.

Nous les détaillons dans la suite, en particulier celui qui nous concerne le plus, le *VRP*.

2.4.1 Problème du voyageur de commerce - *TSP*

Nous allons commencer par aborder le problème le plus simple, dans lequel une seule tournée doit être réalisée, le *TSP*. Il peut être vu comme un cas particulier du problème de localisation-routage dont la somme des demandes n'excède pas la capacité d'un véhicule et où un seul dépôt est disponible. Le but est de trouver un cycle de longueur totale minimale qui passe exactement une fois par chaque point. En considérant un ensemble de nœuds V et d'arcs A de coût c_{ij}, le problème peut se formuler ainsi, avec les variables binaires x_{ij} qui valent 1 si l'arc $(i,j) \in A$ est utilisé (Nemhauser et Wolsey, 1999) :

$$\min \sum_{i \in V} \sum_{j > i} c_{ij} x_{ij} \tag{2.23}$$

Sous les contraintes :

$$\sum_{i:(i,j) \in A} x_{ij} = 1 \qquad \forall j \in V \tag{2.24}$$

$$\sum_{j:(i,j) \in A} x_{ij} = 1 \qquad \forall i \in V \tag{2.25}$$

$$\sum_{(i,j) \in A : i \in U, j \in V \setminus U} x_{ij} \geq 1 \qquad \forall U \subset V \quad \text{avec } 2 \leq |U| < |V| - 2 \tag{2.26}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{2.27}$$

(2.23) est la fonction-objectif. Les contraintes (2.24) et (2.25) correspondent au degré des variables, assurant que pour chaque nœud, il y a un arc entrant et un sortant utilisé. Les contraintes (2.26) éliminent les sous-tours. Enfin, les contraintes (2.27) assurent l'intégrité des variables.

Bien que ce soit ć(2.23 36 TD[())]TJ8(c)e
v

été dédiées au *TSP*, comme par exemple Dantzig *et al.* (1954); Gomory (1958, 1963) ou Chvàtal (1973). Pour un problème asymétrique ($c_{ij} \neq c_{ji}$), une borne inférieure peut être trouvée en résolvant un problème d'affectation, et elle est d'ailleurs optimale si aucun sous-cycle n'apparaît dans la solution. Pour un problème symétrique, une borne inférieure peut être

x_{ij} qui valent 1 si l'arc $(i,j) \in A$ est utilisé (Laporte et Norbert, 1987; Toth et Vigo, 2001) :

$$\text{(VRP1)} \quad \min \sum_{i \in V \setminus \{n\}} \sum_{j>i} c_{ij} \cdot x_{ij} \tag{2.28}$$

Sous les contraintes :

$$\sum_{h<i} x_{hi} + \sum_{j>i} x_{ij} = 2 \qquad \forall i \in V \setminus \{0\} \tag{2.29}$$

$$\sum_{j \in V \setminus \{0\}} x_{0j} = 2 \cdot nv_0 \tag{2.30}$$

$$\sum_{i \in S} \sum_{h<i; h \notin S} x_{hi} + \sum_{i \in S} \sum_{j>i; j \notin S} x_{ij} \geq 2 \cdot r(S) \qquad \forall S \subseteq V \setminus \{0\} \quad S \neq \emptyset \tag{2.31}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in V \setminus \{0\} \quad i < j \tag{2.32}$$

$$x_{0j} \in \{0,1,2\} \qquad \forall j \in V \setminus \{0\} \tag{2.33}$$

(2.28) est la fonction-objectif. Les contraintes (2.29) et (2.30) assurent que deux arêtes utilisées sont incidentes aux clients et que nv_0 tournées sont bien reliées au dépôt. Les contraintes (2.31) imposent à la fois la connectivité de la solution et le respect des capacités sur le dépôt. Enfin, les contraintes (2.32) et (2.33) assurent l'intégrité des variables.

En considérant l'ensemble K de nv_0 véhicules disponibles, une autre formulation est possible. Elle est plus commode quand des contraintes complémentaires sont ajoutées ou quand les coûts de traversée des arêtes sont asymétriques. Elle utilise des variables à trois indices : x_{ijk} qui comptent le nombre de fois que l'arc $(i,j) \in A$ est traversé par le véhicule $k \in K$. Un autre type de variable est également introduit, y_{ik},$(i \in J; k \in K)$, prenant la valeur 1 si un client i est servi par le véhicule k. Le modèle est alors le suivant (Golden *et al.*, 1977) :

$$\text{(VRP2)} \quad \min \sum_{i \in V} \sum_{j \in V} c_{ij} \sum_{k \in K} x_{ijk} \tag{2.34}$$

Sous les contraintes :

$$\sum_{k \in K} y_{ik} = 1 \qquad \forall i \in V \setminus \{0\} \tag{2.35}$$

$$\sum_{k \in K} y_{0k} = nv_0 \tag{2.36}$$

$$\sum_{j \in V} x_{ijk} = \sum_{j \in V} x_{jik} = y_{ik} \qquad \forall i \in V \quad \forall k \in K \tag{2.37}$$

$$\sum_{i \in V} d_i y_{ik} \leq Q \qquad \forall k \in K \tag{2.38}$$

$$\sum_{i \in S} \sum_{j \notin S} x_{ijk} \leq |S| - 1 \qquad \forall S \subseteq V \setminus \{0\} \quad |S| \geq 2 \quad k \in K \tag{2.39}$$

$$y_{ij} \in \{0,1\} \qquad \forall i \in V \quad \forall k \in K \tag{2.40}$$

$$x_{ijk} \in \{0,1\} \qquad \forall i,j \in V \quad \forall k \in K \tag{2.41}$$

(2.34) est la fonction-objectif visant à minimiser les coûts. Les contraintes (2.35) à (2.37) imposent que chaque client soit visité exactement une fois, que nv_0 tournées soit réalisées et qu'un même véhicule arrive et reparte d'un nœud-client. Les contraintes (2.38) correspondent aux restrictions de capacité des véhicules. Les contraintes (2.39) sont les contraintes d'élimination des sous-tours. Enfin, les contraintes (2.40) et (2.41) assurent l'intégrité des variables.

Tout comme pour le *TSP*, il existe de nombreuses références sur le *VRP*. Encore une fois,

des tournées p et q. L'opération consiste alors à échanger les clients du premier groupe avec ceux du deuxième. La recherche lors d'une itération s'arrête quand un mouvement améliorant est identifié. De bons résultats sont obtenus par cette méthode mais elle n'atteint que très rarement les meilleures solutions connues.

Les réseaux de neurones n'ont pas été très étudiés pour le *VRP*. Les seuls articles à notre connaissance sont des généralisations des réseaux dédiés au *TSP* (Ghaziri, 1996).

Parmi les métaheuristiques les plus efficaces pour le *VRP*, celles de type tabou sont les plus répandues. Cordeau et Laporte (2002) répertorient dix des plus efficaces *TS* à l'époque de leur article. Pour n'en citer que trois, nous allons présenter l'algorithme de Taillard (1993), la *Taburoute* de Gendreau *et al.* (1994) et le *Granular Tabu Search - GTS* de Toth et Vigo (2003).

L'algorithme de Taillard (1993) définit son voisinage par le mécanisme de λ-*interchange* de Osman (1993). Il n'utilise pas de ré-optimisation, sauf ponctuellement avec l'algorithme exact du *TSP* de Volgenant et Jonker (1983), et il ne passe que par des solutions réalisables. Pour les problèmes euclidiens, la procédure utilise une décomposition en secteurs centrés sur le dépôt et avec des cercles concentriques, puis travaille dans chaque secteur, ceux-ci étant mis à jour régulièrement pour procurer une diversification. Pour les problèmes non-euclidiens, la décomposition est définie par le plus petit arbre recouvrant ayant le dépôt comme racine.

La Taburoute (Gendreau *et al.*, 1994) utilise un voisinage défini par toutes les solutions qui peuvent être atteintes depuis la solution courante en retirant un nœud d'une tournée et en l'insérant dans une autre contenant un de ses p plus proches voisins, puis cette tournée est ré-optimisée par le mécanisme GENI du *TSP* (Gendreau *et al.*, 1992). Les solutions obtenues peuvent violer les contraintes de capacité, mais ceci est pénalisé dans la fonction-objectif.

Enfin, la recherche de type tabou granulaire (*GTS*) de Toth et Vigo (2003) élimine du graphe initial les plus grands arcs reliant les nœuds afin de réduire la taille du voisinage de recherche, selon l'observation que les arcs les plus coûteux n'ont qu'une très faible chance d'appartenir à la solution optimale. En pratique, le choix des arcs éliminés est actualisé dynamiquement.

Les dernières métaheuristiques performantes que l'on peut citer pour le *VRP* sont celles utilisant une population de solutions. Elles ne sont pas les plus répandues actuellement, mais peuvent fournir de bons résultats. Les algorithmes génétiques sont très robustes. Le plus difficile est de trouver un codage des solutions sous forme de chaîne afin d'obtenir des *chromosomes* sur lesquels vont s'opérer des croisements.

Van Breedam (1996) utilise une séquence avec des duplications de dépôts pour séparer les différentes tournées. Lors des croisements, les solutions non réalisables sont alors rejetées. Plus récemment, Prins (2004) a proposé une version combinée avec une recherche locale (*algorithme mémétique*), conduisant à de très bons résultats. Il utilise un nouveau codage des solutions. La séquence des clients n'est plus représentée avec des duplications de dépôts délimitant les tournées. Pour retrouver la solution qui lui est associée, une procédure de découpage optimal de cette séquence en tournées est exécutée. Toujours sur la base génétique, pour le problème de tournées sur arc (*Capacitated Arc Routing Problem - CARP*), Prins *et al.* (2004b) ont ajouté une gestion de la population afin de travailler avec une population plus diversifiée. Ils obtiennent ainsi des résultats plus compétitifs qu'avec un *GA* classique en des temps bien

moins importants.

Un état de l'art sur les métaheuristiques utilisées pour le *VRP* est donnée par Gendreau *et al.* (2001). Plus dernièrement, une synthèse est réalisée par Crainic et Semet (2006) et Cordeau *et al.* (2005) comparent 9 méthodes récentes : des recherches taboues (Cordeau *et al.*, 2001; Tarantilis et Kiranoudis, 2002; Toth et Vigo, 2003), des méthodes évolutionnaires (Berger et Barkaoui, 2003; Prins, 2004; Mester et Bräysy, .à paraître), un algorithme de colonies de fournis (Reimann *et al.*, 2004) et une recherche sur grand voisinage (Ergun *et al.*, 2002). La conclusion est que les approches taboues, y compris la granulaire (Toth et Vigo, 2003), bien que les plus performantes dans le passé, commencent à être rattrapées, en particulier par des algorithmes à population. Par contre, les recherches taboues restent des approches simples (facile à reproduire) et flexibles (facilement adaptable en cas d'ajout de contraintes), bien que l'algorithme mémétique de Prins (2004) soit tout aussi compétitif sur ces points.

Tout comme pour le *TSP*, il est possible de trouver de nombreuses variantes du problème. A titre d'exemples, les versions pour des tournées avec collectes et livraisons (Lu et Dessouky, 2004), avec des fenêtres de temps (Cordeau *et al.*, 2001), ou encore des versions intégrant la planification de la production lors de l'élaboration des tournées de distribution (Boudia *et al.*, .à paraître) sont possibles. Un cas se rapprochant du *LRP* est celui du *VRP* multi-depot (Renaud *et al.*, 1996b; Mingozzi et Valletta, 2003). De plus amples détails sur le *VRP* sont donnés dans le livre de Toth et Vigo (2001).

2.5 Problèmes de localisation-routage

Après avoir regardé la littérature concernant les problèmes de localisation et ceux de tournées, cette section présente les travaux dédiés au problème combinant ces deux derniers aspects, à savoir le *LRP*. Ce problème consiste d'une part à déterminer l'ouverture d'un certain nombre de dépôts i, éventuellement de capacité limitée W_i, parmi un sous-ensemble I possible, et d'autre part à élaborer des tournées de véhicules de manière à visiter un ensemble J de n clients. Chaque client j a une demande d_j spécifique, et chaque tournée est effectuée par un véhicule, généralement de capacité limitée Q. Le but est minimiser le coût total, comprenant le coût d'ouverture des dépôts O_i, le coût fixe d'utilisation de véhicule F et la somme des coûts c_{ij} des arêtes traversées par le véhicule. Le problème visé dans cette thèse intègre des contraintes de capacité à la fois sur les dépôts et les véhicules.

Le paragraphe 2.5.1 donne d'abord un rapide historique sur l'émergence du *LRP*. Ensuite, les approches exactes existantes pour les versions classiques du problème sont exposées au paragraphe 2.5.2, suivies des résolutions heuristiques pouvant être trouvées dans la littérature au paragraphe 2.5.3. Un aperçu des travaux sur des variantes du problème est proposé au paragraphe 2.5.4. Enfin, des références concernant des états de l'art sur le *LRP* sont données au paragraphe 2.5.5.

2.5.1 Emergence du *LRP*

L'idée de combiner deux niveaux de décision logistique, la localisation de dépôts et l'élaboration de tournées de véhicules, date des années 1960. A cette époque, c'est essentiellement

la relation étroite entre la localisation et le transport qui est mise en évidence, mais la difficulté du problème est loin d'être cernée (Von Boventer, 1961; Maranzana, 1964; Webb, 1968; Lawrence et Pengilly, 1969; Christofides et Eilon, 1969; Higgins, 1972).

Watson-Gandy et Dohrn (1973) sont peut-être les premiers à réellement considérer la visite des clients lors la localisation, en utilisant une fonction non-linéaire des distances pour représenter les arrêts lors d'un trajet et en utilisant une *fonction-vente* dans laquelle les ventes déclinent quand la distance au dépôt augmente. Le but est alors de maximiser les profits.

Ensuite, d'autres travaux ont suivi dans le sens du *LRP*. Or et Pierskalla (1979) étudient un cas concret relatif à la localisation de banques de sang. Ils introduisent quelques simplifications au modèle, comme le fait de déterminer au préalable le nombre de banques à installer ou de supprimer la périodicité en supposant des livraisons journalières. Ils proposent alors une formulation en termes de flot et développent une heuristique de résolution itérative des sous-problèmes de tournées et d'affectation des clients (hôpitaux) aux banques de sang.

Jacobsen et Badsen (1980) abordent un autre cas réel en relation avec la livraison de journaux, avec un modèle à trois niveaux (imprimeries-points de transfert-clients). Le but est de localiser les points de transfert (dépôts) et de créer les tournées pour approvisionner ces dépôts et les clients. Ils comparent trois heuristiques. Dans la première, des tournées sont créées en se basant sur les localisations implicites des points de transfert. La deuxième est une procédure alternant entre localisation et affectation des clients suivie d'un algorithme de fusions pour former les tournées. Enfin, la troisième heuristique est constituée d'une procédure de fusion pour effectuer les tournées de livraison des clients, suivie une procédure dédiée à la localisation et une autre procédure du fusion pour les tournées d'approvisionnement des dépôts.

Nambiar *et al.* (1981) proposent des approches heuristiques pour une version simplifiée d'un problème concernant l'industrie du caoutchouc.

Enfin, on peut également citer Madsen (1983), qui nous donne un premier état de l'art et trois nouvelles heuristiques sur le sujet, et Perl et Daskin (1985) qui présentent une formulation sous forme de programme mathématique et proposent une résolution heuristique basée sur la décomposition en trois sous-problèmes (constitution de tournées en supposant tous les dépôts ouverts, localisation/affectation des tournées et enfin amélioration des tournées).

L'émergence de tous ces travaux donnent une certaine importance au *LRP*. Mais ce n'est qu'en 1989 par Salhi et Rand qu'est vraiment mis en évidence l'intérêt d'intégrer les tournées de véhicules lors de la phase de localisation des dépôts.

2.5.2 Approches exactes pour le *LRP*

Le *LRP* est un problème *NP*-difficile puisqu'il intègre des sous-problèmes eux-même *NP*-difficile. Dès le début des recherches à son propos, des approches exactes ont été développées.

Laporte et Norbert (1981) abordent la résolution du *LRP* avec un seul dépôt à ouvrir par un algorithme exact. Pour cela, ils formulent le problème sous forme de programme linéaire en nombres entiers et utilisent une méthode de coupes.

Laporte *et al.* (1986) étudient un *LRP* avec un nombre déterminé de véhicules par dépôt et donnent un algorithme exact de coupes partant de la relaxation de la plupart des contraintes initiales. Le modèle utilisé intègre des capacités à la fois sur les véhicules et les dépôts, comme pour le problème visé dans cette thèse. Il est donc intéressant de le présenter même si des contraintes particulières sont appliquées. En effet, le nombre de dépôts P utilisés doit se situer entre $\underline{P} \geq 1$ et $\overline{P} \leq |I|$, et les capacités sur ces derniers sont remplacées par un nombre de véhicules nv_i par dépôt i de coût F_i et compris entre $\underline{nv_i} \geq 1$ et $\overline{nv_i}$. En prenant L un nombre arbitraire relativement grand, le modèle peut se formuler ainsi :

$$\min z = \sum_{i \in I} (O_i \cdot y_i + F_i \cdot nv_i) + \sum_{i \in V} \sum_{j \in V} c_{ij} \cdot x_{ij} \tag{2.42}$$

sous les contraintes :

$$\sum_{i<k} x_{ik} + \sum_{k<j} x_{kj} = 2 \qquad \forall k \in J \tag{2.43}$$

$$\sum_{i<r} x_{ir} + \sum_{r<j} x_{rj} = 2 \cdot nv_r \qquad \forall r \in I \tag{2.44}$$

$$\sum_{i,j \in S} x_{ij} \leq |S| - \lceil D(S)/Q \rceil \qquad \forall S \subseteq J \quad |S| \geq 3 \tag{2.45}$$

$$x_{i_1 i_2} + 3x_{i_2 i_3} + x_{i_3 i_4} \leq 4 \qquad \forall i_1, i_4 \in I \quad \forall i_2, i_3 \in J \tag{2.46}$$

$$x_{i_1 i_2} + 3x_{i_{h-1} i_h} + 2 \sum_{i,j \in \{i_2, \ldots, i_{h-1}\}} x_{ij} \leq 2 \cdot h - 5 \qquad \forall h \geq 5 \quad i_1, i_h \in I \quad \forall i_2, \ldots, i_{h-1} \in J \tag{2.47}$$

$$y_i \leq nv_i \leq L \cdot y_i \qquad \forall i \in I \tag{2.48}$$

$$\underline{nv_i} \leq nv_i \leq \overline{nv_i} \qquad \forall i \in I \tag{2.49}$$

$$\underline{P} \leq \sum_{i \in I} y_i \leq \overline{P} \tag{2.50}$$

$$y_i \in \{0,1\} \qquad \forall i \in I \tag{2.51}$$

$$x_{ij} \in \{0,1\} \qquad \forall i \in V \quad \forall j \in V \tag{2.52}$$

$$x_{ij} \in \{0,1,2\} \qquad \forall i \quad ou \quad j \in I \tag{2.53}$$

La fonction-objectif (2.42) minimise les coûts fixes d'ouverture des dépôts et d'utilisation de véhicules, ainsi que les coûts des tournées. Les contraintes (2.43) et (2.44) correspondent au degré des variables et assurent ainsi que deux arêtes traversées sont incidentes aux clients (chaque client est intégré dans une tournée) et que nv_r tournées sont bien reliées au dépôt r. Les contraintes (2.45) imposent à la fois la connectivité de la solution et le respect des capacités sur les tournées. Les contraintes (2.46) et (2.47) sont appelées *chain barring* et assurent que chaque tournée débute et finit au même dépôt. Afin que les véhicules ne soient utilisés que s'ils sont affectés à un dépôt ouvert, les contraintes (2.48) doivent être respectées. Les contraintes (2.49) et (2.50) sont les bornes concernant le nombre de dépôts et de tournées ouverts. Enfin, les contraintes (2.51), (2.52) et (2.53) assurent l'intégrité des variables.

Toujours dans la même période, Laporte *et al.* (1988) ont proposé de transformer les

problèmes asymétriques de *VRP* multi-dépôt et de localisation-routage en problèmes d'affectation, par l'intermédiaire d'une représentation graphique appropriée. Les problèmes ainsi obtenus peuvent être résolus à l'aide d'une adaptation d'une procédure de séparation et évaluation dédiée au *TSP* asymétrique. Le retour à une solution du problème initial se fait par l'extension de l'arbre de recherche aux contraintes initialement relaxées. L'espace de recherche étant limité, l'approche est très rapide.

Plus récemment, Barreto (2004b) a proposé une borne inférieure pour le *LRP* à l'aide une méthode de coupes où un programme linéaire relaxé est résolu en nombres entiers. Cette borne est atteinte par les meilleures solutions heuristiques sur quelques petites instances (au maximum 36 clients et 5 dépôts).

Albareda-Sambola *et al.* (2005) proposent également deux bornes inférieures pour le *LRP* avec capacités uniquement sur les dépôts, et en autorisant une seule tournée par dépôt. La première borne est basée sur un programme linéaire relaxé. La seconde est composée de deux termes. Le premier concerne les coûts de transports entre clients. Un problème de voyageur de commerce asymétrique (*ATSP*) est résolu sur l'ensemble des clients et des dépôts potentiels. Pour cela, les coûts de transport sont mis à zéro sur les connexions directes avec un dépôt. Le second terme est lié aux coûts des dépôts et des ouvertures de tournées, et est résolu à l'aide d'une formulation sous forme d'un problème de sac-à-dos (*knapsack*).

2.5.3 Approches heuristiques pour le *LRP*

La plupart des travaux, en particulier récents, sont des approches de résolution de type heuristique. On peut cependant distinguer les recherches selon les contraintes de capacité utilisées.

2.5.3.1 Capacité limitée sur les véhicules ou les dépôts uniquement

En 1993, des heuristiques traitant le cas intégrant des capacités sur les véhicules commencent vraiment à paraître, comme par exemple l'étude de Chien (1993). Les auteurs utilisent une approche dans laquelle des solutions sont générées en commençant par regrouper aléatoirement les clients selon les capacités des tournées, puis par affecter les groupes aux dépôts. Pour obtenir le coût d'une solution, dans chaque regroupement de clients, le coût de la tournée est calculé soit en utilisant un estimateur des distances parcourues par les véhicules, soit en résolvant un *TSP*. Les meilleures solutions obtenues sont améliorées par recherche locale.

Srivastava (1993) compare trois heuristiques assez rapides pour le *LRP* avec capacités sur les véhicules. La première consiste à ouvrir tous les dépôts au départ, puis à les fermer un par un en choisissant celui offrant le meilleur gain estimé. L'élaboration des tournées est réalisée par un algorithme de type Clarke et Wright. Dans la seconde heuristique, au départ tous les dépôts sont fermés pour être ouverts ensuite un par un. Et enfin, la dernière heuristique groupe les clients en un nombre variable de *clusters* qui sont affectés à un dépôt (le plus proche).

Nagy et Salhi (1996) adoptent une approche plus spécifique visant à substituer l'approche en deux phases par une hiérarchisation des décisions. Ainsi, les tournées de véhicules deviennent un sous-problème de la localisation. Les coûts portent sur l'ouverture de dépôts et sur l'élaboration des tournées (sans coût fixe d'ouverture). Une capacité limitée et une distance maximale contraignent ces dernières. Le but est de déterminer le nombre et les dépôts à ouvrir, le nombre de véhicules à utiliser, ainsi que de construire des tournées. Dans la phase de localisation, à partir d'une solution de départ (ouverture de quelques dépôts et résolution d'un *VRP* multi-dépôt), une estimation des coûts engendrés par l'ouverture, la fermeture et l'échange de dépôts permet de choisir la meilleure amélioration possible. Dans la phase de routage, les tournées sont mises à jour en regroupant d'une part les clients situés à proximité des dépôts ouverts et qui leur seront automatiquement affectés, et les autres d'autre part. Les premiers sont utilisés pour construire des tournées, dans lesquelles les seconds sont ensuite insérés si les capacités le permettent. Sinon, de nouvelles tournées sont créées. Les tournées finalement obtenues sont améliorées par recherche locale. La méthode itère jusqu'à stabilisation de la fonction-objectif ou jusqu'à l'atteinte d'un certain nombre d'itérations.

Des métaheuristiques ont également été développées dernièrement. Tuzun et Burke (1999) proposent un algorithme de recherche taboue (*TS*) à deux phases pouvant résoudre des instances jusqu'à 200 clients. La méthode commence en n'autorisant l'ouverture que d'un seul dépôt, et alterne alors entre une phase de localisation de ce dépôt et une phase de routage. Quand la solution n'est plus améliorée, la recherche se poursuit en autorisant l'ouverture d'un dépôt supplémentaire. Quand à la fin d'une exploration avec un nombre donné de dépôts, aucune solution de coût inférieur à celle trouvée en utilisant un dépôt de moins n'a été visitée, alors l'algorithme s'arrête. Durant la phase de localisation, les mouvements appliqués sont soit une insertion soit un échange de dépôts, contrôlés par un *TS*. La phase de routage quant à elle commence par ré-affecter les clients au plus proche dépôt ouvert. Une valeur Δ_c est alors attribuée à chaque dépôt utilisé. Celle-ci correspond au nombre de changements d'affectation de clients sur ces dépôts. Puis des tournées sont recréées par un algorithme de fusion sur les dépôts ayant un Δ_c supérieur à un certain seuil. Sinon, les tournées initiales sont gardées et les clients devant être affectés sur ces dépôts sont ajoutés par un algorithme de meilleure insertion. Ensuite, des recherches locales de type insertion ou échanges de clients sont utilisées en limitant le voisinage aux clients appartenant aux dépôts avec un Δ_c supérieur à un certain seuil. Les modifications dans les tournées sont également contrôlée par un *TS*.

Albareda-Sambola *et al.* (2005) proposent aussi une heuristique de type tabou à deux phases (intensification avec une recherche sur les tournées et diversification avec une recherche sur les dépôts) pour un *LRP* avec des capacités sur les dépôts cette fois, permettant ainsi de n'avoir qu'une seule tournée par dépôt. Les instances utilisées sont de tailles relativement petites puisqu'elles ne dépassent pas 30 clients.

2.5.3.2 Capacité limitée sur véhicules et dépôts

Les références sur la littérature concernant les problèmes avec capacités à la fois sur les véhicules et sur les dépôts sont plus restreintes. Pour ce type de problème, on parlera de *LRP généralisé*.

Une heuristique originale pour un *LRP* généralisé à 3 niveaux (usines-dépôts-clients) avec

limitation de la durée des tournées est proposée par Bruns et Klose (1996). Il s'agit d'une méthode itérative de type *location first - route second*, dans laquelle des coefficients de coût pour servir les clients à partir d'un dépôt sont utilisés lors de la phase de localisation et ré-évalués après la phase de routage. Ainsi, la localisation est résolue indépendamment par une heuristique lagrangienne basée sur la relaxation des contraintes de capacité. Les dépôts ouverts sont ceux qui minimisent les coûts estimés de transport. Ensuite, une phase de routage utilise une procédure de construction de tournées suivie d'une recherche locale intégrant des fusions de tournées, 2-opt et échanges de clients entre dépôts tout en respectant les capacités. La ré-estimation des coûts de transport se fait selon les nouvelles tournées obtenues. La méthode s'arrête en cas de stabilisation des coefficients de coût ou d'atteinte d'un certain nombre d'itérations. Elle est évaluée sur des instances allant jusqu'à 10 usines, 50 dépôts potentiels et 100 clients.

Wu *et al.* (2002) proposent une métaheuristique pour un *LRP* généralisé avec flottes de véhicules hétérogènes de taille limitée. La méthode consiste à construire des solutions en découpant le problème : une localisation/affectation d'une part, où chaque tournée est agrégée en un nœud fictif de demande égale à la quantité livrée par le véhicule, et l'élaboration des tournées d'autre part. Chaque sous-problème est résolu alternativement en utilisant une approche par recuit simulé avec une liste taboue qui permet d'éviter les retours en arrière.

La même année et utilisant également un principe de recuit simulé, Lin *et al.* (2002) ont proposé une métaheuristique pour le *LRP* avec une considération en plus visant à minimiser le nombre de véhicules utilisé, pour une compagnie de service en télécommunication basée à Hong Kong, et désirant revoir la structure de son réseau d'impression et distribution des factures mensuelles.

Plus récemment, Barreto (2004b) a développé une famille d'heuristiques en quatre-phases basée sur des techniques de classification (clustering). Des groupes de clients respectant les capacités des véhicules sont d'abord formés. Un problème de voyageur de commerce est ensuite résolu dans chaque groupe. Les tournées obtenues sont améliorées dans la troisième phase. Enfin, la quatrième phase consiste à déterminer les dépôts à ouvrir en traitant un problème de localisation de dépôts, dans lequel les groupes de clients sont agrégés pour former des super-nœuds.

2.5.4 Diverses variantes du *LRP*

Dans les paragraphes précédents, nous avons exposé les travaux concernant des versions classiques du *LRP*. Dans la suite, des extensions sont succinctement exposées.

Divers travaux concernant le cas stochastique du *LRP* ont été réalisés depuis le milieu des années 70. Une méthode exacte par séparation et évaluation a été développée par Laporte *et al.* (1989). La demande n'est alors pas connue à l'avance et il faut déterminer à la fois la localisation des dépôts, la taille de la flotte de véhicules et les tournées. En approches heuristiques, on peut citer par exemple Harrison (1979) qui traite un cas concret pour la distribution de produits pharmaceutiques ou Chan *et al.* (2001) et Albareda-Sambola (2003).

Il existe quelques références sur la version dynamique du *LRP*, en particulier les travaux de Laporte et Dejax (1989). Le problème se pose sur un horizon temporel constitué d'un nombre

pré-défini de périodes. Les auteurs proposent alors deux approches. La première est exacte et consiste à représenter le problème par un réseau approprié et à résoudre optimalement un programme linéaire en nombres entiers sur ce réseau. Dans la seconde, certains coûts sont approximés et la solution globale est obtenue par un algorithme de plus court chemin sur un graphe orienté. Les problèmes résolus comprennent deux périodes, 400 clients et deux dépôts possibles par période.

Le *LRP* est généralement formulé en tant que problème de tournées sur nœuds. Cependant, dans les études récentes de Ghiani et Laporte (2001) et Labadi (2003), le problème traité considère des tournées sur arcs.

Les objectifs peuvent aussi être de diverses natures. Melechovský *et al.* (2005) ont développé une métaheuristique pour un *LRP* avec coûts non-linéaires. Le principe est de partir d'une solution obtenue par une approche de type p-médianes, puis d'améliorer cette solution initiale avec une recherche taboue utilisant un voisinage variable (*VNS*). Ceci permet une intensification ou diversification de la recherche.

Des versions multi-objectifs existent également. Une formulation sans capacité est présentée dans les travaux de List et Mirchandani (1991). Le cas intégrant des capacités limitées sur les dépôts est étudié par Caballero *et al.* (.à paraître). De récents travaux commencent à intégrer d'autres contraintes, par exemple Lin et Kwok (.à paraître) proposent des métaheuristiques pour une version dans laquelle les dépôts et les véhicules sous soumis à des contraintes de capacité. De plus, un même véhicule peut être utilisé pour effectuer plusieurs tournées.

Il existe aussi des méthodes adaptées pour le transport de produits dangereux ou non désirables (déchets). Dans ce cas, les dépôts ne doivent plus forcément être situés à proximité des clients. Le but est alors de minimiser les coûts tout en garantissant un certain périmètre de sécurité. Ce n'est pas le type de problème que nous étudions dans cette thèse, mais un article intéressant sur le sujet est celui de Cappanera *et al.* (2003), où une relaxation lagrangienne sur les contraintes de capacités des dépôts permet de séparer le problème en un sous-problème de localisation et un sous-problème sur les tournées, afin d'obtenir une borne inférieure utilisée dans une méthode de séparation et évaluation. Une heuristique lagrangienne est également présentée. Plus récemment encore, Alumur et Kara (.à paraître) proposent un nouveau modèle pour ce type de problème et Boulanger *et al.* (2006) présentent une métaheuristique bi-critère.

2.5.5 États de l'art pour le *LRP*

Pour une bibliographie plus exhaustive des premiers travaux sur le *LRP*, on peut se référer à l'article de Madsen (1983). Un peu plus tard, un autre état de l'art a été publié par Laporte (1988). Berman *et al.* (1995) fournit une synthèse des travaux concernant plus spécifiquement le cas stochastique.

Enfin, plus récemment, Min *et al.* (1998) nous donne une vision des études réalisées sur le *LRP* entre les années 1980 et le milieu des années 1990.

2.6 Conclusion

Le problème considéré dans cette thèse est multi-dépôt et multi-véhicule, chacun ayant une capacité limitée. Le nombre de dépôts à utiliser, leur localisation et les tournées à réaliser sont les décisions à prendre dans le but de minimiser les coûts totaux. On parlera alors de problème de localisation-routage généralisé.

L'intérêt d'une recherche sur le *LRP* est grandissant. En effet, comme nous l'avons dit en introduction, la combinaison de différents niveaux de décision dans la résolution de problèmes a souvent un impact financier intéressant. En effet, contrairement aux problèmes simplifiés de localisation qui approximent les coûts de transport en supposant des livraisons directes, le *LRP* prend en considération l'ordre de visite des clients. De plus, il est intéressant d'intégrer des contraintes de capacité à la fois sur les véhicules et les dépôts de manière à se rapprocher au maximum des problèmes réalistes. Cependant, comme le montre l'état de l'art réalisé, les approches effectuées dans le passé négligent souvent cette hypothèse.

Le *LRP* est un problème complexe. Bien qu'une approche en deux-phases soit possible, il est intéressant d'étudier des méthodes de résolution traitant simultanément les décisions de placement de dépôts et de construction/modifications de tournées, sans hiérarchisation. Cette thèse propose un approfondissement dans cette direction afin de développer de nouvelles approches efficaces pour des instances de grande taille. Enfin, les études concernant le problème de localisation-routage pouvant être vues comme des généralisations du *VRP* et de sa version multi-dépôt, la recherche dans le domaine peut fournir des méthodes transposables à ces cas particuliers.

Dans le chapitre suivant, la description plus précise du problème abordé dans la thèse est donnée avec son intérêt et des propositions de formulations mathématiques.

Chapitre 3

Problème de localisation-routage généralisé

3.1 Introduction

Avant de présenter dans les chapitres suivants les méthodes de résolution que nous avons développées pour le *LRP*, il est important de bien exposer le problème étudié et les hypothèses prises en compte. Pour cela, des formulations mathématiques sont proposées. La section 3.2 rappelle d'abord l'intérêt d'étudier les problèmes de localisation-routage. Ensuite, l'ensemble des données et contraintes sont introduites de manière mathématique afin de proposer une première modélisation dans la section 3.3. D'autres formulations sont suggérées dans les sections 3.4 et 3.5, avec leurs avantages par rapport aux propositions précédentes. Enfin, le chapitre se conclut par la section 3.6.

3.2 Intérêt du *LRP*

Comme il a déjà été dit dans les chapitres précédents, il est important de s'intéresser aux problèmes de logistique du transport. En effet, les coûts qui leurs sont liés représentent une part importante aussi bien d'un point de vue financier, avec la mondialisation du marché et l'impact dans les frais des entreprises, que d'un point de vue environnemental, avec les conséquences des émissions des gaz à effet de serre, dont le transport est en partie responsable.

Dans le passé, lors de la résolution de problèmes de logistique du transport, la démarche majoritairement employée a été de traiter chaque niveau de décision séparément, en commençant par le niveau stratégique. Les raisons de ce type d'approche sont exposées dans le chapitre 1 et semblent naturelles dans l'ensemble. Cependant, les approximations alors nécessaires engendrent souvent des solutions moins avantageuses en terme de coût. C'est d'ailleurs ce qui a été constaté lorsque l'on sépare les décisions de localisation de dépôts et celles concernant l'élaboration des tournées de véhicules (Salhi et Rand, 1989).

A première vue, résoudre un *LRP* peut pourtant sembler peu réaliste puisqu'il combine

des décisions à long terme (concernant les dépôts) et à plus court terme (concernant les tournées). Souvent les futurs clients et leur demande sont mal connus au moment du choix de l'emplacement des dépôts. Dans de telles conditions, le *LRP* peut être peu adapté. Par contre, dans les configurations où les tournées sont stables, comme en maintenance (relevé de compteurs) ou en collecte de déchets ménagers par exemple, la prise en compte des futures tournées au moment de la localisation des dépôts devient plus justifiée. De même, les coûts fixes d'ouverture d'un site peuvent être en fait des coûts d'exploitation pour une période donnée (location d'entrepôts existants par exemple). Ainsi, ces derniers deviennent d'un ordre de grandeur comparable aux coûts des tournées de la période considérée et le *LRP* est alors une formulation plus adaptée qu'une décomposition en localisation puis élaboration de tournées. Enfin, on peut citer le cas des secours humanitaires, où il faut situer temporairement des bases de ravitaillement et élaborer des tournées d'approvisionnement pour venir en aide aux populations dans le besoin.

La résolution du *LRP* représente donc un intérêt non négligeable dans divers cas de figure. Cependant, comme il a été montré dans le chapitre précédent, le nombre de travaux sur le sujet est relativement peu élevé, en particulier pour des cas intégrant des capacités limitées à la fois sur les véhicules et les dépôts. Pourtant, cette version, appelée *LRP* généralisé, est plus réaliste en logistique du transport. Pour ces raisons, ce type de problème est étudié dans cette thèse. La section suivante donne les notations et une première formulation mathématique possible du modèle étudié.

3.3 Description du problème

3.3.1 Modélisation des données et énumération des contraintes

Le *LRP* généralisé est défini ici sur un graphe non-orienté valué et complet, $G = (V,E,C)$. On considère un ensemble de nœuds V composé d'un sous-ensemble I de m dépôts possibles et d'un sous-ensemble $J = V \backslash I$ de n clients. On suppose que chaque client j a une demande d_j qui est connue à l'avance et peut être satisfaite. Un ensemble K de véhicules de capacité limitée à Q est disponible, chacun engendrant un coût fixe d'utilisation F (capacités et coûts fixes tous identiques - flotte homogène). Chaque site i susceptible d'être ouvert est également caractérisé par une capacité limitée W_i et un coût fixe d'ouverture O_i.

Les contraintes du problème sont les suivantes :

- chaque client doit être servi par un et un seul véhicule (il sera inclus dans une seule tournée);
- chaque tournée doit commencer et finir au même dépôt et la somme des demandes des clients la composant ne doit pas dépasser la capacité maximale Q du véhicule;
- la somme des charges des tournées affectées à un dépôt i ne doit pas excéder sa capacité maximale W_i.

Le but est alors de déterminer quels dépôts ouvrir (combien et sur quels nœuds du graphe) et quelles tournées construire (combien, associées à quels dépôts, et composées de quelles chaînes de clients) de manière à respecter les contraintes énumérées ci-dessus et à minimiser

le coût total, comprenant les coûts d'ouverture des dépôts, les coûts fixes d'utilisation de véhicules et la somme des coûts c_{ij} des arêtes traversées par les véhicules.

3.3.2 Modèle mathématique à 3 indices

On peut formuler le *LRP* sous forme d'un programme linéaire en nombres entiers dont les variables ont au plus trois indices. Toutes les variables sont binaires. y_i et f_{ij} valent 1 si le dépôt i est ouvert et si le client j est affecté au dépôt i respectivement. K est l'ensemble des tournées. x_{jlk} vaut 1 si et seulement si l'arête (j,l) est traversée de j vers l dans la tournée réalisée par le véhicule $k \in K$.

$$\text{(LRP1)} \qquad \min z = \sum_{i \in I} O_i y_i + \sum_{i \in V} \sum_{j \in V} \sum_{k \in K} c_{ij} x_{ijk} + \sum_{k \in K} \sum_{i \in I} \sum_{j \in J} F x_{ijk} \tag{3.1}$$

sous les contraintes :

$$\sum_{k \in K} \sum_{i \in V} x_{ijk} = 1 \qquad \forall j \in J \tag{3.2}$$

$$\sum_{j \in J} \sum_{i \in V} d_j x_{ijk} \leq Q \qquad \forall k \in K \tag{3.3}$$

$$\sum_{j \in V} x_{ijk} - \sum_{j \in V} x_{jik} = 0 \qquad \forall k \in K, \quad \forall i \in V \tag{3.4}$$

$$\sum_{i \in I} \sum_{j \in J} x_{ijk} \leq 1 \qquad \forall k \in K \tag{3.5}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ijk} \leq |S| - 1 \qquad \forall S \subseteq J, \quad \forall k \in K \tag{3.6}$$

$$\sum_{u \in V} x_{iuk} + \sum_{u \in V \setminus \{j\}} x_{ujk} \leq 1 + f_{ij} \qquad \forall i \in I, \quad \forall j \in J, \quad \forall k \in K \tag{3.7}$$

$$\sum_{j \in J} d_j f_{ij} \leq W_i y_i \qquad \forall i \in I \tag{3.8}$$

$$x_{ijk} \in \{0,1\} \qquad \forall i \in V, \qquad \forall j \in V, \qquad \forall k \in K \tag{3.9}$$

$$y_i \in \{0,1\} \qquad \forall i \in I \tag{3.10}$$

$$f_{ij} \in \{0,1\} \qquad \forall i \in I, \quad \forall j \in V \tag{3.11}$$

La fonction-objectif (3.1) reprend les coûts définis auparavant. Les contraintes (3.2) spécifient qu'un client n'appartient qu'à une et une seule tournée, et que chaque client n'a qu'un prédécesseur dans sa tournée, (3.3) et (3.8) concernent le respect des capacités. Les contraintes (3.4) représentent les conservations de flot et (3.5) servent à avoir des tournées n'appartenant qu'à un seul dépôt. Les contraintes (3.6) interdisent les sous-cycles. Les contraintes (3.7) spécifient que si une tournée relie un client j au dépôt i alors ce client est affecté à ce dépôt. En effet, la première somme vaut 1 si la tournée du véhicule k débute au dépôt i, et la seconde vaut 1 si le même véhicule k passe par le client j. Ainsi, si le dépôt i ne sert pas le client j

($f_{ij} = 0$), une des deux sommes du membre de gauche dans l'inégalité (3.7) doit être nulle, impliquant que la tournée k ne relie pas i à j. Enfin, les contraintes (3.9), (3.10) et (3.11) sont les contraintes d'intégrité des variables.

Ce modèle à trois indices n'est pas sans rappeler la formulation dite à trois indices du *VRP* (Golden *et al.*, 1977), rappelée au chapitre précédent. En effet, les variables x_{jlk} sont les mêmes pour les deux problèmes. Ainsi, certaines contraintes comme (3.1), (3.2), (3.3) et (3.6) sont de la même forme bien que, pour le *VRP*, des variables à deux indices soient également utilisées pour représenter les clients servis par un véhicule. Les modèles proposés par Bruns et Klose (1995) et par Wu *et al.* (2002) pour le *LRP* adoptent également des variables à trois indices mais ne peuvent être réutilisés sans adapter les contraintes à nos spécifications.

Le nombre de contraintes induites est de l'ordre de $2mn^2$ et le nombre de variables est important, de l'ordre de $(m+n)^3$. A cela s'ajoute le nombre exponentiel de contraintes de sous-tours. A part pour des instances de petite taille, cette formulation en 0-1 est donc hors de portée des solveurs commerciaux.

Mais ce n'est pas l'unique formulation possible de ce problème. D'autres modèles sont présentées aux paragraphes suivants, en particulier n'impliquant que des variables de décisions à deux indices.

3.4 Modèles mathématiques à 2 indices

Afin de faciliter la résolution, une formulation impliquant des variables à 1 ou 2 indices seulement semble plus judicieuse. Le but est d'abaisser le nombre de variables utilisées dans le modèle.

Deux versions sont ici exposées. Elles sont inspirées de celle développée dans les travaux de Laporte *et al.* (1986) sur le *LRP* généralisé et utilisée par Barreto (2004b). Une des différences vient du fait que dans ces précédents travaux, le nombre de véhicules utilisé par dépôt est limité. De plus, une généralisation de certaines contraintes est ici proposée.

Contrairement au modèle (LRP1) à trois indices, les deux versions à deux indices suivantes n'utilisent qu'une variable par arête, indépendamment du sens de traversée de cette dernière. La raison pour énoncer deux nouvelles versions vient de la gestion des éventuelles tournées ne visitant qu'un unique client. Pour une telle tournée, l'arête reliant le dépôt utilisé et le client servi est alors traversée deux fois, contrairement à tous les autres cas pour lesquels elle ne peut être utilisée qu'une seule fois. Ainsi, dans la formulation 1, les variables x_{ij} avec $i \in I$ et $j \in J$ appartiennent à {0,1,2} alors que dans la seconde formulation, toutes les variables sont binaires, moyennant l'introduction de w_{ij} pour gérer les doubles utilisations d'arêtes, comme décrit ci-après.

Mais avant d'exposer ces formulations, quelques notations doivent être introduites. Ce sont les suivantes :

- $\forall H \subseteq E$, $\quad x(H) = \sum_{(i,j)\in H} x_{ij}$;
- $\forall S \subseteq J$, $\quad D(S) = \sum_{j\in S} d_j$;
- $\forall S \subseteq V$, $\quad \delta(S)$ correspond à l'ensemble des arêtes avec une extrémité dans S et l'autre

dans l'ensemble $V \backslash S$;
- $\forall S \subseteq V$, $\gamma(S)$ représente l'ensemble des arêtes ayant leurs deux extrémités dans S;
- $\forall S \subseteq V$ et $\forall S' \subseteq V \backslash S$, $E(S : S')$ correspond à l'ensemble des arêtes avec une extrémité dans S et l'autre dans S'.

3.4.1 Formulation 1

Dans cette première formulation, les variables de décision suivantes sont utilisées : $y_i = 1$ si et seulement si le dépôt i est ouvert, $x_{ij} = 1$ ($i,j \in V$) si et seulement si un véhicule utilise l'arête (i,j) une et une seule fois, $x_{ij} = 2$ ($j \in J$, $i \in I$) si et seulement si un véhicule utilise une même arête (i,j) deux fois (si un véhicule ne sert qu'un seul client j à partir du dépôt i). Les variables x_{ij} avec $(i,j) \in I$ peuvent être exclues.

$$\text{(LRP2)} \quad \min \sum_{(ij)\in E} c_{ij}x_{ij} + \frac{F}{2}\sum_{i\in I}\sum_{j\in J} x_{ij} + \sum_{i\in I} O_i y_i \tag{3.12}$$

sous les contraintes :

$$x(\delta(j)) = 2 \qquad \forall j \in J \tag{3.13}$$

$$x(\delta(S)) \geq 2\lceil D(S)/Q \rceil \qquad \forall S \subseteq J \tag{3.14}$$

$$x_{ij} \leq 2y_i \qquad \forall i \in I, \quad \forall j \in J \tag{3.15}$$

$$x(\delta(S \cup \{i\})) \geq 2 \qquad \forall i \in I, \quad \forall S \subseteq J \quad | \quad D(S) > W_i \tag{3.16}$$

$$2x(\gamma(S \cup \{j\})) + x_{ij} - x(E(S : ((J \backslash (S \cup \{j\}))) \cup \{i\})) \leq 2|S| \qquad S \subseteq J\backslash\{j\} \qquad \forall i \in I \qquad S \neq \emptyset \tag{3.17}$$

$$x_{ij} \in \{0,1,2\} \qquad \forall i \in I, \quad \forall j \in J \tag{3.18}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in E \tag{3.19}$$

$$y_i \in \{0,1\} \qquad \forall i \in I \tag{3.20}$$

Dans ce programme linéaire, la fonction-objectif (3.12) minimise les coûts d'ouverture de tournées et de dépôts ainsi que les coûts pour visiter les clients. Les contraintes (3.13) concernent le degré des variables. Les contraintes (3.14) correspondent au respect des capacités des véhicules. En effet, le terme $\lceil D(S)/Q \rceil$ correspond au nombre minimum de véhicules nécessaires pour couvrir la demande des clients de l'ensemble S. Comme au moins deux arêtes sortent d'un ensemble de clients contenu dans une même tournée afin de la relier à un dépôt, le membre de gauche de l'inégalité doit être au moins égal au double du nombre de tournées minimum nécessaires pour servir l'ensemble S.

Les contraintes (3.15) imposent que les arêtes ne soient reliées aux dépôts que si ces derniers sont ouverts. Les contraintes (3.16), dans le même esprit que les contraintes (3.14), obligent à respecter les capacités des dépôts. (3.17) sont une nouvelle formulation de contraintes généralisant les propositions faites par Laporte *et al.* (1986) (voir chapitre 2). Elles restreignent l'affectation d'un véhicule à un seul dépôt et sont appelées contraintes *path-f1* car elles interdisent ainsi les chemins entre deux dépôts distincts, comme le montre la figure 3.1. (3.18 - 3.20) sont les contraintes d'intégrité des variables.

Preuve de validité des contraintes 3.17

Soit (x,y) une solution réalisable du problème, $x\,(\gamma(S \cup \{j\})) = A$ et $x(E(S : (J\backslash(S \cup \{j\})) \cup \{i\})) = B$. Alors, $A \leq |S|$ et $x_{ij} \leq 2$. Il est possible de considérer différents cas, selon la valeur de A et x_{ij} :

- Si $A = |\,S\,|$, alors il existe un chemin visitant tous les clients de $S \cup j$. Par conséquent, $x_{ij} = 2$, est impossible. Si $x_{ij} = 0$, la contrainte est valide. Si enfin $x_{ij} = 1$, le chemin part du dépôt i et doit y retourner, donc $B = 1$.
- Si $A \leq |\,S\,| - 1$, alors quelque soit la valeur de x_{ij} et B, la contrainte est valide

♢

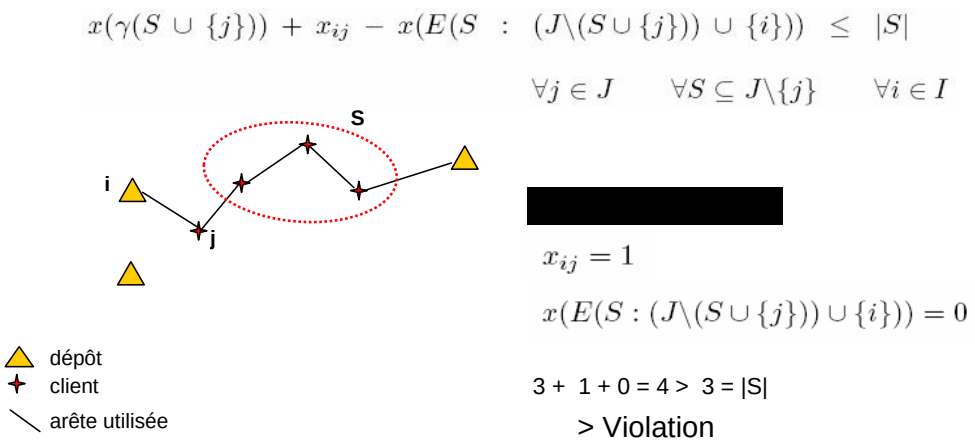


FIG. 3.1 – *Exemple de violation des contraintes de type path*

3.4.2 Formulation 2

Dans la seconde formulation, les mêmes variables y_i sont utilisées concernant les dépôts, ainsi que les mêmes variables x_{ij} $(i,j \in V)$ pour les arêtes (i,j) utilisées une unique fois. Par contre, de nouvelles variables binaires sont introduites : $w_{ij} = 1$ $(j \in J,\, i \in I)$ pour les arêtes utilisées deux fois (véhicules ne servant qu'un seul client j à partir d'un dépôt i). Les variables x_{ij} avec $(i,j) \in I$ peuvent ici encore être exclues du modèle.

$$\text{(LRP3)} \quad \min \sum_{(ij)\in E} c_{ij}x_{ij} + \sum_{i\in I}\sum_{j\in J} 2c_{ij}w_{ij} + \frac{F}{2}\sum_{i\in I}\sum_{j\in J}(x_{ij} + 2w_{ij}) + \sum_{i\in I} O_i y_i \qquad (3.21)$$

sous les contraintes :

$$\sum_{i \in I} 2w_{ij} + x(\delta(j)) = 2 \qquad \forall j \in J \tag{3.22}$$

$$\sum_{i \in I} \sum_{j \in S} 2w_{ij} + x(\delta(S)) \geq 2\lceil D(S)/Q \rceil \qquad \forall S \subseteq J \tag{3.23}$$

$$x_{ij} + w_{ij} \leq y_i \qquad \forall i \in I \quad \forall j \in J \tag{3.24}$$

$$\sum_{t \in I \setminus \{i\}} \sum_{j \in S} 2w_{tj} + x(\delta(S \cup \{i\})) \geq 2 \qquad \forall i \in I, \quad \forall S \subseteq J \quad | \quad D(S) > W_i \tag{3.25}$$

$$x(\gamma(S \cup \{j\})) + x_{ij} - x(E(S : (J \setminus (S \cup \{j\})) \cup \{i\})) \leq |S|$$

$$\forall j \in J, \quad \forall S \subseteq J \setminus \{j\}, \qquad \forall i \in I \tag{3.26}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in E \tag{3.27}$$

$$w_{ij} \in \{0,1\} \qquad \forall i \in I, \quad \forall j \in J \tag{3.28}$$

$$y_i \in \{0,1\} \qquad \forall i \in I \tag{3.29}$$

Comme précédemment, la fonction-objectif (3.21) minimise l'ensemble des coûts sur les tournées et les dépôts. (3.22) sont les contraintes concernant le degré des variables, (3.23) correspondent au respect des capacités des véhicules comme dans (LRP2), (3.24) imposent que les arêtes ne soient reliées aux dépôts que si ces derniers sont ouverts, (3.25) concernent les capacités des dépôts, dans le même esprit que pour les contraintes (3.23). Les contraintes (3.26) restreignent l'affectation d'un véhicule à un seul dépôt, elles sont appelées cette fois-ci contraintes *path-f2* car, tout comme dans (LRP2), elles interdisent en même temps les chemins entre deux dépôts distincts (voir la figure 3.1). (3.27) - (3.29) sont les contraintes d'intégrité des variables.

Preuve de validité des contraintes 3.26

Soit (x,w,y) une solution réalisable du problème, $x(\gamma(S \cup \{j\})) = A$ et $x(E(S : (J \setminus (S \cup \{j\})) \cup \{i\})) = B$. Alors, $A \leq |S|$ et $x_{ij} \leq 1$. Il est possible de considérer différents cas, selon la valeur de A et x_{ij} :

- Si $A = |S|$, alors il existe un chemin visitant tous les clients de $S \cup j$. Si $x_{ij} = 0$, la contrainte est valide. Si enfin $x_{ij} = 1$, le chemin part du dépôt i et doit y retourner, donc $B = 1$.
- Si $A \leq |S| - 1$, alors quelque soit la valeur de x_{ij} et B, la contrainte est valide

♢

3.4.3 Méthodes de résolution envisageables

Dans ces programmes linéaires, le nombre de variables n'est plus que de l'ordre de $(m+n)^2$ contre $(m+n)^3$ dans le programme (LRP1). L'amélioration est intéressante mais le modèle reste peu exploitable pour une résolution exacte par un solveur commercial. En effet, le nombre de contraintes de sous-tours reste exponentiel. Parmi les techniques efficaces pour réduire un

problème d'optimisation, l'une est de le réduire à une suite de problèmes plus simples, par le biais de restrictions ou de relaxations par exemple.

Lorsque les contraintes du problème étudié sont présentes en nombre exponentiel, comme c'est le cas pour le *LRP*, la stratégie la plus intéressante est certainement une méthode de coupes (voir le paragraphe 2.2.3 du chapitre précédent). Le principe consiste à relaxer des contraintes (principalement celles présentes en nombre exponentiel) et de résoudre la relaxation du programme linéaire en nombres entiers obtenu. Ensuite, il faut identifier une ou plusieurs contraintes violées dans le programme relaxé, les inclure dans le modèle, puis le résoudre à nouveau. Les contraintes violées sont identifiées dans des procédures spécifiques basées sur des inégalités valides, représentant ce qu'on appelle des coupes. Les familles d'inégalités valides sont souvent difficile à établir et les procédures d'identification sont souvent *NP*-complets, nécessitant le recours à des heuristiques. L'itération consistant à introduire de nouvelles contraintes est répétée idéalement jusqu'à obtenir une solution du programme linéaire relaxé respectant toutes les contraintes ignorées. Cette solution est alors optimale pour le problème initial.

Ce principe est applicable aux deux dernières formulations du *LRP*. Leur avantage par rapport à la version (LRP1), outre la réduction du nombre de variables, est que l'on se rapproche de la formulation efficace utilisée par Lysgaard *et al.* (2004) pour le *VRP*. Ces raisons encouragent la piste d'une méthode de coupes, même si les contraintes (3.17) et (3.26) par exemple sont nouvelles, spécifiques au *LRP*, et qu'elles nécessitent donc le développement de procédures d'identifications.

Une modélisation ne faisant intervenir que des variables à un indice est également possible comme présenté dans la suite, ouvrant la porte à de nouvelles pistes de résolution.

3.5 Autre modèle possible pour le *LRP*

Le *LRP* comprend différents sous-problèmes, notamment un de localisation et un de routage. Il est donc possible d'imaginer avoir un ensemble pré-défini de tournées respectant les contraintes de capacité des véhicules, et des variables de décision binaires caractérisant l'utilisation de ces tournées. Ainsi, l'objectif devient de sélectionner, à coût minimum, un ensemble de dépôts et de tournées de telle sorte que l'ensemble des clients soient couverts exactement une fois tout en respectant les capacités des dépôts. La formulation du *LRP* peut ainsi se faire par une approche inspirée du modèle de type *set partitioning* pour le *VRP* (Balinski et Quandt, 1964).

3.5.1 Modèle mathématique à 1 indice

En partant des considérations énoncées ci-dessus, on peut définir R comme un ensemble de tournées possibles et une matrice binaire d'affectation des clients aux tournées, telle que $\beta_{jr} = 1$ si et seulement si le client j appartient à la tournée r. Une nouvelle variable de décision binaire est alors introduite : x_r égale à 1 si et seulement si une tournée $r \in R$ de coût c_r est dans la solution. Une manière de formuler le *LRP* peut se faire en utilisant une matrice binaire de recouvrement des clients par les dépôts, composée d'éléments $\xi_{ij} = 1$ si et

seulement si le client j est couvert par le dépôt i :

$$\text{(LRP4)} \qquad \min \sum_{i \in I} O_i y_i + \sum_{r \in R} c_r x_r \tag{3.30}$$

sous les contraintes :

$$\sum_{i \in I} \xi_{ij} y_i = 1 \qquad \forall j \in J \tag{3.31}$$

$$\sum_{j \in J} d_j \xi_{ij} y_i \leq W_i y_i \qquad \forall i \in I \tag{3.32}$$

$$\sum_{r \in R} \beta_{jr} x_r = \sum_{i \in I} \xi_{ij} y_i \qquad \forall j \in J \tag{3.33}$$

$$x_r \in \{0,1\} \qquad \forall r \in R \tag{3.34}$$

$$y_i \in \{0,1\} \qquad \forall i \in I \tag{3.35}$$

Ici, la fonction-objectif (3.30) minimise les coûts d'ouverture des tournées et des dépôts. Les contraintes (3.31) garantissent que chaque client est servi par un et un seul dépôt. Les contraintes (3.32) assurent que la capacité des dépôts couvre la demande des clients. Les contraintes (3.33) impliquent que les clients inclus dans les tournées reliées à un dépôt soient effectivement affectés à ce dépôt. Enfin, les contraintes (3.34) et (3.35) définissent les variables binaires.

En supposant connu l'ensemble des dépôts ouverts, il est possible de résoudre le sous-problème impliquant les variables x_r, c'est-à-dire :

$$\text{(PLSP)} \qquad \min \sum_{r \in R} c_r x_r$$

sous les contraintes :

$$\sum_{r \in R} \beta_{jr} x_r = \sum_{i \in I} \xi_{ij} y_i = 1 \qquad \forall j \in J$$

$$x_r \in \{0,1\} \qquad \forall r \in R$$

Ce programme a une expression simple puisqu'il s'agit d'un problème de partitionnement mais, en pratique, il s'agit d'un VRP multi-dépôt et le nombre de tournées possibles dans R est très grand. Il faut avoir recours à une méthode de génération de colonnes pour le résoudre optimalement.

3.5.2 Méthode de résolution envisageable

La fonction-objectif de (LRP4) est composée de deux termes, chacun n'impliquant qu'un seul type de variables. De plus, seules les contraintes (3.33) et (3.34) font intervenir les variables x_r. (LRP4) peut donc se décomposer en deux parties, dont une est basée sur (PLSP). Une telle

approche séparant les variables peut faire penser à la décomposition de Benders, impliquant un problème-maître et un sous-problème, résolus alternativement. Dans notre cas, le problème-maître fournit un sous-ensemble de dépôts ouverts. Le sous-problème apporte des contraintes au problème-maître par la résolution de son programme dual. Pour une description du principe général, on peut se référer aux articles de Geoffrion (1970a,b).

(LRP4) peut se formuler en fonction des variables y uniquement. Pour cela, les contraintes impliquant x sont projetées sur l'espace des variables y. On obtient ainsi le programme suivant :

$$\text{(LRP4b)} \qquad \min \sum_{(i)\in I} O_i y_i + \inf_{x\in\{0,1\}} \{\sum_{r\in R} c_r x_r \text{ sous les contraintes } (3.33-3.34)\}$$

sous les contraintes :

$$\begin{aligned} \sum_{i\in I} \xi_{ij} y_i = 1 \qquad & \forall j \in J \\ \sum_{j\in J} d_j \xi_{ij} y_i \geq W_i y_i \qquad & \forall i \subseteq I \\ y_i \in \{0,1\} \qquad & \forall i \in I \end{aligned}$$

Le terme Inférieur de la fonction-objectif correspond au programme linéaire (PLSP) qui représente le sous-problème de la décomposition de Benders. En théorie, sa résolution se fait en passant par le dual de sa relaxation linéaire :

$$\text{(DPLSP)} \qquad \max \sum_{j\in J} \mu_j$$

sous les contraintes :

$$\begin{aligned} \sum_{j\in J} \mu_j \beta_{jr} \leq c_r \qquad & \forall r \in R \\ \mu_j \leq 0 \qquad & \forall j \in J \end{aligned}$$

On peut ainsi noter U l'ensemble des μ_j ($\forall j \in J$), $PE \subseteq U$ l'ensemble des points extrêmes et $RE \subseteq U$ l'ensemble des rayons extrêmes de la région réalisable de (DPLSP).

La valeur d'une solution optimale de (DPLSP) pour des y donnés (donc à une itération t de la résolution itérative de la décomposition de Benders) est notée Q^t. Ainsi, sans fixer les variables y, à une itération donnée t le problème-maître s'écrit sous la forme :

$$\text{(PLPM)} \qquad \min \sum_{i\in I} O_i y_i + Q^t$$

sous les contraintes :

$$\sum_{i \in I} \xi_{ij} y_i = 1 \qquad \forall j \in J$$

$$\sum_{j \in J} d_j \xi_{ij} y_i \geq W_i y_i \qquad \forall i \subseteq I$$

$$Q - \sum_{j \in J} (\mu_j)^t \geq 0 \qquad \forall (\mu_j)^t \in PE \tag{3.36}$$

$$\sum_{j \in J} (\mu_j)^t \leq 0 \qquad \forall (\mu_j)^t \in RE \tag{3.37}$$

$$y_i \in \{0,1\} \qquad \forall i \in I$$

En fait, si (DPLSP) n'est pas réalisable ou n'admet pas d'optimum fini, son primal est irréalisable et la fonction-objectif du problème-maître est non bornée. Il faut alors ajouter au problème maître (PLPM) les contraintes liées aux rayons extrêmes de la région réalisable du dual (contraintes (3.37)). Sinon, il faut ajouter les contraintes (3.36) liées aux points extrêmes.

3.6 Conclusion

Dans ce chapitre, le *LRP* généralisé traité dans cette thèse est présenté et des modèles mathématiques sont exposés. Des idées de résolution sont aussi proposées au vu des formulations. Il est certainement possible d'énoncer encore d'autres modèles du *LRP*. Seuls sont exposés ici les programmes linéaires ayant fait l'objet de recherches plus approfondies. En particulier, une approche de coupes basée sur les formulations à deux indices a été développée avec J.M. Belenguer et E. Benavent de l'Université de Valence (Espagne). Elle est d'ailleurs détaillée au chapitre 8. L'idée de la décomposition de Benders a été abordée lors d'un séjour de 6 semaines début 2005 au Centre de Recherche sur les Transports à Montréal, durant lequel d'autres travaux (voir méthode du chapitre 7) ont été menés en collaboration avec P. Soriano et A. Ruiz. Les recherches sur la décomposition de Benders n'ont à ce jour pas été suffisamment approfondies pour donner lieu à un chapitre spécifique.

La méthode coopérative du chapitre 7 est également basée sur un programme linéaire en nombres entiers, mais c'est une relaxation de (LRP1) utilisée uniquement pour la partie localisation. Par contre, les chapitres 4, 5 et 6 sont consacrés à des approches de type heuristiques développées durant cette thèse et s'appuient directement sur le modèle de graphe, non sur des programmes linéaires.

Chapitre 4

Heuristiques

4.1 Introduction

Après avoir présenté plus en détail le *LRP* généralisé à l'aide de quelques formulations mathématiques, ce chapitre est consacré aux premières méthodes de résolution développées pour le problème pendant la thèse. Il s'agit d'heuristiques et de recherches locales, par opposition aux algorithmes plus complexes présentés dans les chapitres qui suivront.

Les heuristiques sont des méthodes approchées qui permettent d'obtenir des solutions de bonne qualité, mais sans garantie d'optimalité. La construction d'une telle solution résulte en général de décisions élémentaires consécutives. Elles sont souvent complétées par des recherches locales effectuant des améliorations successives. Le résultat est en général un minimum local. Le principe de ces deux techniques est donné au paragraphe 2.2.3.2 du chapitre 2.

Les approches proposées ici sont de deux grand types : des heuristiques constructives (section 4.2) et de fusions (section 4.3). Elles construisent une seule solution réalisable. Ces algorithmes seront surtout utiles pour obtenir rapidement des solutions de bonnes qualité pour initialiser des recherches locales ou des métaheuristiques plus performantes. Des recherches locales sont décrites dans la section 4.4. Nous en proposons ici deux types : le premier améliorant uniquement le routage, et le second portant à la fois sur la localisation et les tournées. Les résultats obtenus par ces différentes techniques sont exposés dans la section 4.5 avant de terminer par une conclusion en section 4.6.

4.2 Heuristiques constructives

Trois versions d'heuristiques constructives sont développées ici. Les tournées sont toutes élaborées sur le principe de l'algorithme du Plus Proche Voisin (*PPV*), bien connu pour le *TSP*, et rappelé au paragraphe 2.2.3.2 du chapitre 2. Pour le *LRP*, à chaque itération le principe est toujours de compléter une tournée en choisissant, parmi les clients non encore introduits dans la solution, celui le plus proche du dernier client inséré. Cependant, cette fois-ci, ce n'est pas une chaîne hamiltonienne qui doit être créée, mais un ensemble de cycles

de capacité limitée, reliés potentiellement à différents dépôts. L'initialisation d'un cycle varie selon la version de l'algorithme utilisée, comme décrit plus loin. Le critère d'arrêt est le suivant. Si l'insertion du plus proche client libre viole une des capacités limitées (sur le véhicule servant la tournée en cours ou sur le dépôt auquel cette tournée est rattachée), alors le cycle se termine par un retour au dépôt de départ. Une nouvelle tournée est alors initialisée. L'algorithme s'arrête quand tous les clients sont insérés dans une tournée. La différence entre les trois versions proposées ci-après vient essentiellement de la sélection des dépôts à ouvrir.

Les deux premières fonctionnent sur une idée commune : choisir les dépôts à ouvrir les uns après les autres. Quand un dépôt est ouvert, les nouveaux cycles sont initialisés en partant de ce dépôt pour aller au plus proche client non encore affecté dans une tournée. Les cycles sont complétés comme décrit précédemment. Quand une tournée se termine pour cause de saturation de la capacité du dépôt, un nouveau dépôt est ouvert. La différence entre ces deux versions vient du fait que la première (algorithme de base 1 - *AB1*) adopte une démarche déterministe dans la sélection du prochain dépôt à ouvrir, alors que le second (algorithme de base 2 - *AB2*) est randomisé. La structure générale d'*AB1* et *AB2* est donné dans l'algorithme 1.

Algorithme 1 Heuristique constructive *AB1* et *AB2* - Capacité limitée sur les dépôts

$CoutSol = 0$ //coût total
$DepRest = I$ //dépôts restants
$CltRest = J$ //clients restants
répéter
  Choisir un dépôt i dans DepRest //(Seule différence entre AB1 et AB2, voir texte)
  $y_i = 1$
  $CoutSol = CoutSol + O_i$
  $DepRest = DepRest \backslash \{i\}$
  $CapDep = W_i$ //capacité résiduelle du dépôt i
  répéter
    //création d'une nouvelle tournée pour le dépôt i
    $CoutTour = F$ //coût de la tournée
    $CapTour = Q$ //capacité résiduelle de la tournée
    $k = i$ //nœud courant = dépôt
    répéter
      $j = argmin\{c_{kp} | p \in CltRest\}$
      si ($d_j \leq min(DepRest, CapTour)$ **alors**
        Insérer le client j après k dans la tournée
        $CoutTour = CoutTour + c_{kj}$
        $CapTour = CapTour - d_j$
        $CapDep = CapDep - d_j$
        $CltRest = CltRest \backslash \{j\}$
        $k = j$
      fin si
    jusqu'à ($d_j > CapDep$ ou $d_j > CapTour$ ou $CltRest = \emptyset$)
    $CoutTour = CoutTour + c_{ki}$ //retour au dépôt
    $CoutSol = CoutSol + CoutTour$
  jusqu'à ($CltRest = \emptyset$ ou $d_j > CapDep$)
jusqu'à ($CltRest = \emptyset$)

Dans *AB1*, le choix de l'ouverture d'un dépôt i se détermine par le calcul d'un score,

qui peut être vu comme un coût estimé du dépôt. Il correspond au rapport entre le coût d'ouverture du dépôt O_i et sa capacité W_i, ajouté aux coûts d'affectation des clients libres (ensemble *CltRest*) à ce dépôt, pondéré par les demandes :

$$score(i) = O_i/W_i + \sum_{j \in CltRest} d_j.c_{ij} \qquad \forall i \in DepRest$$

Le dépôt qui sera ouvert est celui de score minimal. Ce calcul est refait pour chaque ouverture de dépôt puisque les clients restants ne sont plus les mêmes. Par conséquent, les scores changent de valeur. Les deux aspects du problème que sont la localisation et le routage ne sont ainsi pas totalement dissociés : on considère les dépôts à ouvrir au fur et à mesure de la construction de la solution en fonction des clients restants.

Dans le deuxième algorithme, *AB2*, le choix des dépôts se fait aléatoirement. Il n'y a donc plus de calcul de score. L'intérêt est de pouvoir construire des solutions différentes lors de chaque appel et ainsi de proposer des alternatives comme solutions de départ pour des méthodes à population ou itératives, par exemple.

Enfin, le troisième algorithme (algorithme de base 3 - *AB3*) trouve son intérêt quand les dépôts ont une capacité illimitée, ce qui est le cas pour certaines instances de la littérature. En effet, avec la manière de procéder décrite précédemment, la méthode ouvre un seul dépôt et lui affecte l'ensemble des tournées. En général, le coût de routage alors engendré est plus important (l'affectation des clients étant imposée à cet unique site) et n'est justifié que si les dépôts ont des coûts d'ouverture suffisamment importants pour que l'objectif soit d'en minimiser le nombre. Ce n'est cependant pas toujours le cas. Afin de permettre l'ouverture de plusieurs sites, le principe d'ouverture des dépôts des versions précédentes est modifié. *AB3* débute par l'ouverture un certain nombre de sites (quantité et localisation aléatoire). Par des tests sur les instances à notre disposition, il semble que l'ouverture randomisée de 2 ou 3 dépôts fournisse de bons résultats. Le principe de *AB3* est donné dans l'algorithme 2. Contrairement à $AB1$ et $AB2$, chaque nouvelle tournée est démarrée en choisissant d'abord un client j (ligne 8) et en calculant le dépôt i le plus proche (ligne 9). On complète ensuite la tournée commençant par les nœuds i et j avec le principe du *PPV*.

4.3 Heuristique de fusions

Une autre approche possible pour obtenir rapidement des solutions réalisables est la fusion de tournées. La plus classique, et donnant de très bons résultats pour le *VRP* (Laporte *et al.*, 2000), est l'algorithme de Clarke et Wright (1964), noté *CWA* dans la suite. Cependant, la version classique a dû être généralisée de manière non triviale afin d'intégrer les aspects multi-dépôt et localisation.

4.3.1 Rappel de l'algorithme de Clarke et Wright pour le *VRP*

L'algorithme classique de Clarke et Wright (1964) commence par dédier une tournée à chaque client de façon à le relier à l'unique dépôt. Le résultat donne une solution initiale

Algorithme 2 Heuristique constructive *AB3* - Capacité illimitée sur les dépôts

$CoutSol = 0$ //coût total
$DepOuv = \emptyset$ //dépôts ouverts
$CltRest = J$ //clients restants
ChoisirDepots(DepOuv) //choisir aléatoirement les dépôts à ouvrir et les mettre dans $DepOuv$
$CoutSol = CoutSol + \sum_{i \in DepOuv} O_i y_i$
répéter
  //création d'une nouvelle tournée
  Choisir un client j dans CltRest
  $i = argmin\{c_{jp} | (p \in DepOuv)\}$
  Insérer le client j après le dépôt i dans la tournée
  $CoutTour = F + c_{ij}$
  $CapTour = Q - d_j$
  $CltRest = CltRest \backslash \{j\}$
  $k = j$
  répéter
    $j = argmin\{c_{kp} | (p \in CltRest)\}$
    si $(d_j \leq CapTour)$ **alors**
      Insérer le client j après k dans la tournée
      $CoutTour = CoutTour + c_{kj}$
      $CapTour = CapTour - d_p$
      $CltRest = CltRest \backslash \{j\}$
      $k = j$
    fin si
  jusqu'à $(d_j > CapTour$ ou $CltRest = \emptyset)$
  $CoutTour = CoutTour + c_{ki}$ //retour au dépôt
  $CoutSol = CoutSol + CoutTour$
jusqu'à $(CltRest = \emptyset)$

triviale ayant une allure de *marguerite* (voir Figure 4.1)

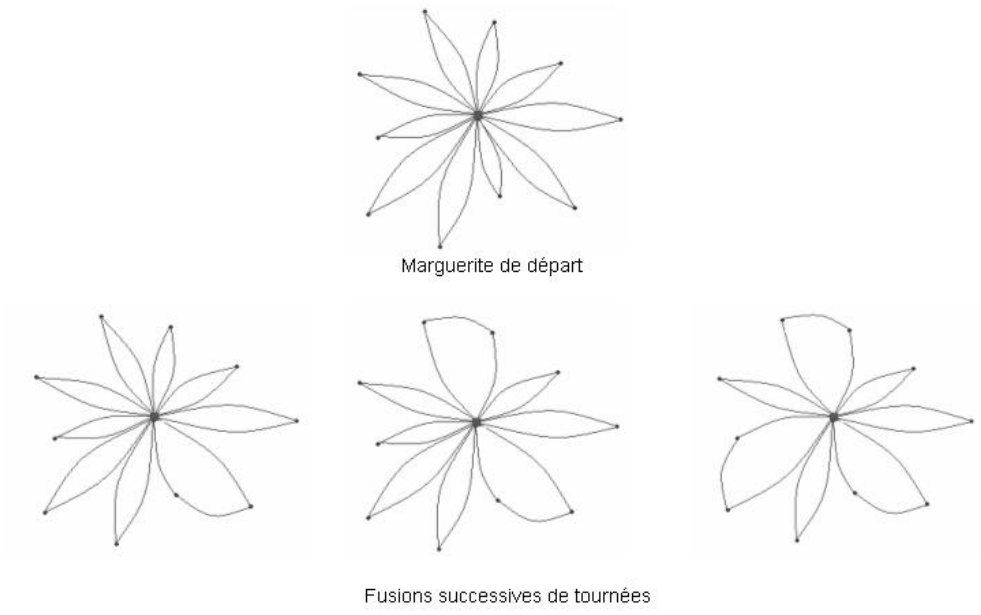


FIG. 4.1 – *Marguerite et fusions dans CWA pour le VRP*

Ensuite, le but est de fusionner les tournées obtenues afin de réduire le coût de la solution courante. Pour le *VRP*, on effectue successivement les fusions réalisables (compatibles avec les capacités des véhicules) et de gain maximal, jusqu'à ce qu'on n'en trouve plus ou que le nombre de tournées désiré soit atteint.

Une itération de l'algorithme consiste à examiner toutes les fusions réalisables et à effectuer celle qui produit le plus gros gain. Pour chaque paire de tournées, 4 fusions sont évaluées. En effet, comme on n'a pas de précédence sur les clients et que le graphe est non-orienté, chacune des tournées peut être considérée indifféremment dans un sens de parcours ou l'autre. Dans la version classique, le gain G réalisé par la fusion de deux tournées R et S (voir partie supérieure de la figure 4.2) peut se calculer ainsi, avec j et k respectivement le dernier client de la tournée R et le premier de la tournée S, et s désignant le dépôt :

$$G = F + c_{js} + c_{sk} - c_{jk} \tag{4.1}$$

Le *CWA* peut être programmé sans précaution en $O(n^3)$ ou mieux en $O(n^2 \log n)$ en utilisant une structure de données appelée tas (Ghiani et Laporte, 2001).

4.3.2 Algorithme généralisé de Clarke et Wright

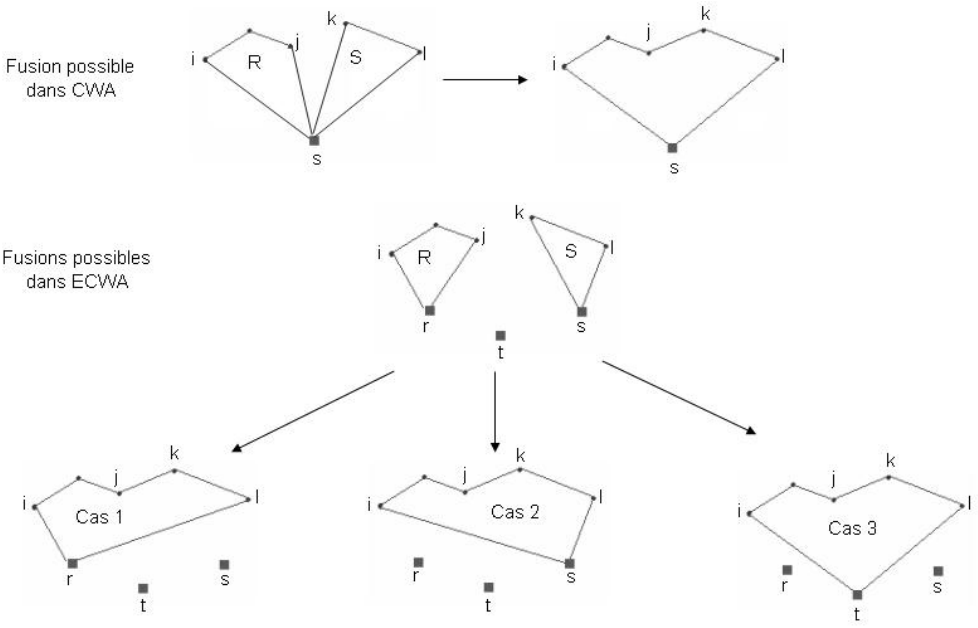


FIG. 4.2 – *Fusions dans CWA et ECWA*

Notre algorithme de Clarke et Wright généralisé (*Extended Clarke and Wright Algorithm - ECWA*) pour le *LRP* est basé sur la version originale (*CWA*), mais il intègre en plus les aspects multi-dépôt et localisation.

Pour l'initialisation de la solution, ce n'est donc plus une marguerite qui est d'abord construite, mais un *bouquet de marguerites*. En fait, plusieurs dépôts étant disponibles au départ, ils sont tous considérés ouverts a priori, et les clients ne sont plus tous reliés à un unique site. Ces derniers sont traités dans un ordre aléatoire et chaque client est affecté au plus proche dépôt ayant encore suffisamment de capacité résiduelle pour le servir. Quand tous les clients sont affectés, les dépôts n'en servant aucun sont considérés comme fermés. En général, plusieurs sites sont ouverts constituant ainsi un ensemble de marguerites.

Ensuite, comme dans le *CWA*, une itération de l'algorithme examine toutes les fusions possibles et réalise celle qui produit le plus gros gain G. Mais cette fois-ci, pour chaque paire de tournées, ce n'est pas 4 mais $4m$ fusions qui sont envisageables (avec m, le nombre de dépôts disponibles). En effet, la tournée résultant d'une fusion peut maintenant être affectée à n'importe quel dépôt disponible de capacité compatible.

Soit par exemple deux tournées : R affectée au dépôt r, commençant par le client i et terminant par le client j, et S affectée au dépôt s, commençant par le client k et terminant par le client l, le résultat de la fusion donne la tournée T pouvant être affectée au dépôt t

(voir partie inférieure de la figure 4.2). Le gain G réalisé peut donc se calculer ainsi :

$$G = F + c_{ri} + c_{jr} + c_{sk} + c_{ls} - c_{jk} - c_{ti} - c_{lt} + O_r\lambda_r + O_s\lambda_s - O_t(1 - y_t) \qquad (4.2)$$

λ_r (respectivement λ_s) est un booléen égal à 1 si et seulement si le dépôt r (respectivement s) n'a plus de tournée après la fusion et peut donc être fermé. La variable booléenne y_t est celle introduite dans le modèle linéaire présenté en section 3.3 du chapitre précédent : elle est égale à 1 si et seulement si le dépôt t est déjà ouvert avant la fusion. La complexité de *ECWA* est $O(mn^3)$.

4.4 Recherches locales

4.4.1 Principes

En s'inspirant des recherches locales pour le *TSP*, le *VRP*, le *MDVRP* et les problèmes de localisation, on peut envisager des *mouvements élémentaires* de deux types : ceux qui modifient les tournées mais sans toucher aux dépôts, et ceux qui changent l'ensemble des dépôts ouverts. Neufs mouvements élémentaires sont proposés dans le paragraphe 4.4.2 : *TransCli1*, *PermCli1*, *DeuxOpt1*, *TransCli2*, *PermCli2*, *DeuxOpt2*, *OuvDep*, *FermDep* et *TransTour*.

Certains mouvements élémentaires comme fermer un dépôt i ont peu de chance d'améliorer une solution du *LRP*. En effet, la fermeture implique d'abord de transférer les tournées de i vers d'autre dépôts, et ces tournées doivent être ré-optimisées. Ce n'est qu'après évaluation de cette suite de mouvements cohérents, appelés *mouvement composite*, qu'on sait si le bilan est rentable ou non. Le paragraphe 4.4.3 décrit trois mouvements composites : ouvrir un dépôt fermé (OuvertureDepot), fermer un dépôt ouvert (FermetureDepot) et échanger les statuts ouvert/fermé de deux dépôts (EchangeDepots). Il est sous-entendu que des tournées sont transférées dans ces trois mouvements.

Enfin, dans le paragraphe 4.4.4, quatre recherches locales à base des différents mouvements sont proposées. La plus simple, $LSPT$, consiste à améliorer une tournée donnée. Elle est surtout utile pour ré-optimiser une tournée après transfert vers un nouveau dépôt. Les deux suivantes, $LS1$ et $LS2$, s'appliquent à l'ensemble des tournées, en effectuant des mouvements élémentaires dans une même tournée ou entre deux tournées. Elles ne diffèrent que par le choix du mouvement à chaque itération : première amélioration trouvée pour $LS1$, meilleure amélioration pour $LS2$. La quatrième et dernière recherche locale, LSG, est la plus complexe. Elle combine les mouvements composites *OuvertureDepot*, *FermetureDepot* et *EchangeDepots*.

4.4.2 Mouvements élémentaires

Dans ce qui suit, S est la solution courante, k et l sont des tournées, d et s sont des dépôts, et i et j sont des clients. Bien entendu, les mouvements ne sont effectués que s'ils conduisent à une solution réalisable et de moindre coût.

Les mouvements proposés ne modifiant pas l'ensemble des dépôts ouverts sont au nombre

de sept :

1. $TransCli1(S,k,i,j)$ - Transfert du client i après le nœud j (éventuellement le dépôt) dans d'une tournée k;
2. $PermCli1(S,k,i,j)$ - Échange de deux clients i et j dans d'une tournée k;
3. $DeuxOpt1(S,k,i,j)$ - Mouvement 2-opt classique du TSP, inversant la sous-séquence allant du client i au client j (inclus) dans d'une tournée k;
4. $TransCli2(S,k,i,l,j)$ - Transfert du client i de la tournée k après le nœud j de la tournée l;
5. $PermCli2(S,k,i,l,j)$ - Échange du client i de la tournée k avec le client j de la tournée l;
6. $DeuxOpt2(S,k,i,l,j)$ - Extension du 2-opt pour le cas multi-dépôt (voir figure 4.3). L'arête quittant i dans k et celle quittant j dans l sont supprimés. Dans le cas où k et l ont le même dépôt r (haut de la figure), il y a deux façons de reconnecter les morceaux. Si k et l ont des dépôts différents r et s, alors quatre reconnexions sont envisageables, car les nouvelles tournées peuvent être affectées à r ou à s. Le bas de la figure ne donne que deux des quatres possibilités;
7. $TransTour(S,d,k,s)$ - Transfert de la tournée k du dépôt r vers le dépôt s.

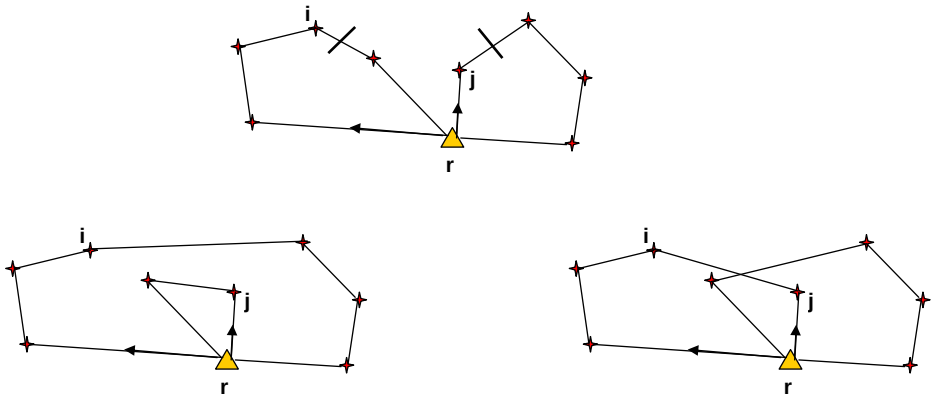
Concernant les mouvements intervenant sur le statut des dépôts, deux sont possibles :

1. $OuvDep(S,d)$ - Ouvre un dépôt d actuellement fermé;
2. $FermDep(S,d)$ - Ferme un dépôt d actuellement ouvert.

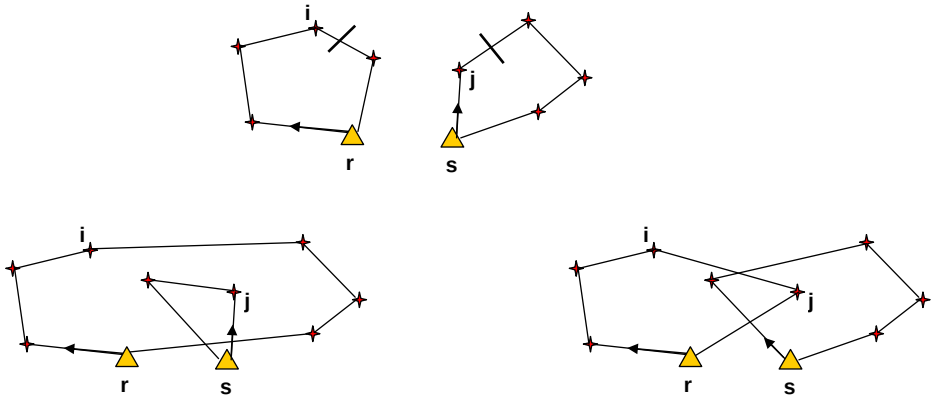
4.4.3 Mouvements composites

Trois mouvements composites sont développés, chacun lié au type de mouvement réalisé sur les dépôts.

1. $OuvertureDepot(S,d)$
   Cette première procédure évalue le gain obtenu si on ouvre le dépôt d de la solution S et qu'on tente de lui affecter les tournées d'autres dépôts. Si le gain est positif, la solution S est modifiée. Les mouvements élémentaires $OuvDep$ et $TransTour$ sont utilisés. Chaque tournée transférée est ré-optimisée avec la recherche locale $LSPT$ décrite au paragraphe 4.4.4.
   L'algorithme 3 précise la structure de *OuvertureDepot*. *ChargeDep* et *ChargeTour* sont des fonctions donnant la charge actuelle d'un dépôt ou d'une tournée. La fonction $Dep(k)$ renvoie le dépôt auquel est affecté une tournée k. La procédure travaille sur une copie S' de S. Elle commence par ouvrir le dépôt d (ligne 2) puis tente de lui affecter une par une les tournées d'autres dépôts. Un transfert de tournée s'effectue sur une copie S" de S', avec *TransTour* (ligne 6). La tournée transférée est ré-optimisée aussitôt par $LSPT$ (ligne 7). S' est mise à jour pour chaque transfert procurant un gain (ligne 9). Si le mouvement composite a un bilan positif, S est écrasée par S' (ligne 14).



a - Mouvements de type DeuxOpt2 entre deux tournées affectées au même dépôt en retirant les arêtes traversées en partant du noeud i et du noeud j



a - Mouvements de type DeuxOpt2 entre deux tournées affectées à deux dépôts différents en retirant les arêtes traversées en partant du noeud i et du noeud j
(afin de commencer et finir au même dépôt, les nouvelles tournées sont réaffectées à r ou s)

FIG. 4.3 – *Exemples pour la procédure DeuxOpt2*

Algorithme 3 Mouvement composite *OuvertureDepot*(S,d)

```
S' = S //travaille sur une copie S' de S
OuvDep(S',d)
pour chaque tournée k faire
   si (ChargeDep(d) + ChargeTour(k) ≤ W_d) alors
      S" = S
      TransTour(S",Dep(k),k,d)
      LSPT(S",k)
      si (Cout(S") < Cout(S')) alors
         S' = S"
      fin si
   fin si
fin pour
si (Cout(S') < Cout(S)) alors
   S = S'
fin si
```

2. *FermetureDepot*(S,d)
   FermetureDepot est le deuxième mouvement composite intervenant sur le statut des dépôts. Cette procédure évalue le gain si on ferme un dépôt d d'une solution S et qu'on transfert ses tournées vers les autres dépôts ouverts. Elle modifie réellement S si ce gain est positif. Elle appelle les mouvements élémentaires *FermDep* et *TransTour*, et ré-optimise les tournées transférées avec la recherche locale $LSPT$. Nous ne donnons pas l'algorithme détaillé qui se rapproche de celui de *OuvertureDepot*.

3. *EchangeDepots*(S,d,s)
   Enfin, le dernier mouvement composite évalue le gain si on ferme le dépôt d d'une solution S et qu'on ouvre le dépôt s. Les tournées de d sont alors transférées vers s. Elle modifie réellement S si l'ensemble de ces opérations est rentable. Les mouvements élémentaires *OuvDep*, *FermDep* et *TransTour* sont utilisés. Comme dans les deux autres mouvements composites, toute tournée transférée est ré-optimisée avec $LSPT$. Là encore, l'algorithme détaillé se déduit facilement de celui de *OuvertureDepot*.

4.4.4 Recherches locales retenues

Nous proposons quatre types de recherches locales. La plus simple, $LSPT$, consiste à améliorer une tournée donnée. Les deux suivantes, $LS1$ et $LS2$, s'appliquent à l'ensemble des tournées. La dernière, LSG, combine les mouvements composites.

1. $LSPT(S,k)$
   La première recherche locale optimise une tournée k d'une solution S en appliquant les trois premiers mouvements du paragraphe 4.4.2 : *TransCli1*, *PermCli1* et *DeuxOpt1*. Pour chaque paire de nœud, elle teste ces trois mouvements dans l'ordre donné et réalise le premier améliorant (*first improvement*). Elle sert principalement à ré-optimiser les tournées transférées dans les mouvements composites.

2. $LS1(S)$ et $LS2(S)$
   Ce sont des recherches locales de type *MDVRP*, déplaçant des clients dans une tournée ou bien dans deux tournées, éventuellement de dépôts différents. Elles utilisent donc les six premiers mouvements du 4.4.2 : *TransCli1*, *PermCli1*, *DeuxOpt1*, *TransCli2*, *PermCli2* et *DeuxOpt2*. $LS1$ fonctionne en mode *première amélioration*. $LS2$ est identique sauf qu'elle fonctionne en mode *meilleure amélioration*. Ces deux recherches locales ne touchent pas aux dépôts sauf dans un cas évident : quand un dépôt i se retrouve sans client. Il est alors fermé pour récupérer son coût d'ouverture O_i et n'est plus rouvert par ces recherches locales.

3. $LSG(S)$
   Cette dernière recherche locale est plus générale et utilise les mouvements composites. Son principe est donné dans l'algorithme 4. Elle commence par améliorer toutes les tournées avec $LSPT$. Puis, à chaque itération, elle essaie tous les mouvements possibles du type *EchangeDepots*, *OuvertureDepot* et *FermetureDepot*. Elle stoppe quand aucun d'entre-eux n'améliore la solution. Notons que si un seul mouvement permet une amélioration, par exemple *FermetureDepot*, il faut refaire une itération car l'exécution de ce mouvement peut faire apparaître de nouveaux mouvements des autres types.

Algorithme 4 Recherche locale générale $LSG(S)$

Améliorer chaque tournée k de S avec LSPT(S,k)
répéter
  $CoutInit = Cout(S)$ //mémorisation du coût en début de chaque itération
  pour tout *dépôt i ouvert* **faire**
    pour tout *dépôt j fermé avec* $ChargeDep(j) + ChargeDep(i) \leq W_j$ **faire**
      EchangeDepot(S,i,j)
    fin pour
  fin pour
  pour tout *dépôt i fermé* **faire**
    OuvertureDepot(S,i)
  fin pour
  pour tout *dépôt i ouvert* **faire**
    FermetureDepot(S,i)
  fin pour
jusqu'à $(CoutInit = Cout(S))$

4.5 Évaluation numérique

4.5.1 Implémentation et instances

Les méthodes de résolutions heuristiques proposées dans ce chapitre sont testées sur deux jeux d'instances présentés en Annexe : un jeu (*I1*) avec des capacités limitées sur les véhicules et les dépôts, et l'autre (*I2*) avec capacité limitée uniquement sur les véhicules. Les résultats sont présentés séparément, en commençant par ceux obtenus par les heuristiques constructives et de fusion ($AB1$, $AB3$, $AB3$ et $ECWA$). Ensuite, les performances obtenues en ajoutant

les recherches locales sur le routage ($LSPT$, $LS1$ et $LS2$) sont exposées, suivies de celles atteintes avec la recherche locale intégrant les dépôts (LSG). Les algorithmes sont codés en C et exécutés sur un PC Dell Optiplex GX260, avec un Pentium 4 à 2.4 GHz, 512 Mo de RAM et Windows XP.

4.5.2 Heuristiques constructives et de fusion

Rappelons que nous avons conçu quatre heuristiques : $AB1$, $AB3$, $AB3$ et $ECWA$. Les trois premières sont des méthodes constructives. $AB1$ et $AB2$ ouvrent des dépôts un par un, $AB1$ avec un critère de score et $AB2$ aléatoirement. Ces deux méthodes donnent de mauvais résultats pour les dépôts sans capacités (instances $I2$) car elles n'ouvrent alors qu'un dépôt. Dans ce cas, elles doivent être remplacées par $AB3$ qui ouvre aléatoirement un petit sous-ensemble de dépôts avant de construire les tournées. La dernière heuristique, $ECWA$ procède par fusion et peut s'appliquer aux deux types d'instances.

Nous utilisons le fait que les heuristiques randomisées $AB2$ et $AB3$ renvoient un résultat différent à chaque appel pour les exécuter 10 fois et augmenter les chances de bons résultats. Par conséquent, nous avons comparé sur $I1$ les heuristiques $AB1$, $AB2$, $AB2 \times 10$ et $ECWA$. Celles testées sur $I2$ sont $AB3$, $AB3 \times 10$ et $ECWA$.

Les tableaux 4.1 et 4.2 montrent les résultats obtenus par les différentes heuristiques présentées dans ce chapitre, respectivement sur les instances *I1* et *I2*. Les valeurs données sont des moyennes par taille d'instance. Les colonnes intitulées E_M (*Ecart avec la Meilleure méthode*) donnent l'écart relatif en pourcentage d'une heuristique par rapport à la meilleure heuristique en moyenne, positionnée la plus à gauche dans les tableaux. Chacune des approches fournit des résultats en moins d'une seconde, sauf pour $ECWA$ qui met 1 à 8 secondes pour résoudre les instances à partir de 100 clients.

L'élément le plus remarquable au vu de ces résultats est l'importance du choix des dépôts, leur localisation mais aussi le nombre à ouvrir. En effet, les meilleures performances sont obtenues par *AB1* et *AB2* exécutées 10 fois ($AB2 \times 10$) dans le cas avec capacité sur les dépôts. L'explication pour *AB1* vient du calcul de score pour déterminer les dépôts à ouvrir. Pour $AB2 \times 10$,10v d(v)0.5tes avCh.9355 TD[esp al li96ppix dép8.93esECWA l71.11J -272.6 -13es c2ivie'ue

		$AB2 \times 10$	$AB1$		$AB2$		$ECWA$	
n	m	Coût	Coût	E_M	Coût	E_M	Coût	E_M
20	5	54694	55759	2.0	63973	17.4	50936	-7.1
50	5	92361	96542	4.5	99883	8.8	84817	-7.6
100	5	221910	228824	3.2	295847	35.0	267220	22.5
100	10	287690	297354	3.8	327083	14.1	457746	58.4
200	10	482683	556318	15.2	575914	20.0	881795	82.2
Moyenne				**5.9**		**18.5**		**29.7**

NB : un écart négatif indique une amélioration par rapport à la solution prise en référence.

TAB. 4.1 – *Résultats des heuristiques - Instances I1 avec capacité sur les dépôts*

		$ECWA$	$AB3 \times 10$		$AB3$	
n	m	Coût	Coût	E_M	Coût	E_M
100	10	1388.3	1714.8	25.2	2088.1	53.8
100	20	1407.0	1856.4	36.5	2304.0	72.1
150	10	1804.5	2383.9	33.9	2546.9	42.0
150	20	1826.4	2163.1	22.9	2494.3	40.5
200	10	2284.1	2964.0	30.8	3346.4	47.2
200	20	2246.7	2983.1	37.7	3313.7	51.0
Moyenne				**31.2**		**51.1**

NB : un écart négatif indique une amélioration.

TAB. 4.2 – *Résultats des heuristiques - Instances I2 sans capacité sur les dépôts*

indicatif, les coûts relatifs à l'ouverture des dépôts et ceux relatifs au routage atteignent 272627 et 78493 respectivement pour $ECWA$, contre 142096 et 107582 pour $AB1$. L'importance des W_i pour les instances de $I1$ est ici mis en avant. La structure des coûts des instances $I2$ semble bien différente car cette fois-ci, concernant les dépôts et les tournées, $ECWA$ obtient des coûts moyens de 594 et 1231 respectivement contre 200 et 2482 avec $AB1$. Le rapport du coût des dépôts sur le coût du routage est alors bien moins fort.

Ces premiers résultats mettent l'accent sur l'importance d'avoir une bonne sélection du sous-ensemble de dépôts à ouvrir. Il faut trouver un bon compromis entre les coûts d'ouverture des sites et le routage en découlant.

4.5.3 Recherches locales sur les tournées

Les tableaux 4.3 - 4.8 montrent les résultats obtenus par les différentes heuristiques suivies des recherches locales sur le routage ($LSPT$, $LS1$ et $LS2$), sur les instances *I1* et *I2*. Les valeurs indiquées sont encore une fois des moyennes par taille d'instances. Les colonnes intitulées E_H et E_M représentent respectivement les écarts relatifs en pourcentage d'une méthode par rapport à l'heuristique correspondante sans recherche locale (*Ecart avec l'Heuristique*) et par rapport à la méthode avec la recherche locale apportant les meilleurs résultats en moyenne (*Ecart avec la Meilleure méthode*) et positionnée le plus à gauche dans les tableaux. Chacune des heuristiques fournit des solutions en moins d'une seconde, sauf pour $ECWA$, $AB2 \times 10$ et $AB3 \times 10$ suivies de *LS1* ou *LS2*, sur les plus grosses instances. En effet, à partir de 100

		$AB2 \times 10 + LSPT$		$AB1 + LSPT$			$ECWA + LSPT$		
n	m	Coût	E_H	Coût	E_H	E_M	Coût	E_H	E_M
20	5	54214	-0.9	54596	-2.3	0.5	50740	-0.5	-6.7
50	5	90773	-1.9	95460	-1.2	5.3	84314	-0.7	-6.3
100	5	220112	-0.8	226572	-1.0	3.1	266218	-0.4	23.0
100	10	285633	-0.8	295254	-0.7	3.9	457099	-0.2	59.4
200	10	479556	-0.7	552735	-0.7	15.2	880704	-0.1	83.1
Moyenne			**-1.1**		**-1.1**	**5.9**		**-0.4**	**30.5**

NB : un écart négatif indique une amélioration par rapport à la solution prise en référence.

TAB. 4.3 – *Résultats de LSPT - Instances I1 avec capacité sur les dépôts*

		$ECWA + LSPT$		$AB3 \times 10 + LSPT$			$AB3 + LSPT$		
n	m	Coût	E_H	Coût	E_H	E_M	Coût	E_H	E_M
100	10	1369.0	-1.2	1646.9	-3.9	21.8	2020.8	-3.2	50.7
100	20	1370.9	-2.3	1786.3	-3.5	34.7	2230.5	-3.1	70.5
150	10	1777.4	-1.5	2297.2	-3.6	31.1	2454.9	3.0	39.1
150	20	1777.0	-2.6	2089.3	-3.3	21.9	2402.1	-1.5	39.2
200	10	2260.1	-1.1	2875.3	-2.9	28.4	3253.0	-2.7	44.7
200	20	2206.4	-1.7	2884.3	-3.3	35.5	3208.5	-3.1	48.9
Moyenne			**-1.7**		**-3.4**	**28.9**		**-3.2**	**48.9**

NB : un écart négatif indique une amélioration par rapport à la solution prise en référence.

TAB. 4.4 – *Résultats de LSPT - Instances I2 sans capacité sur les dépôts*

clients, ces heuristiques mettent alors entre 1 et 9 secondes pour résoudre le problème, les temps les plus longs étant obtenu avec *LS2*. Les comparaisons avec *AB2* exécutées une seule fois ne sont plus mentionnées, puisque l'ouverture des dépôts étant aléatoire, les résultats obtenus avec $AB2 \times 10$ sont plus attrayants.

Ces recherches locales n'intervenant que sur les tournées, le même type de conclusions que dans le paragraphe 4.5.2 reste valide. Le choix des dépôts est un élément-clé de l'efficacité.

Le premier point vraiment notable est la performance des recherches locales *LS1* et *LS2*. En effet, elles améliorent les solutions des heuristiques *AB* de manière importante (entre 8.8% et 26.8% en moyenne). *LSPT*, bien que déjà performante, ne réduit les coûts que de 1.1% à

		$AB2 \times 10 + LS1$		$AB1 + LS1$			$ECWA + LS1$		
n	m	Coût	E_H	Coût	E_H	E_M	Coût	E_H	E_M
20	5	48595	-11.7	48032	-14.3	-1.1	49676	-2.4	3.0
50	5	79307	-14.2	83339	-13.4	5.6	82772	-2.5	5.3
100	5	210193	-5.4	212225	-7.1	1.3	264438	-1.1	28.0
100	10	269374	-6.3	279317	-6.1	4.0	455863	-0.5	68.5
200	10	456762	-5.4	532230	-4.4	16.5	879347	-0.3	91.8
Moyenne			**-8.8**		**-9.0**	**5.7**		**-1.4**	**39.4**

NB : un écart négatif indique une amélioration par rapport à la solution prise en référence.

TAB. 4.5 – *Résultats de LS1 - Instances I1 avec capacité sur les dépôts*

		$ECWA + LS1$		$AB3 \times 10 + LS1$			$AB3 + LS1$		
n	m	Coût	E_H	Coût	E_H	E_M	Coût	E_H	E_M
100	10	1343.1	-2.8	1313.1	-23.4	-1.2	1590.1	-23.6	20.9
100	20	1346.3	-3.8	1409.1	-24.1	7.3	1703.5	-25.7	32.9
150	10	1760.3	-2.4	1778.6	-25.4	2.4	1910.2	-19.6	9.0
150	20	1751.1	-4.0	1672.8	-22.7	-1.1	1880.3	-22.8	10.7
200	10	2239.1	-2.0	2232.6	-24.7	0.4	2501.4	-25.3	12.0
200	20	2169.5	-3.2	2193.5	-26.7	3.8	2432.5	-26.7	14.2
Moyenne			**-3.1**		**-24.5**	**1.9**		**-25.2**	**16.6**

NB : un écart négatif indique une amélioration par rapport à la solution prise en référence.

TAB. 4.6 – *Résultats de LS1 - Instances I2 sans capacité sur les dépôts*

		$AB2 \times 10 + LS2$		$AB1 + LS2$			$ECWA + LS2$		
n	m	Coût	E_H	Coût	E_H	E_M	Coût	E_H	E_M
20	5	48562	-11.7	45647	-18.3	-5.6	49676	-2.4	3.0
50	5	77667	-15.9	81137	-15.7	4.6	82728	-2.6	7.0
100	5	208425	-6.3	211539	-7.4	1.9	264397	-1.2	29.2
100	10	264678	-8.1	280020	-5.8	6.3	455937	-0.5	71.6
200	10	455668	-5.6	529270	-4.9	16.1	879437	-0.3	92.2
Moyenne			**-9.8**		**-10.2**	**5.3**		**-1.4**	**40.9**

NB : un écart négatif indique une amélioration par rapport à la solution prise en référence.

TAB. 4.7 – *Résultats de LS2 - Instances I1 avec capacité sur les dépôts*

		$AB3 \times 10 + LS2$		$ECWA + LS2$			$AB3 + LS2$		
n	m	Coût	E_H	Coût	E_H	E_M	Coût	E_H	E_M
100	10	1268.4	-26.2	1339.6	-3.1	5.1	1549.2	-25.7	23.9
100	20	1370.0	-26.0	1350.2	-3.5	-2.8	1675.6	-26.9	24.3
150	10	1735.5	-27.2	1757.3	-2.6	0.9	1851.9	-27.2	6.0
150	20	1626.7	-24.8	1754.7	-3.9	5.9	1843.9	-25.9	13.4
200	10	2181.5	-26.4	2230.0	-2.3	1.8	2441.7	-27.0	11.9
200	20	2166.0	-27.3	2169.0	-3.3	-1.4	2387.8	-27.9	9.5
Moyenne			**-26.3**		**-3.1**	**1.6**		**-26.8**	**14.8**

NB : un écart négatif indique une amélioration par rapport à la solution prise en référence.

TAB. 4.8 – *Résultats de LS2 - Instances I2 sans capacité sur les dépôts*

		(1) $AB1 + LSPT$	(2) $AB1 + LSG$		(3) $AB1 + LS1$	(4) $AB1 + LSG + LS1$		
n	m	Coût	Coût	E_1	Coût	Coût	E_3	E_2
20	5	54596	53941	-1.1	48032	47788	-0.3	-11.7
50	5	95460	89740	-5.3	83339	80150	-3.5	-10.6
100	5	226572	223098	-1.2	212225	211154	-0.4	-5.3
100	10	295254	279996	-5.5	279317	265345	-5.4	-5.3
200	10	552735	501692	-9.0	532230	475815	-10.4	-5.2
Moyenne				**-4.7**			**-4.2**	**-7.5**

NB : un écart négatif indique une amélioration par rapport à la solution prise en référence.

TAB. 4.9 – *Recherche locale sur routage et dépôts - Instances I1 avec capacité sur les dépôts*

3.4% en moyenne en partant des mêmes solutions de départ. Ceci vient du fait que le voisinage qu'elle explore est beaucoup plus restreint. Elle se limite à des mouvements à l'intérieur d'une même tournée alors que les recherches de type *LS* explorent un voisinage plus large, mais au prix de temps de calcul plus importants.

On remarque que *ECWA* profite peu des recherches locales sur les tournées : l'amélioration moyenne n'est que de 2% alors qu'elle atteint 36% pour les autres heuristiques. Ceci indique que les tournées construites par *ECWA* sont excellentes, même si cette heuristique tend à utiliser trop de dépôts sur les instances $I1$.

Enfin, les résultats avec *LS2* sont un peu meilleurs que ceux obtenus avec *LS1*, mais la différence est de moins de 1%, et même négligeable lorsque la solution de départ est fournie par *ECWA*. Cependant, *LS2* réalisant à chaque itération le meilleur mouvement possible (contre le premier mouvement améliorant dans *LS1*), ses temps de calcul sont un peu plus longs.

4.5.4 Recherche locale sur les tournées et la localisation

Nous donnons enfin les performances de la recherche locale avec mouvements composites (LSG). Dans le tableau 4.9, en moyenne pour les différentes tailles d'instances de type $I1$, sont comparés les combinaisons (1) $AB1 + LSPT$, (2) $AB1 + LSG$, (3) $AB1 + LS1$ et (4) $AB1 + LSG + LS1$. Dans la quatrième combinaison, $LS1$ est utilisée en post-optimisation et apporte un gain supplémentaire. Les colonnes E_i donnent l'écart d'une méthode par rapport à la combinaison $i \in \{1,2,3\}$. Le tableau 4.10 concerne les instances $I2$, et les combinaisons d'algorithmes sont identiques sauf qu'$AB1$ est remplacée par $AB3$.

Les résultats montrent encore une fois l'importance de la localisation des dépôts. La recherche locale intervenant sur ces deux aspects (LSG) permet un gain moyen d'environs 4.5% par rapport à la version ne l'incluant pas sur les instances avec capacité sur les dépôts. Ce gain passe à plus de 12% sur les instances sans capacités, pour lesquelles l'heuristique constructive $AB3$ avait montré précédemment des difficultés par rapport à *ECWA*. Malgré tout, l'utilisation de *LS1* en fin d'algorithme permet un gain moyen supplémentaire de 7.5% pour les instances *I1* et de 22.8% sur celles de *I2*.

		(1)$AB3 + LSPT$	(2)$AB3 + LSG$		(3)$AB3 + LS1$	(4)$AB3 + LSG + LS1$		
n	m	Coût	Coût	E_1	Coût	Coût	E_3	E_2
100	10	2020.8	1697.4	-14.1	1590.1	1344.5	-14.5	-20.6
100	20	2230.5	1657.9	-26.1	1703.5	1269.5	-25.8	-22.8
150	10	2454.9	2377.0	-1.9	1910.2	1830.2	-2.9	-23.0
150	20	2402.1	2137.1	-10.6	1880.3	1665.2	-11.2	-22.2
200	10	3253.0	2843.1	-12.0	2501.4	2176.5	-12.5	-23.7
200	20	3208.5	2957.0	-6.8	2432.5	2238.5	-6.7	-24.4
Moyenne				**-11.9**			**-12.3**	**-22.8**

NB : un écart négatif indique une amélioration par rapport à la solution prise en référence.

TAB. 4.10 – *Recherche locale sur routage et dépôts - Instances I2 sans capacité sur les dépôts*

4.6 Conclusion

Dans ce chapitre, nous avons proposé des heuristiques constructives et de fusion, ainsi que des recherches locales pour le problème de localisation-routage, pour le cas où les dépôts et les véhicules ont des capacités limitées.

Les heuristiques proposées pour le *LRP* sont des adaptations non triviales d'algorithmes connus en tournées de véhicules. Les résultats obtenus par les différentes versions permettent de discerner la difficulté du problème. Pour obtenir de bons résultats, le choix des dépôts à ouvrir est crucial. Pour cela, il est important de gérer les aspects localisation et construction des tournées le plus globalement possible, comme le montre les résultats apportés par les approches utilisant des techniques intelligentes d'élaboration de solution en opposition avec des choix aléatoires. En particulier, le très efficace algorithme de Clarke et Wright généralisé (*ECWA*), qui fait évoluer la solution en changeant l'affectation des tournées (et par ce biais la localisation) lors de la construction de celles-ci, montre l'intérêt de ne pas dissocier les niveaux de décision.

Un grand nombre de mouvements permettant d'explorer l'espace des solutions a également été développé. Ceux-ci ont permis de construire différentes recherches locales, plus ou moins complexes. Un premier type intervient uniquement sur les tournées. Elles permettent des améliorations mais ne compensent pas suffisamment un mauvais choix de dépôts initial. Cependant, une bonne recherche locale sur les tournées est nécessaire car les gains apportés peuvent être très importants (de moins de 5% avec une recherche locale optimisant chaque tournée séparément à plus de 33% avec une version plus performante). C'est pourquoi, au vu des résultats, la version appelée *LS1* sera majoritairement réutilisée par la suite. Ces résultats sont sensiblement semblables à ceux de *LS2*, pour des temps de calcul légèrement inférieurs.

Le second type de recherches locales implique la localisation. Ces recherches sont composées de chaînes complexes de mouvements. Une version en intégrant trois (*LSG*) a été testée et confirme le fait que le positionnement des dépôts est déterminant dans la résolution du problème.

Les approches proposées permettent de résoudre le *LRP* généralisé et ceci en quelques secondes maximum, ce qui est peu pour ce type de problème, particulièrement complexe. Ils peuvent résoudre également des instances de taille beaucoup plus importante que dans

Albareda-Sambola *et al.* (2005), avec des contraintes de capacité sur les véhicules.

Les observations faites sur les solutions obtenues par les méthodes de résolution heuristiques proposées dévoilent les limites de ce type d'approches. Les algorithmes développés sont déjà sophistiqués mais la grande variabilité des résultats laisse espérer une marge de progression si on se tourne vers des métaheuristiques.

Le principe des algorithmes constructifs présentés dans ce chapitre a fait l'objet d'une présentation en conférence internationale francophone, *Conférence Francophone de Modélisation et Simulation* - MOSIM'04 (Prins *et al.*, 2004a). Les heuristiques de fusions ont été proposées pour la première fois lors d'un congrès international, *International Conference on Industrial Engineering and Systems Management* - IESM'05 (Prins *et al.*, 2005c) puis dans un article de revue internationale, *4OR - A Quarterly Journal of Operations Research* (Prins *et al.*, 2006b), en utilisation au sein d'une métaheuristique présentée au chapitre suivant.

Chapitre 5

GRASP

5.1 Introduction

Comme nous venons de le voir dans les chapitres précédents, le *LRP* est un problème très complexe car il combine deux niveaux de décision. Le recours à des algorithmes avec construction heuristique d'une solution de départ améliorée par des recherches locales peut apporter des résultats rapides. Cependant, même en appliquant des techniques sophistiquées comme proposé au chapitre 4, la grande variabilité des coûts obtenus laisse présager une possibilité de progression dans la qualité des solutions, en développant des approches plus puissantes. C'est la motivation qui est à la base des recherches exposées dans la suite.

Les métaheuristiques sont en général plus puissantes que des heuristiques constructives suivies d'une recherche locale car elles incluent des mécanismes permettant de sortir d'un minimum local. Il en existe deux grandes familles, comme décrit au chapitre 2 : celles fondées sur l'exploration d'un voisinage (*GRASP*, *SA* ou encore *TS*), et celles basées sur l'utilisation d'une population qui font évoluer en parallèle un ensemble de solutions (*GA*, *SS* ou *AC*).

Dans ce chapitre, le *LRP* est résolu par une approche métaheuristique de type *GRASP* : *Greedy Randomized Adaptive Procedure*. Cette méthode a été choisie car elle a déjà apporté des bons résultats sur divers problèmes d'optimisation et n'a jamais été appliquée au *LRP*. Afin de le rendre plus efficace, l'algorithme de base a été ici agrémenté d'une technique d'apprentissage et d'une post-optimisation par path relinking (*PR*).

La section 5.2 présente plus en détail le principe général du *GRASP* et de ses versions améliorées. La transposition pour le *LRP* est donnée dans la section 5.3. Les différents composants additionnels ajoutés pour rendre la méthode plus performante sont détaillés dans les sections 5.4 et 5.5. L'algorithme général est exposé en section 5.6. Les sections 5.7 et 5.8 sont dédiées aux résultats obtenus et à l'analyse des contributions apportées par les composants additionnels de la méthode. Enfin, une conclusion termine le chapitre en section 5.9.

5.2 Présentation générale du *GRASP*

5.2.1 Version de base

Le *GRASP* (*Greedy Randomized Adaptive Procedure*) crée une nouvelle solution à chaque itération. Il s'apparente à un échantillonnage de l'espace des solutions. Il peut fournir de bons résultats pour divers problèmes combinatoires. Des techniques plus élaborées peuvent être introduites pour améliorer les performances de la version de base comme l'indique l'état de l'art effectué au chapitre 2. Les solutions produites sont indépendantes et la meilleure est restituée à la fin. Une itération est composée de deux phases :

- la construction d'une solution par une heuristique gloutonne randomisée;
- l'amélioration de celle-ci par une recherche locale.

En effet, une heuristique déterministe donnerait toujours le même résultat à chaque itération. Il est donc nécessaire d'introduire des aléas afin de diversifier la recherche. Le but n'est pas de fournir des solutions aléatoires à la recherche locale, car un tel principe engendrerait en général des coûts assez élevés, mais de calculer des solutions à la fois variées et de bonne qualité. La randomisation est basée sur une liste restreinte (*Restricted Candidate List - RCL*) de choix possibles à chaque étape de la construction de la solution.

L'ensemble de ces éléments donne son nom à la méthode. En effet, un algorithme glouton (*greedy search*) est utilisé avec l'introduction d'aléas (*randomized*) afin de produire des solutions différentes à chaque itération. Ces aléas sont représentés par une liste de choix possibles (*RCL*) qui change à chaque étape de la construction d'une solution (*adaptive*). Deux politiques sont envisageables. La première est de remplir la liste des décisions possibles, sélectionnées au hasard, puis de choisir la meilleure selon le critère glouton. La seconde est d'y placer les meilleures décisions trouvées par l'algorithme glouton et d'en choisir une au hasard. Ici, la deuxième approche est employée.

La conception d'un GRASP repose sur des principes simples. Le vrai point délicat est de randomiser l'algorithme glouton en préservant la qualité des solutions. En particulier, il faut déterminer la taille de la *RCL*. Cette dernière peut être fixée par une cardinalité α ou selon la qualité de ses éléments. Dans le premier cas, le remplissage de la liste se fait simplement en y introduisant α élément-décisions. Dans le second, une décision possible est insérée seulement si le coût qui lui est associé est inférieur à un certain seuil déterminé comme suit. Soit e un élément et $c(e)$ son coût associé. Si c_{min} et c_{max} sont respectivement le plus petit et le plus grand coût possible ajouté, alors e est inséré si et seulement si $c(e) \leq c_{min} + \beta(c_{max} - c_{min})$ avec $\beta \in [0,1]$. Quand $\beta = 1$, c'est une construction aléatoire qui opère, et quand $\beta = 0$, c'est l'algorithme glouton pur qui est utilisé. Il est difficile de déterminer la valeur de β et elle est souvent choisie aléatoirement.

5.2.2 Versions améliorées

Afin d'améliorer les performances de la version de base, diverses techniques peuvent être employées, comme celles basées sur la mémorisation de certaines informations ou l'insertion d'un path relinking. Quelques unes des propositions pouvant être trouvées dans la littérature

sont énoncées dans ce qui suit.

Tout d'abord, il est possible d'intervenir lors de la phase de construction. Par exemple, Fleurent et Glover (1999) ont utilisé une mémorisation des meilleures solutions dites *élites* sélectionnées pendant les différentes itérations du *GRASP*. Puis, des statistiques sont réalisées sur les variables de décision les plus *déterminantes*, c'est-à-dire celles qui ne peuvent être modifiées sans affecter significativement le coût de la fonction-objectif ainsi que le statut d'autres variables. La valeur de ces variables déterminantes est alors biaisée dans la phase de construction vers celles prises dans par la majorité des solution-élites. Pour cela, le coût $c(e)$ d'ajout d'un élément dans la solution peut être ajusté avec une fonction $K(e)$, par exemple $K(e) = \lambda \cdot c(e) + I(e)$, où $I(e)$ est une fonction d'intensification. La constitution de la *RCL* est ainsi biaisée. La valeur de λ peut varier durant le *GRASP*, permettant une diversification quand λ est grand et une intensification de la recherche quand λ est petit.

Toujours en rapport avec la phase de construction de la solution, Prais et Ribeiro (2000) ont proposé de modifier la valeur de α (cardinal de la *RCL*) par un processus d'apprentissage. En fait, α est choisi parmi une série de valeurs possibles, chacune associée à une probabilité qui est mise à jour à chaque itération, afin de favoriser celles procurant les meilleurs résultats. Ils ont nommé leur méthode *Reactive GRASP*.

Une autre possibilité est d'intervenir sur la deuxième phase de la méthode, en restreignant l'appel à la recherche locale pour de ne l'appliquer qu'à des solutions prometteuses. Par exemple, il est possible de conserver les solutions construites lors de chaque itération. Si l'algorithme glouton en fournit une ayant déjà été visitée, il n'est pas nécessaire d'avoir recours à la recherche locale, la solution qui en découlera ne pourra qu'être de coût égal ou supérieur à la meilleure déjà trouvée. Une autre technique est de ne faire une recherche locale que sur les solutions de bonne qualité, par exemple celles ayant un coût inférieur ou égal au coût moyen des solutions déjà construites. Ceci peut éviter de longues explorations qui aboutiront généralement à des résultats de qualité moyenne (Feo *et al.*, 1994).

D'autres techniques d'amélioration de la version de base du *GRASP* peuvent être citées, comme l'application d'une recherche locale pendant la phase de construction (*Proximate Optimality Principle* - *POP*, de Binato *et al.* (2002)), ou en perturbant les coûts, cette dernière étant surtout intéressante lorsque l'algorithme de construction n'est pas très sensible à la randomisation. Plus de détails sur les différentes techniques d'amélioration du *GRASP* sont donnés dans les articles de synthèse de Resende et Ribeiro (2003, 2005).

5.2.3 Versions hybrides

Des versions hybrides du *GRASP* peuvent également être implémentées, comme par exemple la combinaison avec l'utilisation d'un path relinking (*PR*). Le *PR* est une méta-heuristique qui a été proposée pour la première fois par Glover pour intensifier un *TS* ou un *SS* par l'exploration de trajectoires reliant des *solution-élites* (Glover et Laguna, 1997; Glover *et al.*, 2000). La trajectoire d'une solution U vers une *solution-guide* V se parcourt en introduisant dans U des attributs présents dans V. Plusieurs alternatives sont possibles pour un couple (U,V), par exemple :

- *PR* unidirectionnel : réalisé uniquement de la solution U à la solution V;

- *PR* bidirectionnel : réalisé de U vers V puis de V vers U;
- *PR* randomisé : le choix de l'attribut introduit dans U se fait aléatoirement parmi ceux permettant de se rapprocher de la solution guide V;
- *PR* tronqué : la transformation de U en V est seulement partielle et la trajectoire n'est donc qu'en partie explorée.

Un *PR* est rarement utilisé seul. Il sert généralement de post-optimisation à une autre métaheuristique. Dans le cadre du *GRASP*, il fut introduit par Laguna et Marti (1995), et les utilisations sont soit en post-optimisation, soit en intensification pendant les itérations du *GRASP* (Resende et Ribeiro, 2003).

En intensification (Laguna et Marti, 1995), un groupe de solutions-élites est constitué au cours des itérations en y introduisant des solutions sous les conditions suivantes : elles doivent à la fois être de bonne qualité, et suffisamment différentes les unes par rapport aux autres. Une mesure de distance est donc nécessaire. Le *PR* est alors réalisé à chaque itération entre la solution courante U et une solution-élite V choisie aléatoirement dans le groupe. Cependant, il est possible de biaiser ce choix en faveur des plus distantes de U, ce qui donne habituellement de meilleurs résultats (Resende et Werneck, 2004; Piñana *et al.*, 2004).

En tant que post-optimisation (Piñana *et al.*, 2004), un groupe de solution-*élites* est également constitué. Le principe est d'appliquer le *PR* en fin de *GRASP* entre les solution-élites uniquement. Ce groupe est actualisé en y remplaçant les moins bonnes solutions par les nouvelles obtenues par le *PR* si elles répondent au critère de qualité requis.

5.3 Composants basiques du *GRASP* pour le *LRP*

5.3.1 Construction d'une solution

Une solution de base est construite à chaque itération du *GRASP* à l'aide d'une heuristique gloutonne. Celle employée ici est l'heuristique de Clarke et Wright généralisée (*ECWA*) présentée en section 4.3 du chapitre précédent, car elle permet de construire une solution en tenant compte à chaque étape des aspects localisation et élaboration de tournées.

Cependant, afin d'obtenir des solutions différentes à chaque itération du *GRASP*, l'algorithme glouton doit être randomisé. Pour cela, une liste de décisions possibles (*RCL*) est nécessaire. Dans le cadre de *ECWA*, les éléments de la liste sont des fusions possibles définies par :

- les deux tournées à fusionner;
- le sens de parcours de chacune d'elle;
- le dépôt d'affectation de la tournée obtenue;
- le gain réalisable.

La taille α de la *RCL* est tirée au sort dans [1,α_{max}] à chaque étape de la construction de la solution. La *RCL* est ensuite remplie des α meilleures fusions détectées par l'algorithme glouton. Ensuite, la sélection de la fusion à effectuer se fait aléatoirement. On note *RECWA* (*Random ECWA*) l'heuristique gloutonne obtenue.

Comme nous l'avons vu dans le chapitre 4, *ECWA* fonctionne bien pour affecter les clients aux dépôts et pour réaliser les tournées, comme ce serait le cas pour un *VRP* multi-dépôt. Cependant, pour le *LRP*, il a tendance à ouvrir un grand nombre de sites. En effet, lors de la création du *bouquet de marguerites*, les clients sont affectés au dépôt disponible le plus proche. Si ces derniers sont bien répartis au sein des clients, beaucoup risquent alors d'être ouverts. Ensuite, lors des fusions, *ECWA* réussit en principe à en fermer quelques uns, mais il est possible que cela ne soit pas suffisant, induisant un coût global bien plus élevé que la valeur optimale. Partant de cette observation, une idée pour obtenir de meilleurs résultats est de réduire la cardinalité du groupe de dépôts disponibles (*Sous-ensemble Disponible* - SD) dans l'étape élaborant le bouquet de marguerites, puis d'autoriser l'ouverture éventuelle de n'importe quel dépôt lors des fusions ($SD = I$).

Cette idée est exploitée dans le *GRASP*. A la première itération, SD contient tous les dépôts de I. Dans les suivantes, SD est constitué initialement de seulement deux dépôts lors de la construction des marguerites : un sélectionné itérativement dans I afin d'être sûr que chacun soit au moins ouvert une fois durant le *GRASP*, et un second choisi aléatoirement parmi les dépôts restants. Chaque client, considéré dans un ordre aléatoire, est rattaché ensuite au plus proche dépôt de SD de capacité compatible. Si aucun ne peut le servir par manque de capacité, alors le dépôt compatible le plus proche de ce client et non encore dans SD est ajouté à SD. Ce principe ajoute une randomisation supplémentaire, et permet une plus grande diversification et donc une meilleure exploration de l'espace des solutions.

5.3.2 Recherche locale

La deuxième phase d'un *GRASP* requiert une recherche locale après la construction d'une solution par l'algorithme glouton. Celle proposée ici est *LS1*, présentée au chapitre 4. Rappelons qu'elle effectue à chaque itération le premier mouvement améliorant rencontré, en testant les mouvements suivants *TransCli1*, *PermCli1*, *DeuxOpt1*, *TransCli2*, *PermCli2* et *DeuxOpt2* (voir paragraphe 4.4.2).

5.3.3 Algorithme de base du *GRASP*

Algorithme 5 *GRASP* de base pour le *LRP*

$BestCost = +\infty$ //initialiser le coût de la meilleure solution trouvée
//boucle principale
pour ($k = 1$ *to* *MaxIt*) **faire**
  RECWA(S) //construction d'une solution par l'algorithme glouton randomisé
  LS1(S) //amélioration par recherche locale
  //si le coût obtenu améliore BestCost
  si ($cout(S) < BestCost$) **alors**
    $BestSol = S$ //mettre à jour la meilleure solution
    $BestCost = cout(S)$ //et son coût
  fin si
fin pour
Renvoyer($BestSol$) //le résultat est la meilleure solution trouvée

L'Algorithme 5 montre le principe de base du *GRASP* utilisé pour le *LRP* : construction d'une solution et recherche locale. *MaxIt* est le nombre d'itération de la méthode.

5.4 Mécanisme d'apprentissage

Pour avoir une bonne exploration de l'espace des solutions avec une méthode de type *GRASP*, un grand nombre d'itérations est généralement nécessaire. Or, la complexité de *ECWA* ou *RECWA* ($O(mn^3)$) ne le permet pas en des temps raisonnables. Il est donc nécessaire de trouver une technique permettant d'obtenir de bons résultats tout en limitant ce nombre d'itérations.

En reprenant les observations qui ont pu être faite au chapitre 4, il apparaît qu'un des éléments-clés pour obtenir de bonnes solutions repose sur le choix de dépôts à ouvrir. *ECWA* seul peut avoir des difficultés à déterminer ce sous-ensemble mais la restriction des dépôts disponibles avec le sous-ensemble *SD* donne de meilleurs résultats. En effet, ceci permet une bonne diversification des solutions, mais engendre également la visite de solutions impliquant l'utilisation de dépôts peu prometteurs. Il semble donc intéressant de favoriser la recherche sur les sites les plus attrayants, en ayant recours par exemple à un processus d'apprentissage. C'est l'idée appliquée ici par le biais de phases alternées de diversification et d'intensification.

Lors d'une phase de diversification, l'algorithme fonctionne comme décrit précédemment mais mémorise les dépôts utilisés dans la meilleure solution de la phase. Dans la phase suivante d'intensification, l'algorithme n'introduit que ces dépôts dans *SD* qui, cette fois-ci, reste fixe durant toute la construction de la solution, c'est-à-dire aussi bien lors de la création du bouquet de marguerites que lors des fusions (étape durant laquelle, en diversification, $SD = I$). La capacité totale des dépôts de *SD* couvre alors la demande des clients puisqu'une solution réalisable les utilisant a déjà été trouvée. Cependant, le problème de l'existence d'une affectation possible de l'ensemble des clients aux dépôts de *SD* est équivalent à un problème *NP*-complet de *bin-packing*. Il est donc possible que *RECWA* ne réussisse pas à affecter tous les clients lors de la création du bouquet de marguerites. Dans un tel cas, le dépôt fermé le plus proche d'un client non affecté sera ouvert et ajouté à *SD*.

L'idée est d'alterner ces phases afin de proposer un sous-ensemble de dépôts différent à chaque intensification. Le *GRASP* va donc osciller entre :

- diversification : où les dépôts disponibles pour l'initialisation de la solution dans *RECWA* sont ceux contenus dans *SD* comme décrit dans au paragraphe 5.3.1. Cette phase dure *maxitdiv* itérations du *GRASP*;
- intensification : où les dépôts de *SD* pour toute la construction de la solution par *RECWA* (bouquet de marguerites et fusions) sont ceux ouverts dans la meilleure solution trouvée lors de la dernière diversification. Cette phase dure *maxitint* itérations du *GRASP*.

Ce processus d'apprentissage concernant le meilleur sous-ensemble de dépôts à ouvrir s'apparente à la mémorisation proposée dans Fleurent et Glover (1999) et rappelée brièvement au paragraphe 5.2, les variables de décisions concernant l'ouverture des dépôts pouvant représenter les choix déterminants.

5.5 Path Relinking

Un path relinking (*PR*) est ajouté en post-optimisation du *GRASP*. La structure proposée dans Piñana *et al.* (2004) est ici simplifiée afin de trouver un compromis entre la qualité des solutions et les temps de calcul.

Une étape préliminaire consiste à rassembler un ensemble de solutions sur lequel opérer le *PR*. Un groupe *BestSet* de *NBest* solutions est constitué à partir des meilleures solutions trouvées durant les diversifications (solutions supposées assez différentes les unes des autres, les dépôts utilisés étant différents), plus la meilleure de toutes les itérations du *GRASP* si celle-ci n'est pas obtenue pendant la diversification. Ensuite, comme un *PR* entre solutions les plus distantes donne habituellement de meilleurs résultats (Resende et Werneck, 2004), un sous-ensemble *DistSet* de *NMaxDist* solutions en est extrait. La meilleure solution (*BestSol*) du *GRASP* y est d'abord introduite. Ensuite, on enlève de *BestSet* la solution la plus distante de celles de *DistSet* pour l'ajouter à ce dernier. Cette opération est répétée *NMaxDist*-1 fois. *DistSet* est en fait le groupe de solutions-élites de Piñana *et al.* (2004). Le pseudo-code concernant la création de l'ensemble de solutions-élites *DistSet* est donné dans l'algorithme 6.

Ensuite, le *PR* est effectivement réalisé et se fait par l'exploration de trajectoires entre solutions. Il s'opère entre chaque paire (U,V) de solutions de *DistSet* en mode bidirectionnel (exploration des chemins allant de U à V et de V à U). La simplification annoncée vient du fait que le groupe *DistSet* n'est pas actualisé durant le *PR*.

5.5.1 Distance

Pour constituer le groupe *DistSet* de solutions-élites, une mesure de distance doit être élaborée. Resende et Ribeiro (2005) parlent de différence symétrique $\delta(U,V)$ entre les solutions U et V, vue comme des ensembles d'attributs. Pour le *LRP*, une mesure de dissimilarité $D_1(U,V)$, inspirée de Campos *et al.* (2005), peut être définie comme décrit ci-après :

- pour chaque paire (i,j) de clients consécutifs dans une tournée de U et retrouvée dans V sous la forme (i,j) ou (j,i), la paire est dite non-cassée. Elle ne contribue pas à la mesure de distance. Par contre, si i et j ne sont plus adjacents dans une tournée de V, alors, $D_1(U,V)$ peut être incrémentée de 1;
- de plus, si le nombre de tournées dans U est plus grand que dans V, $D_1(U,V)$ peut être incrémentée du nombre de tournées supplémentaires;
- enfin, le nombre de dépôts différents utilisés dans U et V peut être multiplié par 3 et le résultat est ajouté à $D_1(U,V)$.

$D_1(U,V)$ respecte les relations suivantes :

- $D_1(U,V) \geq 0$;
- $U = V \Longrightarrow D_1(U,V) = 0$;
- $D_1(U,V) = D_1(V,U)$.

Cette mesure détecte correctement les solutions équivalentes comme lorsque des mêmes tournées sont numérotées différemment ou qu'elles sont inversées (clients visités en sens inverse

Algorithme 6 Construction de DistSet - *BuildDistSet(BestSet,DistSet)*

```
1: BestSet = ensemble de NBest solutions
2: //initialisation
3: DistSet = ∅
4: NbInSet = 0
5: //mettre la meilleure solution avant PR dans DistSet
6: DistSet = DistSet ∪ {BestSol}
7: NbInSet = NbInSet + 1
8: répéter
9:    maxi = 0
10:   //pour chaque solution U de BestSet
11:   pour tout U ∈ BestSet faire
12:      DistMin = +∞
13:      //Pour chaque solution V de DistSet
14:      pour tout V ∈ DistSet faire
15:         Dist = D2(U,V) //calculer la distance entre U et V
16:         si Dist < DistMin alors
17:            DistMin = Dist //DistMin est la plus petite distance entre U et les solutions de DistSet
18:         fin si
19:      fin pour
20:      //si U est la solution de BestSet qui maximise la distance minimale avec les solutions
21:      //de DistSet, la garder en mémoire BestU
22:      si DistMin > maxi alors
23:         maxi = DistMin
24:         BestU = U
25:      fin si
26:   fin pour
27:   //ajouter BestU à DistSet
28:   NbInSet = NbInSet + 1
29:   DistSet = DistSet ∪ {BestU}
30: jusqu'à (NbInSet < NMaxDist)
```

par le véhicule). Par contre, elle est nulle si une tournée est affectée au dépôt t dans U et que la même séquence de clients est rattachée au dépôt $s \neq t$ dans V.

Une autre mesure, $D_2(U,V)$, détectant cette fois ce type de différence, est donc proposée dans la suite. Ce nouvel opérateur est plus pertinent. Par contre, la propriété $D_2(U,V) = D_2(V,U)$ n'est plus valide. Cependant, les écarts entre $D_2(U,V)$ et $D_2(V,U)$ sont relativement faibles et n'entraînent pas d'aberrations dans la constitution de *DistSet*.

La mesure $D_2(U,V)$ est définie en considérant quatre cas pour chaque paire (i,j) de clients consécutifs dans une tournée de U:

- la paire (i,j) ou (j,i) est retrouvée dans une tournée de V, la paire est donc dite non-cassée et ne contribue pas à la mesure de distance;
- i et j ne sont plus adjacents dans V, mais appartiennent toujours à une tournée commune: ajouter une constante Π_1 à $D_2(U,V)$;
- i et j se retrouvent dans des tournées différentes dans V, mais sont toujours affectés à un dépôt commun: ajouter une constante Π_2;

– i et j sont affectés à deux dépôts différents dans V: ajouter une constante Π_3.

De plus, pour chaque client i servi en début de tournée dans U, ajouter Π_3 si i est affecté à un autre dépôt dans V. $D_2(U,V)$ vérifie les propriétés suivantes :

– $D_2(U,V) \geq 0$;
– $U = V \iff D_2(U,V) = 0$.

De plus, elle repère également correctement les solutions équivalentes. Les valeurs des constantes de la mesure D_2 sont choisies ainsi : $\Pi_1 = 1$, $\Pi_2 = 2$ et $\Pi_3 = 10$. Il est cependant possible d'utiliser un autre paramétrage.

5.5.2 Path Relinking pour le *LRP*

Algorithme 7 Path relinking pour le LRP - *PathRelinking*(*DistSet*,*BestSol*)

DistSet = *ensemble de NMaxDist solutions élites*
BestSol = *meilleure solution* //meilleure solution avant *PR*
//pour chaque paire de solutions de *DistSet*
pour tout {*U*,*V*} ∈ *DistSet* **faire**
  //faire un *PR* de *U* à *V* retournant dans path toutes les solutions faisables obtenues
  Relink(U, V,path)
  //pour chaque solution réalisable *S* de path
  pour tout *S* ∈ *path* **faire**
    *LS*1(*S*) //effectuer une recherche locale
    //si BestSol est amélioré
    si (*cout*(*S*) < *cout*(*BestSol*)) **alors**
      BestSol = *S* //mettre à jour BestSol
    fin si
  fin pour
  //faire un *PR* de *V* à *U* retournant dans path toutes les solutions faisables obtenues
  Relink(V,U,path)
  //pour chaque solution réalisable *S* de path
  pour tout *S* ∈ *path* **faire**
    LS1(S) //effectuer une recherche locale
    //si *BestSol* est amélioré
    si (*cout*(*S*) < *cout*(*BestSol*)) **alors**
      BestSol = *S* //mettre à jour *BestSol*
    fin si
  fin pour
fin pour
Renvoyer(*BestSol*)

Une itération du *PR* proposé pour le *LRP* consiste à trouver un chemin orienté d'une solution U vers une solution de destination V, aussi appelée solution-guide, selon le principe décrit ci-après. La solution U est modifiée jusqu'à devenir équivalente à la solution guide V. A chaque étape, l'attribut de V intégré dans U est en fait une arête reliant deux clients de manière à réparer une paire cassée.

La réparation se fait dans un ordre donné. Toutes les paires de clients consécutifs (i,j) dans les tournées de V sont considérées successivement. Pour chacune, si les clients i et j sont non adjacents dans U, toute la séquence de clients consécutifs, présente à la fois dans U et dans V, et située après i dans V, est déplacée après i dans U. Ainsi, aucune nouvelle paire cassée n'apparaît dans U. La réparation continue ainsi tant que les capacités des tournées et des dépôts de U ne sont pas violées. Si ces contraintes ne sont plus respectées, alors une correction intervient. Pour cela, les clients provoquant les violations sont recherchés. Ce sont ceux qui appartiennent à la tournée ou au dépôt en excès de charge, et qui ne sont pas à la même position dans V. La correction déplace alors des séquences de clients comme dans la réparation des paires cassées, sauf que les paires de clients (i,j) ne sont plus prises dans l'ordre rencontré dans V, mais dans U, tant que les capacités ne sont pas entièrement respectées. Un exemple de réparation d'une paire cassée suivie d'une correction de violation de capacité est donné dans la figure 5.1.

Chaque solution réalisable rencontrée lors du *PR* est améliorée par la recherche locale *LS1* qui calcule un optimum local. L'algorithme 7 donne la structure du path relinking proposé. Il utilise une procédure *Relink* qui génère la trajectoire entre deux solutions données.

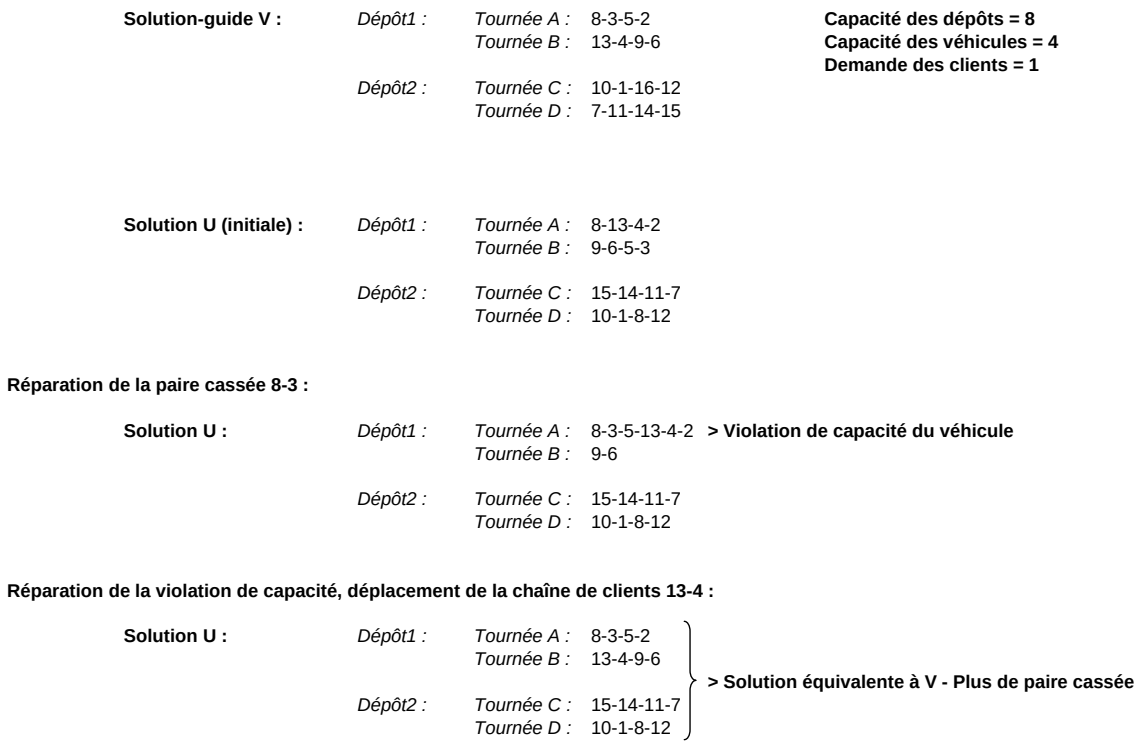


FIG. 5.1 – *Exemple de réparation d'une paire cassée dans le PR pour le LRP*

5.6 Algorithme général

L'algorithme 8 présente le pseudo-code de du principe général du *GRASP* combiné avec la technique d'apprentissage et le Path Relinking.

Algorithme 8 *GRASP* complet pour le *LRP*

1: //GRASP
2: $BestSet = \emptyset$ //ensemble des solutions en diversification
3: $BestSol = +\infty$ //coût de la meilleure solution
4: $divmode = vrai$ //commencement en phase de diversification
5: $BestCostDiv = +\infty$ //coût de la meilleure solution dans la phase de diversification
6: $SD = I$ //tous les dépôts sont permis
7: $itdiv = 0$ //compteur d'itération pour ce mode
8: **pour** ($k = 1$ *à* $MaxIt$) **faire**
9: *RECWA(SD,S(k))* //génèration d'une solution $S(k)$ utilisant les dépôts de SD
10: *LS1(S(k))* //amélioration par recherche locale
11: **si** ($cout(S(k)) < cout(BestSol)$) **alors**
12: $BestSol = S(k)$ //mise à jour de la meilleure solution trouvée
13: **fin si**
14: **si** ($divmode = vrai$) **alors**
15: **si** ($cout(S(k)) < BestCostDiv$) **alors**
16: $BestSoldiv = S(k)$ //mise à jour de la meilleure solution trouvée en diversification
17: $BestCostDiv = cout(S(k))$ //et de son coût
18: **fin si**
19: $itdiv = itdiv + 1$ //compteur d'itérations
20: **si** (itdiv = maxitdiv) **alors**
21: $divmode = faux$ //changement en mode d'intensification
22: $itint = 0$ //ré-initialisation du compteur d'itération pour ce mode
23: $SD =$ *dépôts de BestSoldiv* //restriction des dépôts disponibles
24: **fin si**
25: **sinon**
26: $itint = itint + 1$ //compteur d'itérations pour ce mode
27: **si** ($itint = maxitint$) **alors**
28: $divmode = vrai$ //changement en mode de diversification
29: $BestCostDiv = +\infty$ //ré-initialisation du coût de la meilleure solution pour ce mode
30: $SD = I$ //rétablissement de tous les dépôts disponibles
31: $itdiv = 0$ //ré-initialisation du compteur d'itération pour ce mode
32: **fin si**
33: **fin si**
34: **fin pour**
35: //PATH RELINKING
36: $BestSet =$ *NBest meilleures solutions de diversification*
37: **si** $BestSol \notin BestSet$ **alors**
38: $BestSet = BestSet \cup \{BestSol\}$ //ajout de la meilleure solution trouvée à S
39: **fin si**
40: *BuildDistSet(BestSet,DistSet)* //création du sous-ensemble de solutions-élites
41: *PathRelinking (DistSet,BestSol)* //appelle le path relinking et retourne la meilleure solution finale

5.7 Évaluation Numérique

5.7.1 Implémentation et instances

L'algorithme est codé en C et exécuté sur un PC Dell Optiplex GX260, avec un Pentium 4 à 2.4 GHz, 512 Mo de RAM et Windows XP.

Il est testé sur des instances pour le LRP avec capacité sur les tournées et les dépôts (*I1* et *I3*), ainsi que sur des instances sans capacité sur les dépôts (*I2*), décrites en Annexe. *I3* est un jeu d'instances non utilisé dans le chapitre précédent qui se différencie de *I1* par la structure des coûts. Les coûts fixes d'ouverture des dépôts prennent une part moins importante dans le coût final de la solution (comme dans les instances de type $I2$) et le coût fixe des véhicules est nul. Les résultats sont comparés avec HS (*Heuristique Simple*). HS est en fait l'heuristique obtenant les meilleurs résultats au chapitre précédent. Il s'agit d'un algorithme de base, utilisant un principe inspiré de l'heuristique du PPV, exécuté 10 fois, la meilleure solution obtenue étant améliorée par la recherche locale *LS2* du chapitre 4. Pour les instances avec capacité sur les dépôts, HS correspond plus exactement à $AB2 \times 10 + LS2$, sinon elle représente $AB3 \times 10 + LS2$ (voir chapitre 4).

Les résultats sont donnés dans les tableaux 5.1 - 5.2 et 5.3. Les colonnes n et m indiquent respectivement le nombre de clients et de dépôts de l'instance. Les autres caractéristiques n'apparaissaient pas dans les tableaux du chapitre précédent car les résultats étaient fournis en moyenne par sous-groupe d'instances de taille équivalente. β est le nombre de clusters, *type* est soit relatif à la capacité des tournées (instances $I1$ et $I3$), soit correspond au ratio des clients appartenant à des clusters sur n (instances $I2$) (voir Annexe). Les colonnes *Coût* et *Tps* donnent le coût des solutions obtenues et les temps de calcul nécessaire en secondes pour les obtenir. *dep* et *tr* indiquent le nombre de dépôts et de tournées ouvertes dans la solution restituée. Enfin, la colonne E_{HS} est l'écart relatif en pourcentage entre le coût de la solution trouvée par le $GRASP$ et HS : $E_{HS} = ((Cout(GRASP) - Cout(HS))/Cout(HS)) * 100$. Mais tout d'abord, le paramétrage de l'algorithme utilisé pour obtenir ces résultats est donné au paragraphe suivant.

5.7.2 Paramétrage de l'algorithme

Les paramètres du $GRASP$ avec PR utilisés pour obtenir les résultats présentés sont les suivants : $MaxIt$=60, $NBest$=10, $NMaxDist$=3, bm=7, $maxitdiv$=5 et $maxitint$=7. Ils ont été choisis après des tests préliminaires, afin de réaliser de bon compromis entre temps de calcul et qualité des solutions.

En effet, le temps pour construire une solution n'étant pas négligeable, le nombre total d'itérations du $GRASP$ doit rester assez réduit. La diversification explore l'espace des solutions afin de rechercher un sous-ensemble de dépôts permettant d'obtenir de bons résultats. Selon les expérimentations, *maxitdiv* est fixé à 5. L'intensification quant à elle intervient plutôt dans le but d'améliorer les tournées de véhicules à partir des dépôts sélectionnés. La recherche locale utilisée étant dédiée aux tournées, un grand nombre d'itérations par phase d'intensification est inutile, les solutions construites par l'heuristique gloutonne risquant de converger vers un même optimum local. C'est pourquoi, *maxitint* est fixé à 7. Sachant que la cardinalité de

l'ensemble I dans les instances est de 20 maximum, que $maxitdiv = 5$ et qu'il faut ouvrir chaque dépôt au moins une fois lors des diversifications, ceci donne le nombre minimum de phases à effectuer. Nous en avons ajouté tout de même une afin d'explorer un peu plus l'espace des solutions.

La taille de *BestSet* correspond aux 40% des meilleures solutions trouvées en diversification. C'est une valeur permettant de conserver de bonnes solutions tout en gardant suffisamment de choix pour créer un bon ensemble *DistSet* de solutions distantes. La taille de *DistSet* est choisie afin d'avoir un compromis entre les temps de calcul et l'exploration de l'espace des solutions lors de la post-optimisation avec Path Relinking, soit $NMaxDist$=3. Finalement, bm=7 reflète une taille maximale de la *RCL*. Cette valeur donne la possibilité de varier les choix dans l'heuristique gloutonne, mais sans tomber dans une randomisation trop importante, une diversification étant déjà en partie introduite dans le choix des dépôts dans la solution triviale donnée par le bouquet de marguerites.

5.7.3 Résultats pour le premier groupe d'instances - *I1*

Les résultats montrent que la métaheuristique proposée ici améliore ceux de l'heuristique *HS* proposée au chapitre 4. Le gain peut même être assez important, jusqu'à 16.73% sur une instance de petite taille. Ceci peut être dû au fait que le choix des dépôts ouverts est très influent. Par exemple, en regardant l'instance 20-5-0-b, le nombre de dépôts et de tournées ouverts sont identiques avec les deux méthodes. Notons que sur ces instances, le coût d'ouverture d'un dépôt n'est pas identique pour tous. Ainsi, le choix de la localisation est suffisamment influent pour induire une différence de coût importante (16.73%). Plus précisément, à titre indicatif, la solution de *HS* est composée d'un coût total sur les dépôts égal à 20788 contre 15497 dans la solution du *GRASP*. Sur l'instance 50-5-0-a, le *GRASP* trouve une solution dans laquelle un dépôt de plus est ouvert par rapport à celle fournie par *HS*, tout en induisant un coût total inférieur de 4.97%. Le but n'est donc pas forcément de chercher à réduire le nombre de dépôts à ouvrir, mais bien de trouver un bon compromis entre le nombre de dépôts ouverts, leur positionnement sur le graphe et les tournées de véhicules, de manière à trouver une combinaison minimisant du coût total de la solution.

La meilleure qualité des solutions obtenues par le *GRASP* se fait cependant au détriment des temps de calcul moyens qui passent de 1 s pour *HS* à 96 s. Notons néanmoins que les décisions qui sont prises sont de type stratégique, à long terme, et donc que les gains réalisés méritent bien quelques dizaines de secondes supplémentaires.

5.7.4 Résultats pour le deuxième groupe d'instances - *I2*

Le coûts des solutions obtenues par le *GRASP* sur *I2* est significativement réduit par rapport à *HS*. Le gain relatif est en moyenne de 10.07 %. Comme au chapitre 4, *RECWA* (version randomisée de *ECWA* pour le *GRASP*) obtient de bons résultats sur les instances sans capacité sur les dépôts. Encore une fois, le choix des sites ouverts joue un rôle important même si sur ces instances leur coût d'ouverture sont tous égaux. Malgré le fait qu'un seul dépôt puisse servir l'ensemble des clients, il est intéressant d'en utiliser plusieurs. Ceci permet des gains non négligeables sur le routage. D'ailleurs, en moyenne, les solutions obtenues par

				HS				*GRASP*				
n	m	β	type	Coût	Tps	dep	tr	Coût	Tps	dep	tr	E_{HS} (%)
20	5	0	a	57410	0.0	3	5	55021	0.2	3	5	-4.16
20	5	0	b	46959	0.0	2	3	39104	0.2	2	3	-16.73
20	5	2	a	52338	0.0	3	5	48908	0.1	3	5	-6.55
20	5	2	b	37542	0.0	2	3	37542	0.2	2	3	0.00
50	5	0	a	95369	0.1	2	13	90632	1.8	3	12	-4.97
50	5	0	b	66023	0.1	2	6	64761	1.8	2	6	-1.91
50	5	2	a	96058	0.1	3	13	88786	2.4	3	12	-7.57
50	5	2	b	71675	0.1	3	6	68042	2.5	3	6	-5.07
50	5	2'	a	87530	0.1	3	12	84055	1.7	3	12	-3.97
50	5	2'	b	52720	0.1	3	6	52059	2.6	3	6	-1.25
50	5	3	a	87499	0.1	2	12	87380	2.3	2	12	-0.14
50	5	3	b	64464	0.1	2	6	61890	2.0	2	6	-3.99
100	5	0	a	297904	0.5	3	25	279437	27.6	3	24	-6.20
100	5	0	b	231373	0.5	3	12	216159	23.2	3	12	-6.58
100	5	2	a	199652	0.6	2	25	199520	17.4	2	24	-0.07
100	5	2	b	161621	0.5	2	11	159550	22.4	2	11	-1.28
100	5	3	a	203351	0.5	2	25	203999	21.6	2	25	0.32
100	5	3	b	156647	0.6	2	12	154596	20.3	2	11	-1.31
100	10	0	a	329917	0.6	4	26	323171	37.4	4	26	-2.04
100	10	0	b	280512	0.5	4	12	271477	29.5	4	12	-3.22
100	10	2	a	279961	0.6	3	25	254087	39.1	3	25	-9.24
100	10	2	b	229129	0.6	3	11	206555	29.8	3	11	-9.85
100	10	3	a	262022	0.5	3	25	270826	35.4	3	25	3.36
100	10	3	b	206525	0.7	3	12	216173	39.8	3	11	4.67
200	10	0	a	531092	4.3	3	49	490820	517.5	3	48	-7.58
200	10	0	b	431668	3.9	3	22	416753	379.1	3	22	-3.46
200	10	2	a	490036	4.8	3	50	512679	554.3	3	49	4.62
200	10	2	b	391996	4.1	3	23	379980	367.4	3	23	-3.07
200	10	3	a	494746	4.6	3	49	496694	424.8	3	46	0.39
200	10	3	b	394470	4.1	3	22	389016	290.2	3	22	-1.38
Moyenne					1.1	2.7	17.5		96.5	2.8	17.2	-3.27

NB : un écart négatif indique une amélioration par rapport à la solution prise en référence.

TAB. 5.1 – *Résultats du GRASP - Instances I1 avec capacité sur les dépôts*

le *GRASP* ouvrent plus de sites tout en réduisant les coûts totaux. Cependant, ce n'est pas toujours le cas. Sur l'instance 100-20-0-1 par exemple, *HS* en ouvre 3 contre 2 seulement pour le *GRASP*, ce dernier obtenant un gain de plus de 11%.

Les conclusions sur les temps de calcul sont similaires à celles faites pour les instances *I1*. Cependant, il faut noter que les coûts sont nettement inférieurs avec le *GRASP* (presque 10% de gain par rapport à *HS*). L'intérêt de la métaheuristique est donc plus marqué. Dans le paragraphe 5.8, une explication de la différence des temps de calcul selon le type d'instance est proposée.

				HS				*GRASP*				
n	m	β	type	Coût	Tps	dep	tr	Coût	Tps	dep	tr	E_{HS} (%)
100	10	0	0.75	1621.66	0.9	2	12	1525.25	32.4	3	11	-5.95
100	20	0	0.75	1734.43	1.0	2	11	1526.90	40.7	3	11	-11.96
100	10	0	1	1457.57	0.9	3	11	1423.54	27.6	2	11	-2.33
100	20	0	1	1672.82	1.1	3	12	1482.29	36.2	2	11	-11.39
100	10	3	0.75	1345.17	1.1	3	11	1200.24	27.7	2	11	-10.77
100	20	3	0.75	1334.31	1.1	3	11	1123.64	34.3	3	11	-15.79
100	10	3	1	825.75	1.1	2	12	814.00	22.5	3	12	-1.42
100	20	3	1	1006.91	1.2	2	11	747.84	37.3	3	11	-25.73
100	10	5	0.75	1385.95	1.1	2	11	1273.10	21.5	3	11	-8.14
100	20	5	0.75	1303.17	1.2	2	11	1272.94	36.0	2	11	-2.32
100	10	5	1	974.29	1.0	3	12	912.19	20.3	3	12	-6.37
100	20	5	1	1168.28	1.0	2	11	1022.51	38.4	3	11	-12.48
150	10	0	0.75	2024.41	3.6	3	16	2006.70	113.0	3	16	-0.88
150	20	0	0.75	1964.31	3.3	3	17	1888.90	161.4	4	16	-3.84
150	10	0	1	2102.04	3.3	2	16	2033.93	100.0	3	17	-3.24
150	20	0	1	2031.61	3.0	2	16	1856.07	132.4	4	16	-8.64
150	10	3	0.75	1574.04	3.8	2	16	1508.33	117.7	3	16	-4.17
150	20	3	0.75	1539.31	3.5	2	17	1456.82	166.1	2	16	-5.36
150	10	3	1	1299.25	4.0	2	16	1240.40	106.7	2	16	-4.53
150	20	3	1	1169.77	3.8	2	17	940.80	142.4	3	17	-19.57
150	10	5	0.75	1900.46	3.7	3	18	1736.90	92.8	3	17	-8.61
150	20	5	0.75	1642.43	3.9	3	18	1425.74	128.4	3	16	-13.19
150	10	5	1	1512.69	4.0	2	17	1223.70	88.5	3	17	-19.10
150	20	5	1	1412.73	3.8	3	17	1231.33	134.9	4	17	-12.84
200	10	0	0.75	2486.07	8.0	3	22	2384.01	308.0	3	21	-4.11
200	20	0	0.75	2427.49	7.8	3	22	2288.09	410.0	4	22	-5.74
200	10	0	1	2382.77	8.3	3	23	2273.19	311.4	3	21	-4.60
200	20	0	1	2511.30	7.9	3	22	2345.10	418.9	3	22	-6.62
200	10	3	0.75	2372.72	8.9	3	22	2137.08	338.0	3	22	-9.93
200	20	3	0.75	2010.45	9.1	3	22	1807.29	370.0	4	21	-10.11
200	10	3	1	1819.35	9.1	3	21	1496.75	242.7	2	21	-17.73
200	20	3	1	1640.91	10.0	2	22	1095.92	308.5	3	22	-33.21
200	10	5	0.75	2184.87	8.7	3	22	2044.66	282.8	4	23	-6.42
200	20	5	0.75	2466.92	8.6	2	21	2090.95	399.2	4	22	-15.24
200	10	5	1	1843.43	8.3	2	23	1788.70	199.0	2	21	-2.97
200	20	5	1	1938.76	8.9	2	22	1408.63	296.3	5	22	-27.34
	Moyenne				4.4	2.5	16.7		159.6	3.0	16.4	-10.07

NB : un écart négatif indique une amélioration par rapport à la solution prise en référence.

TAB. 5.2 – *Résultats du GRASP - Instances I2 sans capacité sur les dépôts*

5.7.5 Résultats pour le troisième groupe d'instances - *I3*

Le tableau 5.3 montre que le *GRASP* avec *PR* dépasse les performances de *HS* en terme de coût. Une fois encore, si en moyenne le nombre de dépôts et de tournées ouverts sont quasiment identiques avec les deux méthodes, une observation plus attentive permet de constater qu'une baisse du nombre de dépôts utilisés n'implique pas forcément une réduction du coût total de la solution.

En ce qui concerne les temps de calcul, aucune méthode ne nécessite plus d'une seconde avec moins de 50 clients. Par contre, pour les plus grosses instances, si *HS* obtient des résultats en moins de 3 s, le *GRASP* peut parfois nécessiter plusieurs minutes. Cependant, comme sur les instances précédentes, le gain apporté sur le coût est intéressant et les temps de calcul sont raisonnables pour des décisions de type stratégique.

			HS				*GRASP*				
n	m	type	Coût	Tps	dep	tr	Coût	Tps	dep	tr	E_{HS} (%)
21	5	6000	435.4	0.0	2	5	429.6	0.2	2	5	-1.34
22	5	4500	619.0	0.0	2	4	585.1	0.2	1	3	-5.48
27	5	2500	3068.4	0.0	2	4	3062.0	0.4	2	4	-0.21
29	5	4500	520.2	0.0	2	4	515.1	0.4	2	4	-0.96
32	5	8000	609.6	0.0	2	5	571.9	0.6	2	4	-6.17
32	5	11000	551.5	0.0	2	4	504.3	0.5	1	3	-8.55
36	5	250	500.9	0.0	2	5	460.4	0.8	1	4	-8.10
50	5	160	598.7	0.1	2	7	599.1	2.3	3	8	0.08
75	10	140	909.3	0.5	2	10	861.6	9.8	3	9	-5.25
88	8	9000000	370.6	0.6	2	7	356.9	17.3	2	8	-3.70
100	10	200	896.9	1.0	2	8	861.6	25.5	3	9	-3.95
134	8	850	7145.2	1.7	3	11	5965.1	49.6	4	11	-16.52
150	10	8000000	47223.1	2.8	3	12	44625.2	156.0	3	13	-5.50
Moyenne				0.5	2.2	6.6		20.3	2.2	6.5	-5.05

NB : un écart négatif indique une amélioration par rapport à la solution prise en référence.

TAB. 5.3 – *Résultats du GRASP - Instances I3 avec capacité sur les dépôts*

5.8 Impact des composants de l'algorithme

Comme le montre les tableaux 5.1, 5.2 et 5.3, le *GRASP* peut fournir des solutions de bien meilleure qualité qu'une heuristique constructive suivie d'une recherche locale (-6.66% en moyenne sur les coûts par rapport à *HS*). Mais il peut être intéressant de comprendre l'intérêt des différents composants ajoutés à l'algorithme, en particulier l'alternance entre intensification et diversification sur un sous-ensemble de dépôts et la post-optimisation par le Path Relinking.

5.8.1 Impact de la mémorisation

Une mémorisation sur les dépôts a été ajoutée à la structure classique du *GRASP*, afin d'alterner entre des phases de diversification et d'intensification. Cette intensification permet de concentrer la recherche sur un secteur de l'espace des solutions (nombre de dépôts disponibles réduits) lors de la construction de la solution par l'heuristique gloutonne randomisée.

		GRASP		Sans Mem		
n	m	Coût	Tps	Coût	Tps	E_{GRASP}
Premier ensemble - $I1$:						
20	5	45144	0.2	45107	0.2	-0.07
50	5	74701	2.1	74497	2.6	-0.29
100	5	202210	22.1	202015	24.4	-0.16
100	10	257048	35.2	252825	47.2	-1.66
200	10	447657	422.2	432896	496.7	-3.01
Moyenne			96.5		114.4	-1.05
Deuxième ensemble - $I2$:						
100	10	1191.39	25.3	1192.71	35.8	0.03
100	20	1196.02	37.2	1197.98	59.4	0.11
150	10	1624.99	103.1	1622.50	129.8	0.06
150	20	1466.61	144.3	1474.78	225.7	0.49
200	10	2020.73	280.3	2030.07	361.1	0.69
200	20	1839.33	367.1	1848.26	590.5	0.81
Moyenne			159.6		233.7	0.37
Troisième ensemble - $I3$:						
≤ 50	5	840.95	0.7	847.18	0.8	1.31
≤ 150	≤ 10	10534.05	51.6	10655.69	61.5	0.80
Moyenne			20.3		24.2	1.11
Moyenne			**112.7**		**153.9**	**-0.05**

NB : un écart négatif indique une amélioration par rapport à la solution prise en référence.

TAB. 5.4 – *Impact de la mémorisation*

Le tableau 5.4 présente l'influence de cette mémorisation sur le comportement de l'algorithme. *GRASP* est la version complète de la méthode, *Sans Mem* est la version sans intensification. Les colonnes *Coût* et *Tps* donnent le coût moyen des solutions et les temps de calcul moyens nécessaire en secondes par sous-ensemble d'instances. E_{GRASP} représente le gain relatif sans intensification par rapport au *GRASP* complet.

Le fait d'intensifier la recherche concentre cette dernière sur un sous-ensemble de dépôts prometteurs selon la dernière phase de diversification. Ainsi, bien que les variables concernant l'ouverture des dépôts soit parfois fixées, limitant l'espace de recherche, la technique de mémorisation améliore 50% des solutions obtenues, tout en apportant de la rapidité d'exécution. En effet, l'utilisation de phases d'intensification réduit d'environ 30% les temps de calcul.

Par contre, le tableau 5.4 montre une très légère dégradation de la qualité moyenne des résultats. En fait, si sur les instances des ensembles *I2* et *I3*, la mémorisation est très profitable aussi bien en terme de coût que de temps de calcul, l'avantage apporté sur les instances de *I1* est moins évident. La réduction des temps d'exécution est incontestable. Par contre, les solutions obtenues sont de coût plus élevé, en particulier sur les plus grosses instances. L'intensification est alors certainement trop restrictive. Pourtant, la cardinalité de I n'est pas particulièrement grande (maximum 10 dépôts). Cependant, ce qui n'apparaît pas dans les tableaux, c'est le rapport entre les coûts relatifs aux dépôts et ceux liés aux tournées. Sur *I1*, ce rapport est environ 10 fois plus grand que sur les autres jeux d'instances, rendant le choix sur l'ouverture des dépôts très influent. Selon la structure des instances, la mémorisation est donc plus ou moins intéressante.

5.8.2 Impact du Path Relinking

Le tableau 5.5 montre l'intérêt d'une post-optimisation avec le path relinking. *GRASP* est la version complète de la méthode, *Sans PR* est la version sans post-optimisation. Les colonnes *Coût* et *Tps* donnent le coût moyen des solutions et les temps de calcul moyens nécessaires en secondes par sous-ensemble d'instances. E_{GRASP} représente le gain relatif sans path relinking par rapport au *GRASP* complet.

L'impact du *PR* est confirmé. Il permet une amélioration de la solution de 0.89% en moyenne. Néanmoins, les résultats montrent que les instances de *I1*, avec des capacités limitées sur les dépôts, profitent moins du *PR* par rapport au cas sans capacité. En effet, le gain moyen n'est alors que de 0.3%.

Durant le *PR*, pour chaque réparation de paire cassée de clients, des violations de capacités peuvent apparaître, nécessitant la réparation d'autres paires cassées de clients pour obtenir une solution réalisable. Les violations sont plus probables lorsque ce type de contrainte porte à la fois sur les véhicules et les dépôts. Le nombre de solutions réalisables visité durant la post-optimization est ainsi moins grand, et cette dernière perd de l'intérêt. De ce fait, seules 2/3 des solutions des instances avec capacité limitée sur les dépôts sont améliorées durant le *PR*, contre près de 90% de celles des autres jeux d'instances.

La post-optimisation proposée demandant des temps de calcul relativement importants (elle représente à elle seule moitié du temps passé dans l'algorithme complet en moyenne sur l'ensemble des instances), et spécialement lorsque des réparations de violations de capacité

		GRASP		Sans PR		
n	m	Coût	Tps	Coût	Tps	E_{GRASP}
Premier ensemble - $I1$:						
20	5	45144	0.2	45151	0.1	0.01
50	5	74701	2.1	74997	1.4	0.39
100	5	202210	22.1	203029	9.1	0.37
100	10	257048	35.2	258748	15.2	0.68
200	10	447657	422.2	448397	112.1	0.16
Moyenne			96.5		27.7	0.35
Deuxième ensemble - $I2$:						
100	10	1191.39	25.3	1210.71	15.8	1.60
100	20	1196.02	37.2	1205.72	26.8	0.90
150	10	1624.99	103.1	1636.66	52.2	0.70
150	20	1466.61	144.3	1493.11	90.9	1.84
200	10	2020.73	280.3	2052.86	126.0	1.58
200	20	1839.33	367.1	1861.52	215.4	1.14
Moyenne			159.6		87.9	1.29
Troisième ensemble - $I3$:						
≤ 50	5	840.95	0.7	845.93	0.5	0.91
≤ 150	≤ 10	10534.05	51.6	10545.07	26.0	1.26
Moyenne			20.3		10.3	1.04
Moyenne			**112.7**		**52.2**	**0.89**

NB : un écart négatif indique une amélioration par rapport à la solution prise.

TAB. 5.5 – *Impact du Path Relinking*

sont nécessaires, il peut être envisagé d'appliquer le *PR* uniquement sur des problèmes sans capacité sur les dépôts. Ainsi, pour l'ensemble *I1*, l'amélioration des solutions par rapport à *HS* reste importante (-2.92% au lieu de -3.27%, en comparant les tableaux 5.1 et 5.5) et les temps de calcul diminuent fortement.

5.9 Conclusion

Dans ce chapitre, une nouvelle métaheuristique pour le *LRP* avec contraintes de capacité sur les tournées et les dépôts a été proposée. La méthode est un *GRASP* qui utilise une généralisation et une randomisation de l'algorithme de Clarke et Wright, une mémoire permettant de guider la recherche en agissant sur la localisation des dépôts, et un Path Relinking utilisé en post-optimisation.

Cette métaheuristique est efficace puisqu'elle améliore près de 95% des solutions obtenues par la meilleure approche heuristique du chapitre 4, qui utilise déjà une recherche locale. Les performances les plus remarquables sont atteintes sur les instances sans capacités sur les dépôts (gain triplé par rapport aux instances avec capacité). Ceci s'explique en partie par le fait que les techniques ajoutées au *GRASP* (mémorisation et *PR*) profitent plus aux instances des ensembles *I2* et *I3*, qui ont des coûts d'ouvertures de dépôts moins importants que dans l'ensemble *I1* et qui, pour *I2*, n'ont pas de capacité limitée sur les dépôts.

Le *GRASP* explorant plus de solutions, les temps de calcul sont plus longs comparés à ceux de *HS* mais restent raisonnables pour un problème stratégique de taille réaliste puisqu'ils ne dépassent pas quelques minutes. Il est néanmoins possible d'obtenir des résultats plus rapidement en stoppant la méthode avant l'exécution de la post-optimisation par *PR*. Les coûts obtenus ne sont alors dégradés que de 1% en moyenne par rapport à ceux découlant de la méthode complète. L'intérêt est plus marqué pour les instances avec capacité sur les dépôts pour lesquelles, la hausse moyenne est bien plus faible (0.3% seulement) pour des temps réduits de 2/3.

Une première version du *GRASP* pour le *LRP* a été présentée dans la conférence internationale avec actes *International Conference on Industrial Engineering and Systems Management* - IESM'05 (Prins *et al.*, 2005c) et la version finale est déjà disponible en ligne sur le site web de *4OR - A Quarterly Journal of Operations Research* (Prins *et al.*, 2006b).

Le *GRASP* apporte donc des résultats intéressants. Cependant, il construit des solutions les unes après les autres sans exploiter les relations qui peuvent exister entre elles. Cette exploitation n'intervient que tardivement, quand le *PR* est utilisé en post-optimisation. Le chapitre suivant propose de tester un autre type de métaheuristique pour le *LRP*. Il s'agit cette fois-ci d'une vraie méthode à population, plus agressive. Des solutions bien diversifiées obtenues lors des itérations du *GRASP* peuvent être conservées pour son initialisation.

Chapitre 6

Algorithme mémétique avec gestion de la population - *MA|PM*

6.1 Introduction

Dans le chapitre 5, un algorithme d'échantillonnage de l'espace des solutions est proposé pour le *LRP*. Il s'agit d'un *GRASP*.

Une autre grande famille de métaheuristiques pour résoudre les problèmes d'optimisation combinatoire est celle des algorithmes dits évolutifs. Ce sont généralement des méthodes qui utilisent des mécanismes imitant des phénomènes observés en biologie. On peut citer les algorithmes à colonies de fourmis, et plus spécifiquement, ceux basés sur la biologie génétique, s'inspirant de phénomènes comme la sélection naturelle, la reproduction, les mutations, etc. Ils fonctionnent généralement de manière itérative : une population initiale de solutions réalisables évolue progressivement, guidée par les mécanismes cités précédemment. Les algorithmes de ce type étaient à l'origine les algorithmes génétiques (Holland, 1975; Reeves, 2003), mais d'autres formes plus évoluées ont fait leur apparition.

Ce chapitre propose une version toute récente ayant déjà prouvé ses performances sur des problèmes de tournées (Prins *et al.*, 2004b; Boudia *et al.*, 2006) [illegible]

6.2 Présentation du $MA|PM$

6.2.1 Principe des algorithmes de type génétique

Le principe de base d'un algorithme génétique (GA) est exposé au paragraphe 2.2.3.3 du chapitre 2. Il consiste à partir d'une population de solutions et à la faire évoluer.

Les résultats obtenus par les GA sont en principe de bonne qualité mais souvent, ils ne rivalisent pas avec les approches de type tabou par exemple. Par contre, des versions plus récentes introduisent des compléments dans la méthode pour la rendre plus performante, citons les algorithmes mémétiques (*memetic algorithm* - MA), une forme hybridisée généralement avec une recherche locale (Moscato (1989); Moscato et Cotta (2003); Lima *et al.* (2004) ou Prins (2004)).

Une nouvelle génération a même été proposée très récemment par Sörensen et Sevaux (2003). Il s'agit d'une forme agrémentée d'une gestion de la population de solutions. Cette version s'appelle $MA|PM$ pour *Memetic Algorithm with Population Management*. Elle utilise une mesure de distance afin d'apporter une certaine diversité dans les chromosomes-parents. Ces versions hybrides permettent de combler l'écart entre les performances des GA et des TS.

6.2.2 Principe du $MA|PM$

Le principe du $MA|PM$ étant appliqué dans la suite au LRP, ce paragraphe détaille son fonctionnement général. Il est basé sur un algorithme génétique mais se différencie des versions classiques par trois principaux éléments :

1. Une amélioration des solutions par une recherche locale (*Memetic Algorithm* - MA);
2. Une technique de gestion de la population par une mesure de distance (*Population Management* - PM);
3. Une petite population P de solutions de bonne qualité.

Le PM signifie qu'une nouvelle solution T ne peut intégrer la population courante que si sa distance $d_P(T)$ à la population courante P est telle que $d_P(T) \geq \Delta$, avec Δ un seuil donné.

Autrement, Sörensen et Sevaux proposent deux options : soit l'enfant est muté jusqu'à ce que sa distance à la population atteigne au moins le seuil Δ, soit il est tout simplement rejeté. Dans le premier cas, plusieurs mutations successives peuvent être nécessaires et le temps passé pour obtenir un enfant admissible dans la population peut être parfois important. Dans la version proposée ici pour le LRP, c'est la seconde stratégie qui est retenue.

Si $\Delta = 0$, l'algorithme se comporte comme un classique MA (sans gestion de la population). $\Delta \geq 1$ garantit que chaque solution de la population P est distincte des autres (la distance définie est entière). Par contre, si Δ est fixé à une grande valeur, alors la majorité des solutions-enfants est rejetée et l'algorithme passe beaucoup de temps dans des itérations improductives. Trouver un juste équilibre est parfois délicat. La valeur de Δ peut être alors ajustée dynamiquement entre ces deux extrêmes afin de contrôler la diversité de la population.

Sörensen et Sevaux (2003) proposent différentes politiques de contrôle pour Δ :

1. Δ constant : garantit un niveau constant de diversification;
2. Initialiser Δ avec une grande valeur, puis décroître doucement au fur des itérations : favorise une intensification en fin d'algorithme;
3. Initialiser Δ avec faible valeur, puis l'augmenter au fil des itérations : permet une diversification en fin d'algorithme;
4. Initialiser Δ avec une grande valeur au début de l'algorithme, puis décroître doucement tant que des solutions améliorant la meilleure sont trouvées, sinon, après un certain nombre d'itérations sans amélioration de la meilleure solution, ré-augmenter la valeur de Δ afin d'introduire de la diversification dans la population : stratégie adaptative ajustant dynamiquement la valeur de Δ pour contrôler la diversité de la population.

Les performances du $MA|PM$ ont déjà été testées sur le problème de tournées sur arcs avec capacités (Prins *et al.*, 2004b). Les résultats témoignent du fait que la méthode est très prometteuse. En effet, sur ces problèmes, $MA|PM$ a montré qu'il convergeait plus rapidement qu'un algorithme mémétique conventionnel, alors que sa structure générale est bien plus simple que les autres métaheuristiques basées sur une population avec mesure de distance, telles que les méthodes de type recherche dispersée (SS) ou path relinking. De plus, il est très facile de modifier un algorithme mémétique existant en $MA|PM$.

L'observation de ces caractéristiques a encouragé le choix d'une telle approche pour le LRP.

6.3 Adaptation au *LRP*

6.3.1 Chromosomes et évaluation

Parmi les éléments caractéristiques des algorithmes de type génétique, le codage des solutions en chromosomes adéquats est une décision particulièrement critique, surtout pour des problèmes comportant différents niveaux d'information, comme c'est le cas pour le problème de localisation-routage. En effet, un chromosome doit ici contenir le statut des dépôts, l'affectation des clients, ainsi que l'ordre de visite des clients servis par un même véhicule. De plus, afin de faciliter l'exploitation de ces chromosomes (lors des croisements ou de la mesure de distance pour la gestion de la population par exemple), il est important de trouver un codage de longueur fixe.

Afin de regrouper les 2 niveaux de décision dans le chromosome, le codage choisi ici se fait en 2 parties : DS (*Depot Status*) qui donne le statut des dépôts, et CS (*Customer Sequence*) qui est une séquence de clients. Les figures 6.1 et 6.2 illustrent cette représentation.

La partie DS est un vecteur de m nombres, chacun représentant le statut d'un dépôt :

- si $DS(i) = 0$, alors le dépôt i est fermé;
- sinon, la valeur $DS(i)$ donne la position (indice) du premier client affecté au dépôt i dans la séquence CS.

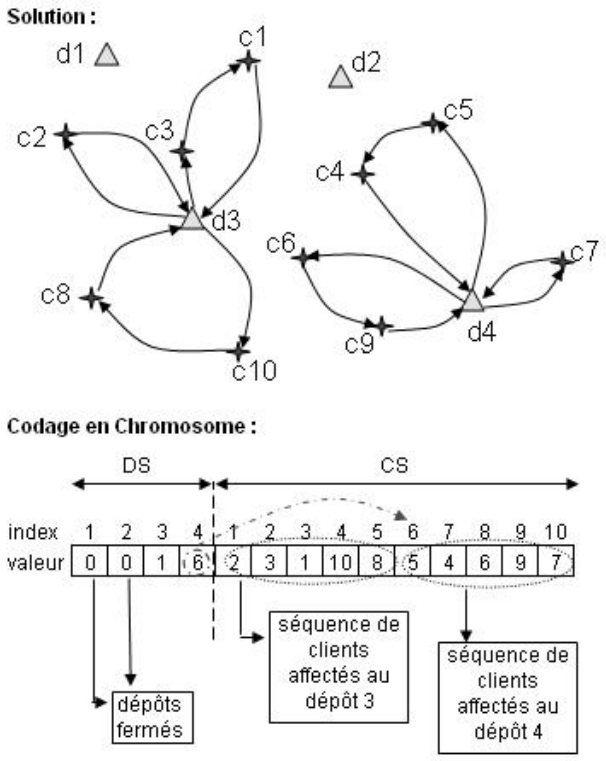


FIG. 6.1 – *Exemple de représentation d'une solution du LRP en chromosome*

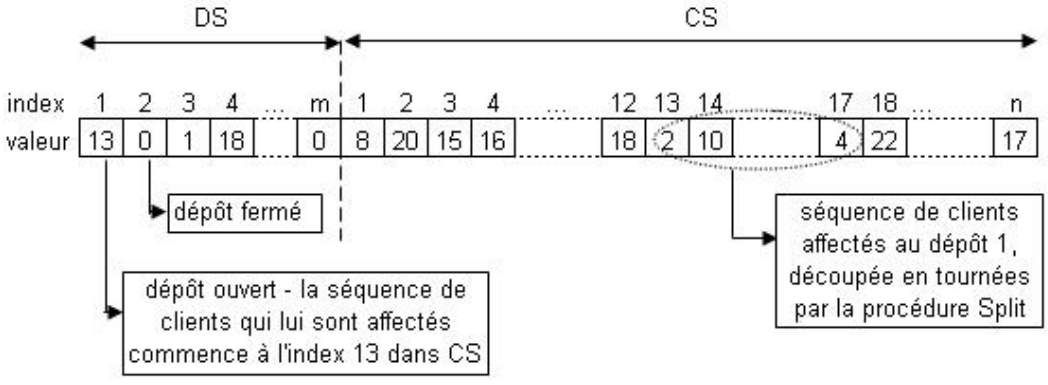


FIG. 6.2 – *Représentation générale d'une solution du LRP en chromosome*

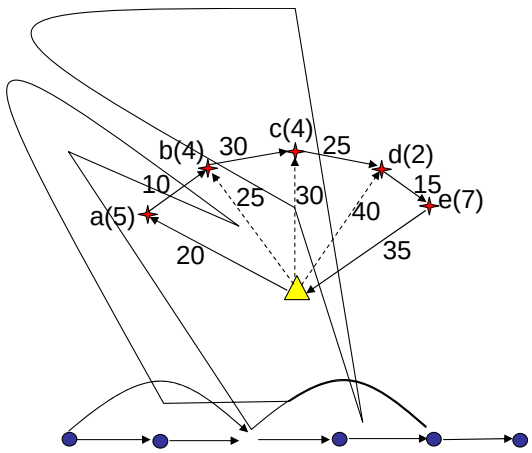


FIG. 6.3 – *Exemple de la procédure Split*

CS est une permutation de l'ensemble de tous les clients et a donc une taille fixe égale à n. C'est en fait une concaténation de tournées, les clients étant donnés dans l'ordre de visite par les véhicules, sans délimitation. Les tournées sont facilement retrouvables grâce à l'algorithme *Split* utilisé par Prins (2004) pour le VRP, comme expliqué plus loin.

Chaque chromosome est évalué par une valeur appelée *fitness* $F(S)$, représentée par le coût total de la solution réalisable S. Pour trouver le fitness du chromosome S, DS est d'abord utilisé. Il permet de connaître les dépôts ouverts, donc le coût d'ouverture associé, et pour chacun d'eux, la séquence de clients qui lui sont affectés. Chacune de ces séquences peut être découpée optimalement en tournées grâce à l'utilisation de la procédure *Split*. C'est une procédure qui a été développée à l'origine pour le VRP. Pour une séquence de p clients, on construit un graphe auxiliaire $H = (X,A,Z)$ avec X un ensemble de $p+1$ nœuds indicés de 0 à p, et un ensemble A contenant un arc (i,j), $i < j$, si une tournée desservant les clients de S_{i+1} à S_j (inclus) est réalisable en terme de capacité. Le poids z_{ij} de (i,j) est égal au coût de la tournée allant de i à j. Le découpage optimal de la séquence des p clients correspond à un chemin de coût minimum dans H, allant du nœud 0 au nœud p, comme le montre la figure 6.3. Les affectations des clients aux dépôts données par le chromosome grâce à DS sont représentées sous forme de séquences de clients sans délimitation de tournées. La procédure Split peut donc être appliquée à chaque séquence dédiée à un dépôt particulier afin de retrouver la solution réalisable correspondante ainsi que le coût de routage. De plus, Split étant une procédure exacte, si les séquences des clients affectés à chaque dépôt correspondent à un ordre permettant de fournir une solution globale optimale, lors de l'évaluation du chromosome, c'est bien cette solution qui sera restituée.

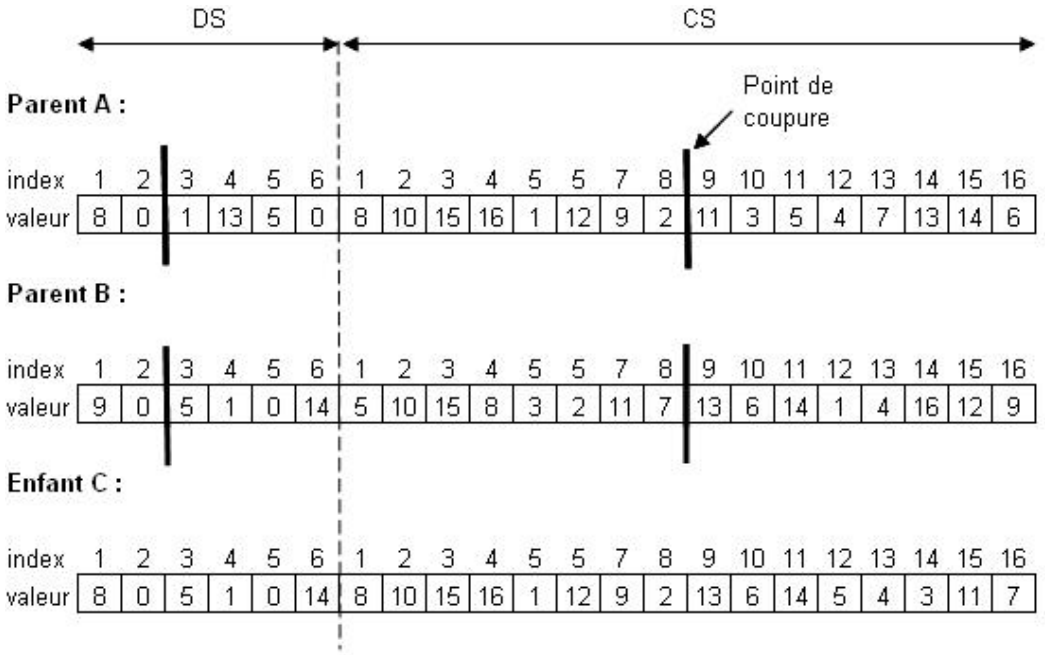


FIG. 6.4 – *Croisement de chromosomes*

6.3.2 Sélection des parents et croisement

Grâce au codage des individus (solutions) sous forme de chromosomes, il est possible de générer des enfants par croisement de deux parents de la population. Diverses techniques de croisement ont été proposées pour des chromosomes de permutation (Bean, 1994; Potvin, 1996; Drezner, 2003). La structure plus complexe de nos chromosomes nécessite la conception d'un croisement *ad hoc*.

La première étape est la sélection de deux parents. Ici, elle se fait par tournoi binaire pour chacun d'entre-eux. Pour cela, deux individus sont tirés au sort et celui de meilleur fitness est conservé. Le premier parent provient d'un tournoi effectué sur les β meilleurs individus de la population (selon le fitness). Le second est issu d'un tournoi sur la population complète sans le premier parent sélectionné.

Ensuite, pour deux parents A et B, le croisement a lieu en deux étapes, chacune utilisant un croisement à un point de coupure. La première étape concerne le vecteur DS. Le point de coupure est choisi aléatoirement, et le croisement s'effectue sur cette partie du chromosome comme sur une séquence binaire : le début du vecteur DS appartenant à A est d'abord copié dans le chromosome-enfant, puis après le point de coupure, c'est la fin du vecteur DS appartenant à B qui est copié dans l'enfant. Cette première étape est illustrée dans la partie gauche de la figure 6.4.

La seconde étape se fait par rapport à la partie CS. L'opérateur de croisement est alors adapté pour combiner deux permutations de clients. L'enfant C reçoit d'abord la sous-séquence de CS venant de A située avant la coupure. Ensuite, C est complété en se basant sur B

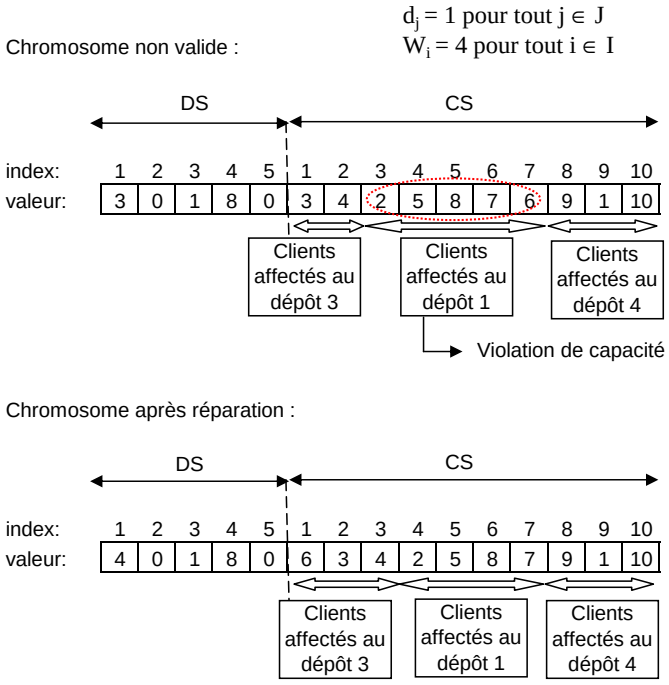


FIG. 6.5 – *Exemple de réparation*

parcouru à partir du point de coupure. Les clients ne pouvant être dupliqués, seuls ceux non encore recopiés dans l'enfant sont positionnés dans le vecteur CS de C, à l'indice (position) occupé dans B. Une fois arrivé au bout du vecteur CS du parent B, si tous les clients n'ont pas été copiés dans l'enfant, les positions vacantes dans CS sont complétées dans l'ordre rencontré en parcourant le début du vecteur CS de l'individu B. Une illustration du croisement sur CS est donnée dans la partie droite de la figure 6.4.

Ce type de croisement à un point sur chaque sous-partie du chromosome permet une transmission génétique à la fois fidèle aux parents (une seule coupure par sous-partie), et suffisamment diversifiante. En effet, le routage issu des individus A et B est perturbé à la fois par le point de coupure sur le vecteur CS, mais également lors de l'affectation au dépôt par l'héritage combiné des parents du vecteur DS. Ainsi, malgré la taille relativement grande du vecteur CS (permutation de tous les clients), une seule césure est suffisante pour obtenir une certaine diversité dans les enfants engendrés.

Par contre, cet opérateur de croisement peut générer des chromosomes correspondant à des solutions non-réalisables, en particulier à cause des contraintes de capacité sur les dépôts. Il est donc nécessaire de vérifier les solutions obtenues et de procéder à une réparation si besoin. La figure 6.5 donne un exemple de solution non-réalisable et de réparation d'un chromosome.

La première étape de réparation a pour but de vérifier que tous les clients sont bien affectés à un dépôt. Pour cela, il suffit de s'assurer que la valeur 1 apparaît dans le vecteur DS. Si

c'est le cas, cela signifie au pire qu'au moins un dépôt est ouvert et que toute la séquence des clients est affecté à ce dépôt. Sinon, les clients non desservis (situés au début du vecteur CS) sont affectés à un nouveau dépôt. Il suffit d'ouvrir le premier dépôt i encore fermé et de poser $DS(i) = 1$.

La deuxième étape de réparation concerne les contraintes de capacités sur les dépôts. Si un dépôt est en surcharge, il est possible de remédier au problème en lui retirant des clients. Pour cela, l'algorithme parcourt la séquence de clients desservis par ce dépôt en partant du dernier et les déplace un par un, jusqu'à ce que la contrainte concernée soit respectée. Les clients alors retirés sont ré-affectés au premier dépôt ouvert avec suffisamment de capacité pour le servir. Si aucun n'est trouvé, alors un nouveau site est ouvert.

Dans la méthode développée ici, la reproduction n'est pas générationnelle mais incrémentale. L'enfant obtenu, s'il est accepté par la procédure de gestion de la population décrite dans la section 6.3.4, remplace le moins bon chromosome de la population (celui de coût maximal).

6.3.3 Recherches locales

Une des particularités du *MA|PM* est de travailler sur une petite population de solutions de grande qualité (Sörensen et Sevaux, 2006). Pour cela, les solutions générées subissent une recherche locale. Dans le cadre du *LRP*, c'est la recherche locale *LS1*, décrite dans le chapitre 4, qui est appliquée. *LS1* étant relativement performante, un appel systématique conduirait à une convergence prématurée de la population et a des temps de calcul excessifs (dans tout algorithme mémétique, l'essentiel du temps de calcul est dépensé dans les recherches locales). En pratique, *LS1* est donc appliquée aux enfants avec une probabilité fixe p_1. Si *LS1* n'est pas utilisée, une version *allégée*, *LS1b*, est exécutée avec une probabilité p_2. Il est possible qu'un enfant échappe alors à la recherche locale, ceci arrivant dans $(1-p_1) \times (1-p_2)$ des cas. *LS1b* est en fait identique à *LS1*, sauf que les mouvements autorisés ne se font qu'entre clients affectés à un même dépôt. Son appel étant moins fréquent que *LS1*, cela permet de maintenir une population de bonne qualité en des temps de calcul plus raisonnables.

Aussi bien *LS1* que *LS1b* exécutent les premiers mouvements améliorant rencontrés, non les meilleurs, et s'arrêtent quand de tels mouvements ne sont plus détectés. Ces recherches locales permettent une amélioration du routage, mais ne permettent en aucun cas l'ouverture de dépôts, opération exclusivement réservée à la procédure de croisement.

6.3.4 Gestion de la population

La principale caractéristique du *MA|PM* vient de la gestion de la population grâce à une mesure de distance dans l'espace des solutions. Cette opération permet de contrôler la diversité des individus en filtrant l'entrée des enfants dans la population.

Pour cela, nous utilisons la même mesure de dissimilarité que celle décrite au paragraphe 5.5, dans le but de créer l'ensemble de solutions élites utilisées dans le Path Relinking en post-optimisation du *GRASP*. La dissimilarité d'une solution T à la population courante P est définie par :

$$d_P(T) = \min_{U \in P} D_2(U,T)$$

Ainsi, une nouvelle solution T ne peut intégrer la population courante que si $d_P(T) \geq \Delta$, Δ étant un seuil donné, sinon elle est tout simplement rejetée.

Pour trouver un juste équilibre entre $\Delta = 0$, auquel cas, l'algorithme adopte un comportement de type *MA* classique sans gestion de la population, et une grande valeur pour laquelle et où la majorité des solutions enfants est rejetée. Δ est ajusté dynamiquement selon la quatrième stratégie proposée par Sörensen et Sevaux (2006) (voir paragraphe 6.2.2). Après génération de la population de départ, Δ est donc initialisé à une relativement grande valeur Δ_{max}. Puis, si une série de *MaxNbRej* itérations successives avec rejet d'enfants est atteinte, Δ est alors décrémenté de 1, sans toutefois descendre en dessous de 1, afin d'interdire les clônes. Ensuite, Δ reste à une valeur constante tant qu'une nouvelle série de rejets successifs n'est pas atteinte. Par contre, si un enfant améliore la meilleure solution courante, alors le seuil Δ est rétabli à sa valeur initiale Δ_{max}. Cette stratégie de gestion de la population est utilisée pour la première fois dans une application du $MA|PM$.

6.4 Algorithme général

L'algorithme débute par la création d'une population de *NbIndiv* individus de bonne qualité, en utilisant deux heuristiques randomM

Algorithme 9 Algorithme $MA|PM$ pour le LRP

```
1: BestCost = +∞  ;  NbAcc = NbRej = NbNoAdd = 0
2: Δ = 1
3: GenPop(P)
4: Δ = Δ_max
5: //boucle principale sur les rafraîchissements de P
6: répéter
7:   //seconde boucle sur l'évolution de la population entre 2 rafraîchissements
8:   répéter
9:     Selection(A,B)
10:    Crossover(A,B,C)
11:    si (random < p1) alors
12:      LS1(C)
13:    sinon
14:      si (random < p2) alors
15:        LS1b(C)
16:      fin si
17:    fin si
18:    si (cout(C) < BestCost) alors
19:      Δ = Δ_max
20:      BestCost = cout(C)  ;  BestSol = C
21:      NbRej= 0  ;  NbAcc = NbAcc +1
22:      AddInPop(C)
23:    sinon
24:      si (d_P(C) < Δ) alors
25:        NbRej = NbRej + 1  ;  NbNoAdd = NbNoAdd + 1
26:      sinon
27:        NbRej= 0  ;  NbAcc = NbAcc +1
28:        AddInPop(C)
29:      fin si
30:    fin si
31:    si (NbRej > MaxNbRej) alors
32:      Δ = Max(1,Δ − 1)
33:    fin si
34:  jusqu'à (NbNoAdd > MaxNbNoAdd ou NbAcc > MaxNbAcc)
35:  Δ = Δ_max  ;  NbNoAdd = 0
36:  Refresh(P,BestSol)
37: jusqu'à (NbAcc > MaxNbAcc)
38: Retourne (BestSol)
```

Renvoyer retourne la meilleure solution trouvée. *LS1* et *LS1b* correspondent aux recherches locales.

6.5 Évaluation Numérique

6.5.1 Implémentation et instances

L'algorithme est codé en C et exécuté sur un PC Dell Optiplex GX260, avec un Pentium 4 à 2.4 GHz, 512 Mo de RAM et Windows XP. $MA|PM$ pour le LRP est évalué, comme pour les méthodes précédentes, sur les trois ensembles d'instances décrites en Annexe et comparé avec $GRASP$, ce dernier désignant la métaheuristique de type $GRASP$ avec PR décrite dans la section 5.

Les résultats sont donnés dans les tableaux 6.2 à 6.4. Les colonnes n et m indiquent respectivement le nombre de clients et de dépôts de l'instance. β est le nombre de clusters, *type* est soit relatif à la capacité des tournées (instances *I1* et *I3*), soit correspond au ratio des clients appartenant à des clusters sur n (instances $I2$). Les colonnes $Coût$ et Tps donnent le coût des solutions obtenues et les temps de calcul nécessaire en secondes pour les obtenir. *dep* et *tr* indiquent le nombre de dépôts et de tournées ouvertes dans la solution restituée. Enfin, la colonne E_{GRASP} est l'écart relatif en pourcentage entre le coût de la solution trouvée par le $MA|PM$ et le $GRASP$. ($E_{GRASP} = ((Cout(MA \mid PM) - Cout(GRASP))/Cout(GRASP)) * 100$). Mais tout d'abord, le paramétrage de l'algorithme utilisé pour obtenir ces résultats est donné au paragraphe suivant.

6.5.2 Paramétrage de l'algorithme

La taille de la population dans un $MA|PM$ doit être relativement restreinte selon Sörensen et Sevaux (2006). Cependant, afin de bien explorer l'espace des solutions, il est intéressant de la lier à la taille de l'instance. La même remarque est valable concernant le nombre d'itérations total à réaliser, ou pour déterminer les paramètres permettant de contrôler la diversité des individus.

Lors de la sélection des parents, selon notre algorithme, au moins un doit appartenir aux meilleures solutions de la population. Le critère déterminant est de figurer parmi les β individus de moindre coût dans la population courante. β dépend de la taille de la population.

Afin d'avoir des individus de bon fitness, une recherche locale est nécessaire au $MA|PM$. Il faut cependant éviter les rejets d'enfants pour cause de manque de diversité et ne pas alourdir les temps de calcul. Pour cela, deux types de recherche locale sont ici appliquées, chacune associée à une probabilité d'utilisation.

Après une phase de tests préliminaires sur les instances, les paramètres utilisés dans cette métaheuristique sont fixés aux valeurs données dans le tableau 6.1.

Parametres	Valeur	Parametres	Valeur
NbIndiv	$\lceil((n+m)/5)+1\rceil$	β	$NbIndiv \cdot 1/3$
MaxNbAcc	$(n+m)\cdot 10$	Δ_{max}	$\lceil(n+m)/10)+10\rceil$
MaxNbNoAdd	$\lceil(n+m)\cdot 2.5\rceil$	p_1	0.5
MaxNbRej	*NbIndiv*	p_2	0.7

TAB. 6.1 – *Paramètres du MA|PM pour le LRP*

6.5.3 Résultats pour le premier groupe d'instances - *I1*

Le tableau 6.2 montre que sur le premier ensemble d'instances, les solutions de la méta-heuristique *MA|PM* proposée ici améliorent de 2.06% celles du *GRASP* décrit dans la section 5. Le gain peut dépasser les 5% sur les grosses instances, avec un pic à plus de 11%. Pourtant, en regardant les dépôts ouverts, chacune des méthodes comparées utilise exactement le même nombre de sites. Par contre, *MA|PM* tend à utiliser un véhicule de moins sur certaines instances à partir de 100 clients. Néanmoins, cela ne suffit pas à justifier les gains réalisés en appliquant *MA|PM*. Une fois encore, le sous-ensemble de dépôts ouvert, en localisation et non en nombre, semble jouer un rôle important. Le *MA|PM* utilise des techniques de contrôle dynamique de la diversification avec le seuil Δ d'acceptation des solutions. De plus, il travaille non pas sur une solution, mais sur une population de bonnes solutions. Ces caractéristiques en font une méthode très efficace apportant des performances appréciables par rapport au *GRASP*, même si ce dernier trouve 6 des 30 meilleures solutions (dont 4 en commun avec *MA|PM*) sur cet ensemble d'instances.

L'amélioration de la qualité des solutions se fait en des temps de calcul inférieurs à ceux de la méthode *GRASP*, avec une moyenne de 76.7 s au lieu de 96.5 s. Néanmoins, il faut rappeler que sur les instances *I1*, l'algorithme complet du *GRASP* proposé passe 2/3 de son temps dans la post-optimisation qui ne rapporte qu'un gain de 0.3%. Avec une version sans *PR*, l'écart relatif de *MA|PM* par rapport au *GRASP* passerait à 2.4% mais la méthode à population serait cette fois-ci presque 3 fois plus lente.

6.5.4 Résultats pour le deuxième groupe d'instances - *I2*

Sur *I2*, *MA|PM* est encore une fois la méthode offrant les coûts les plus faibles comme le montre le tableau 6.3. Le gain est en moyenne de 1.56%. Cette déviation est du même ordre de grandeur que sur les instances *I1*, par contre, l'amélioration la plus forte ne dépasse pas ici les 6% et le *GRASP* donne de meilleures solutions que *MA|PM* pour 6 instances sur 36. Encore une fois, *MA|PM* tend à utiliser moins de véhicules, mais également ici moins de dépôts.

Les temps de calcul sont similaires à ceux du *GRASP*, mais cette fois légèrement supérieurs alors que, tout comme sur les instances *I1*, la suppression du *PR* dans l'algorithme du *GRASP* permettrait une diminution de moitié des temps de calcul. Cependant, il faut rappeler que la post-optimisation sur les instances *I2* permet une réduction non négligeable des coûts de 1% environ.

				GRASP				*MAPM*				
n	m	β	type	Coût	Tps	dep	tr	Coût	Tps	dep	tr	E_{GRASP}
20	5	0	a	55021	0.2	3	5	54793	0.3	3	5	-0.41
20	5	0	b	39104	0.2	2	3	39104	0.3	2	3	0.00
20	5	2	a	48908	0.1	3	5	48908	0.4	3	5	0.00
20	5	2	b	37542	0.2	2	3	37542	0.3	2	3	0.00
50	5	0	a	90632	1.8	3	12	90160	2.6	3	12	-0.52
50	5	0	b	64761	1.8	2	6	63242	3.2	2	6	-2.35
50	5	2	a	88786	2.4	3	12	88298	3.4	3	12	-0.55
50	5	2	b	68042	2.5	3	6	67893	2.9	3	6	-0.22
50	5	2'	a	84055	1.7	3	12	84055	3.2	3	12	0.00
50	5	2'	b	52059	2.6	3	6	51822	4.2	3	6	-0.46
50	5	3	a	87380	2.3	2	12	86203	3.1	2	12	-1.35
50	5	3	b	61890	2.0	2	6	61830	4.9	2	6	-0.10
100	5	0	a	279437	27.6	3	24	281944	26.3	3	24	0.90
100	5	0	b	216159	23.2	3	12	216656	34.5	3	12	0.23
100	5	2	a	199520	17.4	2	24	195568	35.8	2	24	-1.98
100	5	2	b	159550	22.4	2	11	157325	36.4	2	11	-1.39
100	5	3	a	203999	21.6	2	25	201749	28.7	2	24	-1.10
100	5	3	b	154596	20.3	2	11	153322	33.3	2	11	-0.82
100	10	0	a	323171	37.4	4	26	316575	24.7	4	25	-2.04
100	10	0	b	271477	29.5	4	12	270251	36.0	4	11	-0.45
100	10	2	a	254087	39.1	3	25	245123	24.6	3	24	-3.53
100	10	2	b	206555	29.8	3	11	205052	31.6	3	11	-0.73
100	10	3	a	270826	35.4	3	25	253669	29.0	3	24	-6.34
100	10	3	b	216173	39.8	3	11	204815	36.5	3	11	-5.25
200	10	0	a	490820	517.5	3	48	483497	345.1	3	47	-1.49
200	10	0	b	416753	379.1	3	22	380044	463.0	3	22	-8.81
200	10	2	a	512679	554.3	3	49	451840	280.6	3	48	-11.87
200	10	2	b	379980	367.4	3	23	375019	321.0	3	22	-1.31
200	10	3	a	496694	424.8	3	46	478132	212.9	3	46	-3.74
200	10	3	b	389016	290.2	3	22	364834	272.0	3	22	-6.22
	Moyenne				96.5	2.8	17.2		76.7	2.8	16.9	-2.06

NB : un écart négatif indique une amélioration par rapport à la solution prise en référence.

TAB. 6.2 – *Résultats du MAPM - Instances I1 avec capacité sur les dépôts*

				GRASP				*MAPM*				
n	m	β	type	Coût	Tps	dep	tr	Coût	Tps	dep	tr	E_{GRASP}
100	10	0	0.75	1525.25	32.4	3	11	1493.92	31.5	3	11	-2.05
100	20	0	0.75	1526.90	40.7	3	11	1471.36	35.6	2	11	-3.64
100	10	0	1	1423.54	27.6	2	11	1418.83	36.2	3	10	-0.33
100	20	0	1	1482.29	36.2	2	11	1492.46	36.4	2	11	0.69
100	10	3	0.75	1200.24	27.7	2	11	1173.22	31.9	2	11	-2.25
100	20	3	0.75	1123.64	34.3	3	11	1115.37	42.7	3	11	-0.74
100	10	3	1	814.00	22.5	3	12	793.97	38.0	2	11	-2.46
100	20	3	1	747.84	37.3	3	11	730.51	49.3	2	11	-2.32
100	10	5	0.75	1273.10	21.5	3	11	1262.32	36.8	3	11	-0.85
100	20	5	0.75	1272.94	36.0	2	11	1251.32	47.7	3	11	-1.70
100	10	5	1	912.19	20.3	3	12	903.82	35.1	3	12	-0.92
100	20	5	1	1022.51	38.4	3	11	1022.93	62.6	4	11	0.04
150	10	0	0.75	2006.70	113.0	3	16	1959.39	128.5	3	16	-2.36
150	20	0	0.75	1888.90	161.4	4	16	1881.67	144.2	3	16	-0.38
150	10	0	1	2033.93	100.0	3	17	1984.25	111.0	3	16	-2.44
150	20	0	1	1856.07	132.4	4	16	1855.25	144.1	3	16	-0.04
150	10	3	0.75	1508.33	117.7	3	16	1448.27	166.6	2	16	-3.98
150	20	3	0.75	1456.82	166.1	2	16	1459.83	154.8	2	16	0.21
150	10	3	1	1240.40	106.7	2	16	1207.41	161.4	3	17	-2.66
150	20	3	1	940.80	142.4	3	17	934.79	196.1	3	17	-0.64
150	10	5	0.75	1736.90	92.8	3	17	1720.30	143.8	3	16	-0.96
150	20	5	0.75	1425.74	128.4	3	16	1429.34	155.7	4	16	0.25
150	10	5	1	1223.70	88.5	3	17	1203.44	153.8	3	17	-1.66
150	20	5	1	1231.33	134.9	4	17	1158.54	223.0	3	17	-5.91
200	10	0	0.75	2384.01	308.0	3	21	2293.99	418.3	3	21	-3.78
200	20	0	0.75	2288.09	410.0	4	22	2277.39	458.4	3	21	-0.47
200	10	0	1	2273.19	311.4	3	21	2274.57	376.8	3	21	0.06
200	20	0	1	2345.10	418.9	3	22	2376.25	436.3	3	21	1.33
200	10	3	0.75	2137.08	338.0	3	22	2106.26	350.5	3	21	-1.44
200	20	3	0.75	1807.29	370.0	4	21	1771.53	377.0	2	20	-1.98
200	10	3	1	1496.75	242.7	2	21	1467.54	322.2	2	21	-1.95
200	20	3	1	1095.92	308.5	3	22	1088.00	505.0	3	22	-0.72
200	10	5	0.75	2044.66	282.8	4	23	1973.28	412.8	4	22	-3.49
200	20	5	0.75	2090.95	399.2	4	22	1979.05	406.1	5	21	-5.35
200	10	5	1	1788.70	199.0	2	21	1782.23	352.8	3	22	-0.36
200	20	5	1	1408.63	296.3	5	22	1396.24	529.8	5	22	-0.88
Moyenne					159.6	3.0	16.4		203.1	2.9	16.2	-1.56

NB : un écart négatif indique une amélioration par rapport à la solution prise en référence.

TAB. 6.3 – *Résultats du MAPM - Instances I2 sans capacité sur les dépôts*

			GRASP				*MAPM*				
n	m	type	Coût	Tps	dep	tr	Coût	Tps	dep	tr	E_{GRASP}
21	5	6000	429.6	0.2	2	5	424.9	0.3	2	4	-1.08
22	5	4500	585.1	0.2	1	3	611.8	0.3	2	3	4.56
27	5	2500	3062.0	0.4	2	4	3062.0	1.0	2	4	0.00
29	5	4500	515.1	0.4	2	4	512.1	0.8	2	4	-0.59
32	5	8000	571.9	0.6	2	4	571.9	0.8	2	4	0.00
32	5	11000	504.3	0.5	1	3	534.7	1.0	2	3	6.02
36	5	250	460.4	0.8	1	4	485.4	1.4	2	4	5.44
50	5	160	599.1	2.3	3	8	565.6	3.8	2	6	-5.60
75	10	140	861.6	9.8	3	9	866.1	9.4	3	9	0.52
88	8	9000000	356.9	17.3	2	8	355.8	34.2	2	7	-0.30
100	10	200	861.6	25.5	3	9	850.1	44.5	2	8	-1.33
134	8	850	5965.1	49.6	4	11	5950.1	110.5	4	10	-0.25
150	10	8000000	44625.2	156.0	3	13	44011.7	255.0	3	11	-1.37
Moyenne				20.3	2.2	6.5		35.6	2.3	5.9	0.46

NB : un écart négatif indique une amélioration par rapport à la solution prise en référence.

TAB. 6.4 – *Résultats du MAPM - Instances I3 avec capacité sur les dépôts*

6.5.5 Résultats pour le troisième groupe d'instances - *I3*

Le tableau 6.4 montre que le *GRASP* avec *PR* est cette fois-ci plus performant que le *MA|PM*, et obtient des résultats de coût inférieur en moyenne (gain de 0.46%). En regardant plus en détails, le *GRASP* ne trouve en fait que 6 des meilleures solutions sur 13, plus particulièrement sur les petites instances. D'ailleurs, sur l'ensemble *I1*, composé de petites et grandes instances, les gains les plus remarquables obtenu par le *MA|PM* apparaissent quand $n = 200$.

Sur l'ensemble *I3*, quand *MA|PM* obtient des coûts moins important que le *GRASP*, souvent le nombre de tournées ouvertes, et parfois le nombre de dépôts, sont alors inférieurs par rapport au *GRASP*. Pour ce qui est des temps de calcul, les conclusions sont similaires à celles pour les instances *I1* et *I2*.

6.6 Particularités de l'algorithme

6.6.1 Intérêt général

Les bons résultats obtenus par *MA|PM* sont dus en partie au fait que même si les temps de calcul sont similaires à ceux du *GRASP*, le nombre de solutions générées est plus important. De plus, en dehors de la sélection des parents, le croisement permet une intervention *aveugle* sur la solution. C'est le critère d'acceptation lié à la gestion de la population et la qualité de la solution qui va guider les choix et non, comme dans le *GRASP*, des critères gloutons. Ceci fait que l'algorithme *MA|PM* est plus agressif et accède à des solutions diverses et de grande qualité, souvent non visitées par le *GRASP*, lui donnant ainsi plus de chances d'atteindre des solutions de coûts inférieurs. Enfin, *MA|PM* est une méthode à population. Cette dernière évolue et la qualité moyenne des individus qui la compose augmente progressivement. Cette

L'autre avantage du $MA|PM$ est sa simplicité d'implémentation. A partir d'un algorithme génétique de base, l'ajout des composants pour en faire un algorithme mémétique puis un $MA|PM$ est relativement facile. Le point délicat est de trouver une bonne représentation des solutions sous forme de chromosomes. Ensuite, il suffit de disposer d'une recherche locale et d'une mesure de distance pour passer d'une version classique à un $MA|PM$.

6.6.2 Impact du générateur de nombres aléatoires

n	m	$MAPM$	$MAPM \times 10$	Dev	E_{MAPM}
Premier ensemble - $I1$:					
20	5	45087	45092	0.01	0.01
50	5	74188	74294	0.06	0.11
100	5	201094	200413	0.09	-0.25
100	10	249248	250285	0.24	0.44
200	10	422228	423557	0.35	0.34
Moyenne				0.15	0.14
Deuxième ensemble - $I2$:					
100	10	1174.34	1174.57	0.25	0.19
100	20	1180.66	1178.33	0.19	-0.19
150	10	1587.18	1596.27	0.23	0.58
150	20	1453.24	1460.07	0.32	0.43
200	10	1982.98	1994.03	0.25	0.54
200	20	1814.74	1813.87	0.28	-0.06
Moyenne				0.25	0.25
Troisième ensemble - $I3$:					
≤ 50	5	846.06	846.41	0.03	0.06
≤ 150	≤ 10	10406.76	10462.19	0.33	-0.18
Moyenne				0.15	-0.03
Moyenne				**0.16**	**0.20**

NB : un écart négatif indique une amélioration.

TAB. 6.5 – *Impact du générateur de nombres aléatoires*

Grâce à l'utilisation fréquente d'une recherche locale pour obtenir des solutions de bonne qualité, les enfants générés sont ramenés à des solutions localement optimales. Ainsi, les résultats des algorithmes de type mémétique sont plus robustes que les GA classiques : ils sont moins affectés par le type de générateur de nombres aléatoires utilisé pour la population initiale, la sélection des parents et les croisements. Pour s'en convaincre, il peut être intéressant d'effectuer plusieurs lancements de l'algorithme et de rapporter la valeur moyenne obtenue. Le tableau 6.5 montre de tels résultats. $MA|PM \times 10$ correspond à 10 exécutions indépendantes de la méthode. Les colonnes $MA|PM$ ou $MA|PM \times 10$ et Tps donnent le coût moyen des solutions et les temps de calcul moyens en secondes par sous-ensemble d'instances. La colonne E_{MAPM} donne l'écart relatif en coût entre 1 et 10 exécutions, et la colonne Dev, l'écart relatif (en valeur absolue) moyen sur les 10 exécutions avec la valeur donnée dans la colonne $MA|PM \times 10$. On peut voir que les résultats restent très stables sur plusieurs lancements. En effet, en moyenne, la déviation est de 0.16%, confirmant le côté robuste de la méthode par rapport à l'utilisation de paramètres aléatoires. Par rapport à la moyenne des coûts des solutions sur 10 exécutions, les résultats particuliers donnés comme performance de la méthode, et utilisés pour la comparaison avec le GRASP, sont 0.20% meilleurs. Cependant, même en prenant les valeurs moyennes, $MA|PM$ donne toujours des coûts de plus faible valeur.

6.6.3 Impact de la gestion de la population

		$MAPM$		MA		
n	m	Coût	Tps	Coût	Tps	E_{MAPM}
Premier ensemble - $I1$:						
20	5	45087	0.3	45087	0.2	0.00
50	5	74188	3.4	74194	2.6	0.00
100	5	201094	32.5	200995	24.4	-0.04
100	10	249248	30.4	251579	47.2	0.93
200	10	422228	315.8	424279	265.3	0.54
Moyenne			76.7		68.1	0.29
Deuxième ensemble - $I2$:						
100	10	1174.34	34.9	1183.41	28.5	0.85
100	20	1180.66	45.7	1184.52	35.3	0.29
150	10	1587.18	144.2	1588.79	107.5	0.10
150	20	1453.24	169.6	1445.18	134.4	-0.47
200	10	1982.98	372.2	1996.37	304.4	0.62
200	20	1814.74	452.1	1816.67	344.0	0.03
Moyenne			203.1		159.0	0.24
Troisième ensemble - $I3$:						
≤ 50	5	846.06	1.2	847.02	0.9	0.18
≤ 150	≤ 10	10406.76	90.7	10439.84	70.4	-0.44
Moyenne			35.6		27.6	-0.06
Moyenne			**127.6**		**102.9**	**0.21**

NB : un écart négatif indique une amélioration par rapport à la solution.

TAB. 6.6 – *Impact de la gestion de la population*

La gestion de la population (PM) permet de conserver une certaine diversité des solutions et ainsi donne l'accès, par la combinaison de parents distants, à de nouvelles solutions rarement accessibles par des critères gloutons tels que ceux utilisés dans le *GRASP*. Le tableau 6.6 montre les répercussions du PM sur le comportement de l'algorithme. $MA|PM$ est la version complète, la colonne MA est une version dans laquelle le critère de distance utilisé est $\Delta = 1$, écartant ainsi seulement les clônes (solutions identiques) de la population. Aucune autre diversification des solutions n'est contrôlée. Les colonnes *Coût* et *Tps* donnent le coût moyen des solutions et les temps de calcul moyen en secondes par sous-ensemble d'instances. La colonne E_{MAPM} donne l'écart relatif en coût de MA avec le $MA|PM$.

Il est possible que le PM n'apporte pas de meilleures solutions comme on peut le voir dans le tableau lorsque l'écart relatif entre MA et $MA|PM$, est négatif. Néanmoins, en moyenne, les solutions obtenues sans gestion de la population sont de coût supérieur (0.21%).

Par contre, le rejet de solutions dans la version complète ($MA|PM$) est plus fréquent et les temps de calcul en sont affectés. Sur les trois ensembles d'instances *I1*, *I2* et *I3*, le temps moyen passé sur $MA|PM$ est de 127 s alors qu'il est de 102 s pour MA. Néanmoins, ce surplus de temps n'est pas excessif et il est intéressant d'appliquer cette gestion des individus générés puisque le type de décisions prises implique en principe des coûts importants et que le PM est une technique simple qui permet d'accroître en moyenne la qualité des solutions obtenues.

6.7 Conclusion

Dans ce chapitre, une nouvelle métaheuristique pour le problème de localisation-routage avec des véhicules et des dépôts de capacité limitée est présentée. C'est une méthode évolutionnaire appelée algorithme mémétique avec gestion de la population (*Memetic Algorithm with Population Management* - *MA|PM*). Il s'agit d'un algorithme génétique hybridisée avec des techniques de recherche locale et de mesure de distance permettant de contrôler la diversification des solutions dans la population. La méthode, testée sur les trois types d'instances *I1*, *I2* et *I3*, est comparée au *GRASP* présenté précédemment au chapitre 5. Les solutions obtenues montrent que cet algorithme est très performant car il permet d'obtenir d'excellents résultats (gain moyen sur les trois jeux d'instances de 1.42% par rapport au *GRASP*) en des temps de calcul raisonnables, similaires à ceux nécessaires pour le *GRASP*. De plus, son implémentation est relativement facile si un codage adéquat des solutions en chromosome est disponible.

La version du *MA|PM* proposée dans cette partie a fait l'objet d'une présentation dans un congrès avec actes publiés dans la série *Lecture Notes in Computer Science* (Springer) (Prins *et al.*, 2006a). Une des pistes de recherche pour l'améliorer pourrait être l'inclusion de techniques d'apprentissage permettant de guider la méthode sur un bon sous-ensemble de dépôts, comme c'est le cas pour le *GRASP*, et ainsi de réduire le nombre d'itérations et donc les temps de calcul.

Les deux métaheuristiques proposées jusqu'à maintenant sont basées sur des concepts génériques. Les résultats semblent très bons, en particulier ceux obtenus par le *MA|PM* qui est une approche plus agressive que le *GRASP*. Par contre, une autre idée est d'essayer d'exploiter de manière plus approfondie les spécificités du problème, afin de guider au mieux la localisation et le routage, sans dissocier ces deux étapes. C'est l'objectif visé dans le chapitre suivant avec une approche coopérative.

Chapitre 7

Approche Coopérative

7.1 Introduction

Jusqu'à maintenant, nous avons vu que le problème de localisation-routage est un problème intégrant différents niveaux de décision, nécessitant des méthodes de résolution globale afin d'obtenir des solutions de bonne qualité, et approchées pour avoir des temps de calcul raisonnables pour des instances de grande taille.

Les heuristiques constructives manquent de puissance pour obtenir de bons résultats car elles n'ont pas une vision assez large du problème pour orienter la recherche. Les métaheuristiques *classiques* apportent une réponse plus satisfaisante. Cependant, dans les deux chapitres précédents, nous avons renforcé un *GRASP* (avec un apprentissage et un path relinking) et un algorithme mémétique (avec une gestion de la population), pour en faire des méthodes plus agressives.

Le but de ce chapitre est de développer une approche encore plus pertinente, tirant parti des connaissances du problème. Pour cela, une décomposition non hiérarchique est proposée, apportant une certaine synergie basée sur l'exploitation des spécificités du *LRP*. Des échanges d'informations permettent une coopération entre des phases dédiées à la localisation et d'autres concernant l'élaboration de tournées de véhicules. De plus, cette décomposition permet une réduction de la taille des problèmes à résoudre.

Ce type de méthode est appelé ici *coopérative*. Le principe est d'aborder alternativement les deux principaux sous-problèmes à l'aide de techniques performantes individuellement, tout en échangeant des informations judicieuses afin d'exploiter les relations des deux phases.

La localisation utilise la configuration des tournées pour déduire un sous-ensemble de dépôts à utiliser en adéquation. Pour cela, le choix des sites est basé sur une formulation simplifiée du problème de départ : les clients d'une même tournée sont tous agrégés en un *super-client* qu'il faut affecter à un dépôt. Le problème de localisation ainsi obtenu subit une relaxation lagrangienne des contraintes d'affectation. Une optimisation par sous-gradients et le recours à une heuristique lagrangienne permettent alors de déterminer un bon sous-ensemble de dépôts à ouvrir selon les super-clients (et donc les tournées) de départ.

La seconde phase est basée sur le sous-ensemble de sites issu de la phase de localisation et améliore les tournées en fonction de la nouvelle structure du réseau de dépôts ouverts. Ceci s'effectue par une heuristique de recherche de type tabou granulaire (*Granular Tabu Search - GTS*).

Ce schéma de résolution utilise un transfert naturel d'informations. Malheureusement, la méthode peut converger prématurément vers un optimum local. Une perturbation de la solution est alors nécessaire : de nouvelles tournées sont reconstruites, basées sur un compteur de fréquences d'utilisation des arêtes reliant les clients. Cette technique peut être vue comme un branchement dans un arbre de recherche sur les arêtes les plus prometteuses.

Ce chapitre se décompose comme suit. La phase de localisation, résolue à l'aide d'une relaxation lagrangienne, est présentée en section 7.2, suivie par la section 7.3 où est décrite la phase de routage utilisant un *GTS*. La section 7.4 développe la démarche coopérative liant chacune des deux phases. Les performances de la méthode sont données avec les résultats expérimentaux en section 7.5. La section 7.6 discute de l'impact des composants et le chapitre se termine par quelques remarques de conclusion en section 7.7.

7.2 Phase de localisation basée sur une relaxation lagrangienne

En considérant un service des clients par camion dédié (*truckload*), le *LRP* devient un problème de localisation d'entrepôts comme décrit au paragraphe 2.3.3 du chapitre 2. Dans notre cas, cette hypothèse n'est pas valide. Cependant, en supposant que les tournées, non reliées à des dépôts mais visitant l'ensemble des clients, soient connues, il suffit alors de définir le coût d'affectation de ces tournées à chaque dépôt pour se ramener à ce problème de localisation. C'est sur cette observation qu'est basée l'approche développée dans cette section.

7.2.1 Problème de localisation de dépôts

Afin de disposer de tournées, la phase de localisation de la méthode de résolution proposée dans ce chapitre part d'une solution réalisable, issue soit d'une heuristique constructive (première itération), soit de la phase de routage précédente. De cette solution sont extraits ce que nous appellerons des *super-clients*, représentant chacun une tournée k de demande totale équivalente D_k, et dans laquelle tous les clients sont agrégés (voir la figure 7.1).

La transformation des tournées en super-clients permet donc d'obtenir un problème de localisation de dépôts à capacité limitée et à source unique (*single-source*), dans lequel chaque super-client doit être servi à partir d'un unique dépôt. L'idée des super-clients n'est pas particulièrement nouvelle. Elle a été adoptée par Wu *et al.* (2002) par exemple. Elle est également utilisée dans les méthodes d'agrégation (*clustering*) en classification. Le problème ainsi obtenu a déjà été présenté au paragraphe 2.3.3 du chapitre 2 avec un état de l'art concernant les méthodes classiques de résolution. Une formulation utilisant les notations relatives à la relaxation du *LRP* est donnée ci-après.

Soit un ensemble de m localisations potentielles pour des dépôts, et un ensemble K de super-clients, l'objectif est de minimiser le coût total d'ouverture des dépôts i (O_i) et d'affec-

tation des super-clients k à ces derniers (C_{ik}), tout en respectant des contraintes de capacité. Pour obtenir les coûts d'affectation, on suppose que le dépôt actuel de la tournée représentée est enlevé, puis on calcul la meilleure insertion pour le dépôt i :

$$C_{ik} = min(C_{ij} + C_{il} - C_{jl} | j \text{ et } l \text{ consécutifs dans k })$$

Les variables de décisions X_{ik} sont égales à 1 ssi le super-client k est affecté au dépôt i. Les variables binaires y_i valent 1 ssi le dépôt i est ouvert. On obtient ainsi le modèle suivant :

$$\min Z = \sum_{i \in I} O_i y_i + \sum_{i \in I} \sum_{k \in K} C_{ik} X_{ik} \tag{7.1}$$

Sous les contraintes :

$$\sum_{i \in I} X_{ik} = 1 \qquad \forall k \in K \tag{7.2}$$

$$\sum_{k \in K} D_k X_{ik} \leq W_i y_i \qquad \forall i \in I \tag{7.3}$$

$$X_{ik} \in \{0,1\} \qquad \forall i \in I, \forall k \in K \tag{7.4}$$

$$y_i \in \{0,1\} \qquad \forall i \in I \tag{7.5}$$

(7.1) est la fonction-objectif, (7.2) assurent que les clients ne sont affectés qu'à un seul site, (7.3) correspondent au respect des capacités des dépôts, et (7.4) et (7.5) sont les contraintes d'intégrité des variables.

Les tournées étant représentées par les super-clients, les coûts fixes d'ouverture d'une tournée présents dans la fonction-objectif du modèle (LRP1) du chapitre 3 disparaissent, tout comme les contraintes spécifiques au routage : (3.3) pour les capacités, (3.4) et (3.6) concernant le fait que les tournées doivent être fermées et sans sous-cycle et (3.7) spécifiant qu'un client ne peut être affecté à un dépôt que si une tournée lie le client et le dépôt concerné.

7.2.2 Relaxation lagrangienne

Afin de résoudre le sous-problème de localisation obtenu, une relaxation lagrangienne est appliquée aux contraintes d'affectation (7.2). Le choix de ce type d'approche s'appuie sur les bons résultats qu'elle apporte selon la littérature sur le sujet (voir le paragraphe 2.3.3 du chapitre 2). Les relaxations lagrangiennes ont été largement exploitées en programmation entière pour obtenir de bonnes bornes inférieures, à partir desquelles des solutions réalisables peuvent être déduites de manière heuristique. Le principe de base est d'éliminer un ensemble de contraintes de la formulation initiale du problème et de les incorporer dans la fonction-objectif avec une pénalité donnée. Des précisions sur cette méthodes peuvent être trouvées dans les articles de Geoffrion (1974), Shapiro (1979), Fisher (1981) ou Beasley (1993b).

Ici, la relaxation porte sur les contraintes d'affectation comme proposé par Pirkul (1987) car Beasley (1993a), suivi de Ahlander (1994) dans sa thèse, ont confirmé qu'il s'agit de la meilleure relaxation possible pour ce problème de localisation.

Soit $u_k \in \mathbb{R}$ $(k \in K)$ les multiplicateurs lagrangiens associés à chaque contrainte (7.2). Le modèle relaxé obtenu pour des valeurs fixées de ces multiplicateurs est le suivant et fournit une borne inférieure au problème (7.1)-(7.5):

$$\min g(u) = \sum_{i \in I} O_i y_i + \sum_{i \in I} \sum_{k \in K} C_{ik} X_{ik} + \sum_{k \in K} u_k (1 - \sum_{i \in I} X_{ik}) \tag{7.6}$$

Sous les contraintes :

$$\sum_{k \in K} D_k X_{ik} \leq W_i y_i \qquad \forall i \in I \tag{7.7}$$

$$X_{ik} \in \{0,1\} \qquad \forall i \in I, \forall k \in K \tag{7.8}$$

$$y_i \in \{0,1\} \qquad \forall i \in I \tag{7.9}$$

En décomposant pour chaque dépôt i, il est possible d'obtenir m sous-problèmes indépendants qui se révèlent être, en posant $y_i = 1$, des problèmes de sac-à-dos (le terme constant $\sum_{k \in K} u_k$ de la fonction-objectif peut être retiré) :

$$\min \overline{g}_i(u) = O_i + \sum_{k \in K} (C_{ik} - u_k) X_{ik} \tag{7.10}$$

Sous les contraintes :

$$\sum_{k \in K} D_k X_{ik} \leq W_i \tag{7.11}$$

$$X_{ik} \in \{0,1\} \qquad \forall k \in K \tag{7.12}$$

Nous résolvons ces m problèmes par un algorithme classique de programmation dynamique (voir Pisinger (1997) par exemple). Afin de pouvoir utiliser cette même approche sur le LRP avec des dépôts sans capacité, il faut choisir une valeur fictive pour W_i inférieure à la demande totale des clients pour contraindre les problèmes de sac-à-dos. Ceci implique l'ouverture de deux dépôts minimum. Nous avons choisi $W_i = 1/2 \sum_{k \in K} D_k$.

Pour des valeurs données des u_k, la solution optimale ($\overline{y}_i$ et $\overline{X}_{ik}$) de (7.6)-(7.9) représentent une borne inférieure pour (7.1)-(7.5). Elle peut être déduite à partir des solutions optimales des m problèmes de sac-à-dos en considérant les valeurs de la fonction-objectif de (7.10)-(7.12) à l'optimum, $\overline{g}_i{}^*$. En effet, celles-ci peuvent être vues comme les coûts réduits de chaque dépôt i :

- si $\overline{g}_i{}^* < 0$, alors le dépôt i est ouvert ($\overline{y}_i = 1$) et les valeurs des $\overline{X}_{ik}$, $\forall k \in K$, sont celles trouvées par la résolution du problème de sac-à-dos pour le dépôt i (X_{ik}).
- sinon, le dépôt i est fermé, et $\overline{y}_i$ ainsi que les $\overline{X}_{ik}$ associées sont alors mises à zéro.

7.2.3 Optimisation par sous-gradients

La solution obtenue avec $\overline{y}_i$ et $\overline{X}_{ik}$ ne donne qu'une borne inférieure au problème d'origine de localisation puisque la relaxation autorise les super-clients à être affectés à plusieurs dépôts

en même temps, ce qui va à l'encontre des contraintes initiales. Une manière d'améliorer la borne inférieure ainsi obtenue est de jouer sur les valeurs données aux multiplicateurs. Pour cela, la méthode d'optimisation par sous-gradients est utilisée. Au début, il semble judicieux de favoriser les dépôts les plus proches des super-clients en choisissant les u_k de la manière suivante :

$$u_k = \min_i(C_{ik}) \qquad \forall k \in K \tag{7.13}$$

Ensuite, pour chaque itération, le problème (7.6)-(7.9) avec les valeurs courantes des u_k est résolu par la méthode décrite au paragraphe 7.2.2. Le résultat est optimal pour le problème d'origine de localisation si chaque super-client est affecté à un unique dépôt, ce qui signifie alors que la solution est réalisable. Sinon, c'est une borne inférieure (lb) qui est obtenue. Il est alors possible d'en déduire une borne supérieure (ub) en utilisant une heuristique lagrangienne rendant la solution réalisable par une procédure de réparation qui traite les super-clients desservis par plusieurs dépôts ou non affectés (voir section suivante).

Avec les bornes inférieures et supérieures ainsi obtenues, les multiplicateurs lagrangiens sont ajustés en utilisant l'équation (7.14), dans laquelle q désigne le numéro d'itération de la méthode de sous-gradient. Le pas $t^{(q)}$ est donné par (7.15). $\overline{ub}$ représente la meilleure borne supérieure connue. $\lambda^{(q)}$ est fixé initialement à 1.5, divisé par 2 toutes les 3 itérations consécutives sans amélioration et remis à sa valeur initiale quand il devient inférieur à un ε prédéterminé ou quand $\overline{ub}$ est améliorée. Le sous-gradient $d^{(q)}$ est un vecteur de m valeurs dans lequel $d_k^{(q)}$ est défini par (7.16).

$$u_k^{(q)} = u_k^{(q-1)} + t^{(q)} d_k^{(q)} \tag{7.14}$$

$$t^{(q)} = \lambda^{(q)} \frac{\overline{ub} - lb}{\|d^{(q)}\|} \tag{7.15}$$

$$d_k^{(q)} = 1 - \sum_{i \in I} X_{ik}^{lb} \qquad \forall k \in K \tag{7.16}$$

L'optimisation par sous-gradients continue jusqu'à ce qu'une convergence survienne ($lb = \overline{ub}$ ou $\|d^{(q)}\| = 0$), auquel cas les dépôts et l'affectation des super-clients sont directement donnés, ou quand un nombre prédéterminé d'itérations *LagrIt* est atteint. La solution alors renvoyée correspond à la meilleure borne supérieure trouvée au cours des itérations, $\overline{ub}$.

7.2.4 Heuristique lagrangienne

Il est possible d'obtenir une solution réalisable à partir des résultats obtenus par la relaxation lagrangienne. Celle-ci est alors une borne supérieure au problème (7.1)-(7.5), nécessaire pour l'optimisation par sous-gradients.

La relaxation lagrangienne respecte les contraintes de capacité des dépôts. Les affectations des super-clients obtenues en résolvant les problèmes de sac-à-dos vérifient donc aussi ces contraintes. Par contre, une attention particulière doit être portée aux super-clients non affectés. La procédure de réparation utilisée est inspirée de celle décrite par Cortinhal et Captivo (2003), mais se réalise de la manière expliquée ci-après.

Soit PD_k l'ensemble des dépôts auxquels le super-client k est affecté. Les premiers super-clients considérés sont ceux affectés à un unique dépôt ($|PD_k| = 1$). Ils fournissent un sous-ensemble initial de dépôts ouverts SOD (*Subset of Open Depots*), et leur affectation est gardée. Ensuite, ce sont les super-clients avec $|PD_k| > 1$ qui sont pris en compte. Pour un tel super-client k, les dépôts issus de $SOD \cap PD_k$ sont d'abord sélectionnés et le plus proche est finalement choisi pour le livrer. La capacité résiduelle de ce dépôt est alors ajustée comme il se doit. Si le super-client n'est affecté à aucun dépôt déjà ouvert, il est alors rattaché au site de $PD_k \backslash \{SOD \cap PD_k\}$ ayant le plus de super-clients affectés dans la solution du problème (7.6)-(7.9). Ce dépôt est alors ajouté à SOD. Enfin, l'algorithme traite un à un, dans l'ordre des demandes décroissantes, les super-clients avec $|PD_k| = 0$. Un tel super-client est affecté au plus proche dépôt de SOD ayant suffisamment de capacité pour le servir. Si aucun dépôt de SOD ne peut répondre à sa demande, alors un nouveau pouvant le servir est ouvert et ajouté à SOD : on choisit celui qui minimise la somme du coût d'ouverture et du coût d'affectation du super-client.

7.2.5 De la solution du problème de localisation de dépôts à la solution du *LRP*

Après la phase de localisation, la solution correspondante du LRP est extraite en reformant les tournées initiales contenues dans chaque super-client (voir une illustration dans la figure 7.1). L'ordre de visite des clients est conservé mais leur nouveau dépôt d'affectation est inséré à coût minimal entre deux clients consécutifs. La solution réalisable obtenue est alors transmise à la phase de routage, qui va tenter d'améliorer les tournées en fonction du nouveau sous-ensemble de dépôts ouverts.

7.3 Phase de routage par une recherche taboue granulaire

7.3.1 Principe général

Après une phase de localisation, seul le routage est considéré. Le problème à optimiser devient donc un VRP multi-dépôt. Il est résolu ici par une recherche taboue granulaire (GTS). A notre connaissance, le VRP classique est le seul problème de tournées auquel un GTS a été appliqué (Toth et Vigo, 2003). Cette méthode est un peu moins performante que les méthodes taboues classiques (en terme de résultat) mais elle est beaucoup plus rapide. Cette vélocité est intéressante pour réduire les temps de calcul dans notre méthode car la phase de routage va être exécutée plusieurs fois. De toute façon, soulignons que cette thèse propose pour la première fois d'utiliser un GTS pour le LRP et dans une approche coopérative.

Les recherches de type tabou (TS) sont présentées au chapitre 2 au paragraphe 2.2.3.3. Des

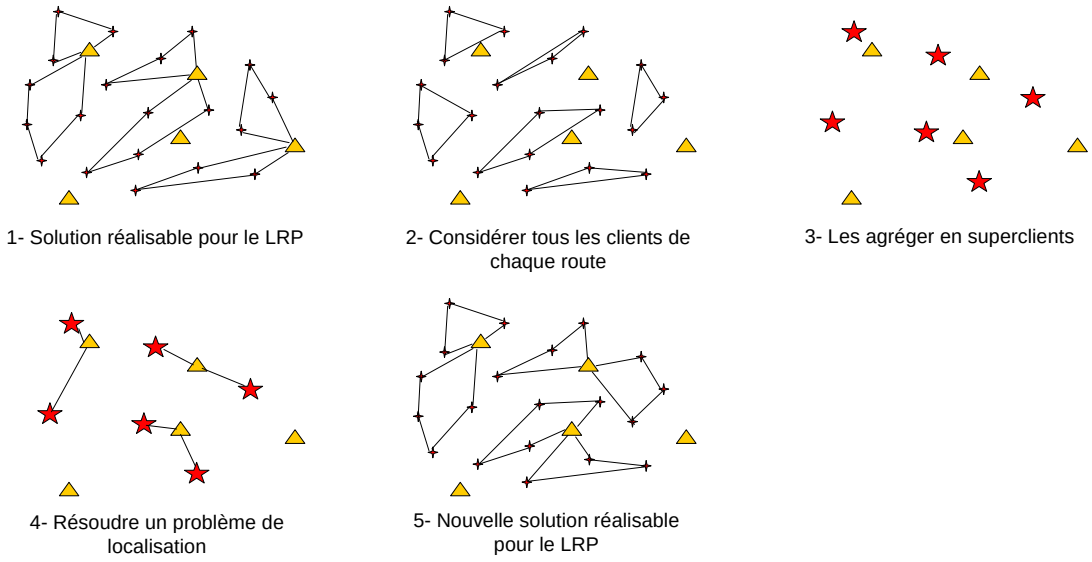


FIG. 7.1 – *Phase de Localisation*

articles sur le *VRP* résolu par cette approche sont également cités dans le paragraphe 2.4.2. Le principe général et surtout la version granulaire sont rappelés ici. *TS* est une métaheuristique qui effectue à chaque itération le meilleur mouvement réalisable dans un voisinage donné, même si la variation de coût est positive. Cette technique permet d'échapper aux optima locaux. Cependant, un mécanisme de mémorisation des derniers mouvements réalisés doit être mis en place afin d'interdire les retours en arrière : les mouvements entraînant de tels retours sont alors rendus *tabous* durant un nombre d'itérations.

L'idée de la granularité a été introduite par Toth et Vigo (2003). Le principe de la méthode taboue reste le même, mais la taille du voisinage est réduite. Pour cela, les mouvements basés sur des éléments de solution ayant peu de chances d'appartenir à la solution optimale sont interdits. Ce principe peut être vu comme une intensification dans l'espace de recherche. Le *GTS* explorant un voisinage plus restreint, il a l'avantage d'être plus rapide que le *TS* classique.

7.3.2 Voisinage utilisé

Dans l'article de Toth et Vigo (2003), le voisinage se compose des solutions accessibles en échangeant deux, trois ou quatre arcs. Dans notre cas, le voisinage utilisé au sein du *TS* représente toutes les solutions qui peuvent être atteintes en transférant un nœud-client k de sa position courante à une autre. Ce transfert nécessite l'introduction d'arcs dont un arrivant au nœud k et un en repartant. L'exploration du voisinage peut ainsi se faire en testant l'insertion de chaque arc (i,j) non-tabou et non-utilisé dans la solution courante. Une telle insertion implique deux voisins possibles à évaluer. Un exemple est donné dans la figure 7.2. Afin

d'obtenir une solution réalisable, trois arcs doivent être supprimés de la solution courante, et deux autres ajoutés. Quand un mouvement est sélectionné et effectué, les trois arcs retirés sont considérés comme tabous. Il est possible que tous les clients affectés à un dépôt donné soient progressivement déplacés vers un autre dépôt ouvert. Si cela se produit, les dépôts ne servant aucun client sont alors fermés. Lors de la fermeture d'un site, la variation de coût engendrée est alors comptabilisée dans le gain réalisé par le mouvement considéré. La même chose peut se produire pour la fermeture d'une tournée.

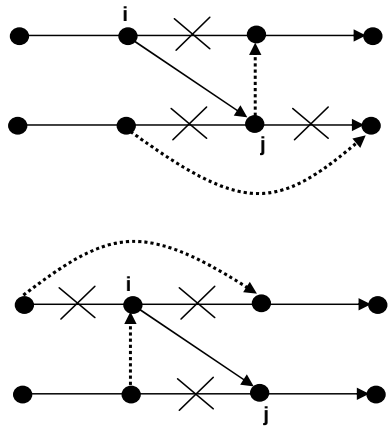


FIG. 7.2 – *Mouvements du GTS possibles lors de l'insertion d'un arc*

7.3.3 Mécanisme de granularité

L'exploration du voisinage décrit précédemment se fait en parcourant la liste des arcs non-tabous et non-utilisés dans la solution courante. Le principe de granularité qui va accélérer les calculs doit donc restreindre cette liste. Pour cela, l'idée est de se baser sur un graphe réduit. Il contient uniquement les arcs (i,j) reliant deux clients et de coût inférieur à $\beta\tilde{d}$, où β est une constante et $\tilde{d}$ est le coût moyen des arcs reliant deux clients dans une *bonne* solution réalisable, comme suggéré par Toth et Vigo (2003). On exclut ainsi les arcs connectant deux clients éloignés, qui ont peu de chance de figurer dans une solution optimale. Cependant, la réduction du voisinage doit être temporaire pour éviter d'écarter une bonne solution. Si aucune amélioration n'est obtenue après un nombre donné *NoImp* d'itérations successives du *GTS* avec la valeur initiale de β, la granularité est modifiée afin d'élargir le voisinage : les arcs de coût inférieur à $\beta'\tilde{d}$, avec $\beta' > \beta$, sont ajoutés au graphe réduit. Le retour à la granularité initiale se fait quand une solution améliorant la meilleure de la phase de routage considérée est trouvée ou quand un certain nombre d'itérations *Div* a été exécuté. Un tel ajustement dynamique de la granularité permet de diversifier la recherche.

La *bonne* solution réalisable mentionnée précédemment pour déterminer $\tilde{d}$ est calculée pendant l'initialisation de la méthode coopérative. Elle correspond à la meilleure solution

trouvée après $2m$ exécutions des heuristiques constructives randomisées ($AB2$ ou $AB3$ selon les capacités sur les dépôts, voir chapitre 4) suivie de la recherche locale *LS1*.

7.3.4 Pénalisation de violation de capacité

Malheureusement, malgré l'utilisation de deux granularités, il reste difficile de sortir de certains extrema locaux. Ceci est dû aux contraintes de capacités à la fois sur les dépôts et sur les véhicules. Afin de donner un peu plus de liberté à la recherche, les violations de capacités sont autorisées pour les véhicules et les dépôts, en contrepartie d'une pénalité dans la fonction-objectif, comme suggéré dans d'autres travaux sur le *VRP* (Gendreau *et al.*, 1994). Dès qu'une contrainte de capacité est violée, un terme $P \times OverCapa$ est ajouté à la fonction-objectif. *OverCapa* est le dépassement de capacité, et P est un facteur unitaire de pénalité. Ce facteur est ajusté dynamiquement afin de refléter le besoin d'élargir la recherche ou au contraire de la recentrer sur des solutions réalisables.

Si le nombre de solutions irréalisables visitées durant un certain nombre d'itérations consécutives atteint *PenaltyIt*, ou si le nombre de tournées et de dépôts ayant une charge dépassant leur capacité dans une solution visitée excède *MaxPenalty*, alors il faut recentrer la recherche dans le domaine réalisable en augmentant le facteur de pénalité ainsi : $P = P \times IncPenalty$ avec $IncPenalty > 1$. Inversement, si aucune solution irréalisable n'a été visitée durant les *NoPenaltyIt* dernières iterations, cela vient peut-être d'un facteur de pénalité trop élevé, alors P est réduit par la formule $P = P \times DecPenalty$ avec $DecPenalty < 1$.

Le *GTS* est exécuté pendant un nombre fixé d'itérations *GTSit*. Comme le *GTS* ne considère que des transferts de clients, la solution obtenue est parfois améliorée en utilisant en post-optimisation la recherche locale *LS1* du chapitre 4. Cette dernière inclut en effet des mouvements plus complexes.

7.4 L'heuristique cooperative complète

L'approche proposée dans ce chapitre pour résoudre le *LRP* est appelée *LRGTS* pour *Lagrangean Relaxation and Granular Tabu Search* et exploite des caractéristiques spécifiques au problème. Elle résout alternativement des phases de localisation des dépôts et de routage. En localisation, le but est d'affecter des super-clients, formés par agrégation des clients de chaque tournée, à un sous-ensemble de dépôts à déterminer. Cette phase est résolue par une approche basée sur une relaxation lagrangienne. En routage, le but est d'améliorer les coûts des tournées en fonction des dépôts ouverts déterminés par la phase de localisation précédente. Ce *VRP* multi-depot est traité par une recherche de type tabou granulaire. Le résultat donne de nouvelles tournées à agréger pour le prochain problème de localisation.

Les échanges naturels entre ces deux phases successives leur permettent d'interagir sans hiérarchie. Cette synergie renforce la méthode. Une itération globale regroupe une étape de localisation suivie d'une de routage.

Cependant, ce schéma d'alternance simple tend à converger prématurément vers des optima locaux. En effet, le choix des dépôts dépend fortement des super-clients qui, de par le

Algorithme 10 Approche coopérative complète pour le *LRP - LRGTS*

Créer une bonne solution pour le GTS avec une heuristique //voir section 7.3.3
Créer les graphes réduits en utilisant le coût moyen des arcs de la solution //voir section 7.3.3
$NvxTours = 0$ //Nombre de perturbations effectuées par création de nouvelles tournées
$BestCost = +\infty$ //Coût de la meilleure solution
$LastCost = +\infty$ //Coût de la solution obtenue lors de l'itération précédente
Créer une solution initiale S pour la méthode coopérative //voir section 7.4
répéter
  si $(Cout(S) \geq LastCost)$ **alors**
    Créer de nouvelles routes à partir du compteur de fréquences des arêtes //voir section 7.4
    $LastCost = +\infty$
    $NvxTours = NvxTours+1$
  sinon
    $LastCost = Cout(S)$
  fin si
  Créer des super-clients à partir des tournées //voir section 7.2.1
  Résoudre le problème de localisation à partir des super-clients //voir section 7.2.1
  Extraire la solution correspondante du LRP //voir section 7.2.5
  $GTS(S)$ //voir section 7.3
  $LS1(S)$
  Mise à jour du compteur de fréquences des arêtes utilisées dans S
  si $(Cout(S) < BestCost)$ **alors**
    $BestCost = Cout(S)$
    $BestSol = S$
  fin si
jusqu'à $(NvxTours > MaxPertub)$
Renvoyer$(BestSol)$

routage, sont eux-même tributaires de la partie localisation. Pour remédier au risque de convergence précoce, une heuristique, tirant parti d'informations récoltées au cours de la recherche, est utilisée ponctuellement pour fournir de nouvelles tournées à la phase de localisation, à la place de celles renvoyées par le *GTS*.

A la fin de chaque itération globale, un compteur de fréquences est mis à jour pour chaque arête utilisée dans la solution courante. Si de plus cette dernière n'a pas été améliorée au cours de l'itération globale se terminant, une régénération de la solution est effectuée avant le début de la prochaine étape de localisation (voir l'algorithme 10). Les tournées issues de la meilleure solution obtenue lors du dernier *GTS* sont ignorées et de nouvelles sont construites par une heuristique utilisant un nombre limité Δ d'arêtes (i,j), $i,j \in J$. L'idée est d'apporter de la flexibilité à la phase de localisation. En effet, comme détaillé ci-dessous, Δ permet de réduire le nombre d'arêtes par tournée et donc d'obtenir, après agrégation des clients, des super-clients de plus faible demande.

Cette heuristique construit une liste L d'arêtes reliant des clients, classées par ordre décroissant de fréquence, et initialise une liste U des arêtes déjà insérées dans des tournées, à l'ensemble vide. Ensuite, le principe est de parcourir la liste L, en commençant par les arêtes les plus fréquemment utilisées et d'ajouter itérativement une par une des arêtes dans U. Deux conditions doivent cependant être respectées : aucune fourche ne doit être créée et la charge de chaque composante connexe de U ne doit pas excéder la capacité Q des véhicules. L'heu-

ristique se termine quand $|U| = \Delta$ ou quand plus aucune composante connexe ne peut être agrandie sans violer la capacité d'un véhicule. De cette liste U sont alors issus les super-clients fournit à la phase de localisation. Ils sont représentés par l'agrégation des clients de chaque composante connexe de U. Néanmoins, afin d'apporter de la diversité lors des appels aux régénérations de tournées, chaque arête de L est prise en compte avec une probabilité donnée, *prob*, avant son insertion éventuelle dans U.

Δ est initialisé à une valeur relativement faible Δ_{init} en rapport avec le nombre maximal d'arêtes pouvant être introduite dans une solution. Ensuite, à chaque appel à une régénération de solution, l'information fournie par les compteurs de fréquences sur les arêtes devient de plus en plus pertinente, et la valeur de Δ augmente progressivement d'un facteur $\gamma > 1$. Ce principe d'insertion d'un nombre grandissant d'arêtes peut être comparé à un arbre de branchement fixant itérativement de nouvelles variables à chaque nouvelle régénération. Pour la solution initiale S de l'algorithme complet du *LRGTS*, aucune fréquence n'étant disponible concernant les arêtes, le procédé consiste à construire des tournées par plus proche voisin mais en réduisant leur charge à la plus forte demande des clients ($max_{j \in J}(d_j : j \in J)$).

Un aperçu de la méthode est présenté dans l'algorithme 10. $MaxPertub$ représente le nombre de régénérations de tournées autorisé, utilisé comme critère d'arrêt dans l'algorithme. Grâce à la technique de perturbation exploitant des informations récoltées au fil des itérations sur l'utilisation des arêtes, l'algorithme est plus performant et permet d'obtenir de meilleures solutions comme le montre les résultats présentés au paragraphe suivant.

7.5 Évaluation Numérique

7.5.1 Implémentation et instances

La méthode proposée dans ce chapitre, *LRGTS*, est évaluée sur les trois ensembles d'instances *I1*, *I2* et *I3* décrits en Annexe, et comparée avec le $MA|PM$ exposé dans le chapitre 6. Les algorithmes sont codés en C et exécutés sur un PC Dell Optiplex GX260, avec un Pentium 4 à 2.4 GHz, 512 Mo de RAM et Windows XP.

Les résultats sont fournis dans les tableaux 6.2-6.3 et 6.4. Les colonnes n et m indiquent respectivement le nombre de clients et de dépôts de l'instance. β est le nombre de clusters, *type* est soit relatif à la capacité des tournées (instances *I1* et *I3*), soit correspond au ratio des clients appartenant à des clusters sur n (instances $I2$). Les colonnes *Coût* et Tps donnent le coût des solutions obtenues et les temps de calcul nécessaire en secondes pour les obtenir. *dep* et *tr* indiquent le nombre de dépôts et de tournées ouvertes dans la solution restituée. Enfin, la colonne E_{MAPM} est l'écart relatif en pourcentage entre le coût de la solution trouvée par le $LRGTS$ et le $MA|PM$ ($E_{MAPM} = ((cout(LRGTS) - cout(MA \mid PM))/cout(MA \mid PM)) * 100$).

Une analyse de l'impact des différents composants de la méthode est proposée au Paragraphe 7.6. Mais avant, le paragraphe suivant donne le paramétrage de la méthode.

7.5.2 Paramétrage de l'algorithme

Les paramètres suivants, utilisés pour obtenir les résultat présentés, ont été choisi après une phase de test. Tout d'abord, concernant la méthode globale, le seul paramètre concerne le nombre de perturbations des tournées autorisées : $MaxPertub = n/3$.

Ensuite, concernant la phase de localisation, sont nécessaires :

- le nombre d'arêtes insérées lors de la première régénération de tournées : $\Delta_{init} = 3 + (n - \hat{k}) \times 0.25$ avec $\hat{k} = \sum_{j \in J} d_j/Q$;
- le facteur d'accroissement du nombre d'arêtes insérées lors de la régénération de tournées : $\gamma = 1.25$;
- la probabilité d'acceptation d'une arête lors de la régénération de tournées : $prob = 0.80$;
- le nombre d'itérations pour l'optimisation par sous-gradients : $LagrIt = 200$.

Enfin, pour la phase de routage, on a besoin des valeurs suivantes :

- le nombre d'itérations dans le GTS : $GTSit = n$;
- les paramètres relatifs au graphe granulaire : $\beta = 1.20$, $\beta' = 1.85$;
- les paramètres relatifs à la pénalisation des violations de capacité : $NoImp = 0.05n$, $Div = 0.20n$, valeur initiale de $P = 2\beta\tilde{d}/AvDem$ ($AvDem$ étant la demande moyenne des clients), $NoPenaltyIt = 3$, $MaxPenalty = 3$, $PenaltyIt = 0.08n$, $DecPenalty = 0.8$, $IncPenalty = 1.4$;
- la taille de la liste-taboue : $0.15n$.

7.5.3 Résultats pour le premier groupe d'instances - *I1*

Le tableau 7.1 montre une comparaison entre $LRGTS$ et $MA|PM$ sur l'ensemble $I1$.

$LRGTS$ est de loin plus rapide que $MA|PM$, avec une moyenne de 17.1 s contre 76.7 s pour $MA|PM$. Même le $GRASP$ sans PR, qui est réputé comme une métaheuristique rapide, est plus lent puisqu'il nécessite 27.7 s en moyenne. Cette rapidité ne se fait cependant pas au détriment de la qualité des résultats, bien au contraire. Il réduit le coût des solutions de 0.55% en moyenne par rapport au $MA|PM$ qui améliore déjà les coûts du $GRASP$ de 2.06%.

Par contre, en observant instance par instance, il apparaît que $MA|PM$ obtient de meilleures solutions dans 50% des cas. De plus, sur 1/5 des instances, les coûts sont identiques. $LRGTS$ n'est donc plus performant que pour moins de 30% des solutions. Si l'écart moyen est en faveur de $LRGTS$, c'est parce que les écarts sont plus faibles quand $MA|PM$ trouve de meilleurs résultats : de moins de 1%, sauf une instance avec 1.97%. Par contre, quand $LRGTS$ fournit des coûts inférieurs, son gain peut être important, allant jusqu'à 12.85%, et les deux plus gros écarts en faveur de $LRGTS$ ont lieu quand ce dernier utilise un dépôt de moins que $MA|PM$. Sur toutes les autres instances, le nombre de dépôts ouverts est identique pour les deux méthodes.

7.5.4 Résultats pour le deuxième groupe d'instances - *I2*

Le tableau 7.2 donne les résultats obtenus sur *I2*. Là encore, *LRGTS* est la méthode la plus rapide, avec des temps de calcul près de 10 fois inférieurs à *MA|PM*. Par contre, concernant le coût moyen des solutions, les deux approches comparées fournissent les résultats semblables. Le nombre de fois où *MA|PM* fournit des solutions de moindre coût est plus grand qu'avec *LRGTS* (70% des cas) mais, comme sur le jeux d'instances *I1*, quand *LRGTS* apporte un gain, celui-ci est en général plus important que lorsque c'est *MA|PM*. Une des raisons possibles pouvant expliquer le fait que les performances de *LRGTS* sont moins remarquables sur l'ensemble des instances de *I2* est l'absence de capacité sur les dépôts. En effet, lors de la phase de localisation, des problèmes de type sac-à-dos sont résolus par dépôt, nécessitant ici l'introduction de capacités fictives W_i, choisies arbitrairement. Cette approche n'est donc certainement pas la plus adaptée aux instances de type $I2$, même si malgré tout, les résultats restent de très bonne qualité.

				MAPM				*LRGTS*				
n	m	β	type	Coût	Tps	dep	tr	Coût	Tps	dep	tr	E_{MAPM}
20	5	0	a	54793	0.3	3	5	55131	0.4	3	5	0.62
20	5	0	b	39104	0.3	2	3	39104	0.2	2	3	0.00
20	5	2	a	48908	0.4	3	5	48908	0.5	3	5	0.00
20	5	2	b	37542	0.3	2	3	37542	0.1	2	3	0.00
50	5	0	a	90160	2.6	3	12	90160	0.3	3	12	0.00
50	5	0	b	63242	3.2	2	6	63256	1.0	2	6	0.02
50	5	2	a	88298	3.4	3	12	88715	1.8	3	12	0.47
50	5	2	b	67893	2.9	3	6	67698	1.8	3	6	-0.29
50	5	2'	a	84055	3.2	3	12	84181	2.0	3	12	0.15
50	5	2'	b	51822	4.2	3	6	51992	0.9	3	6	0.33
50	5	3	a	86203	3.1	2	12	86203	0.3	2	12	0.00
50	5	3	b	61830	4.9	2	6	61830	0.5	2	6	0.00
100	5	0	a	281944	26.3	3	24	277935	8.7	3	24	-1.42
100	5	0	b	216656	34.5	3	12	214885	8.3	3	11	-0.82
100	5	2	a	195568	35.8	2	24	196545	2.3	2	24	0.50
100	5	2	b	157325	36.4	2	11	157792	3.3	2	11	0.30
100	5	3	a	201749	28.7	2	24	201952	2.4	2	24	0.10
100	5	3	b	153322	33.3	2	11	154709	2.9	2	12	0.90
100	10	0	a	316575	24.7	4	25	291887	14.1	3	26	-7.80
100	10	0	b	270251	36.0	4	11	235532	14.0	3	12	-12.85
100	10	2	a	245123	24.6	3	24	246708	14.4	3	24	0.65
100	10	2	b	205052	31.6	3	11	204435	10.1	3	11	-0.30
100	10	3	a	253669	29.0	3	24	258656	13.3	3	25	1.97
100	10	3	b	204815	36.5	3	11	205883	10.8	3	11	0.52
200	10	0	a	483497	345.1	3	47	481676	62.0	3	47	-0.38
200	10	0	b	380044	463.0	3	22	380613	60.3	3	22	0.15
200	10	2	a	451840	280.6	3	48	453353	60.3	3	48	0.33
200	10	2	b	375019	321.0	3	22	377351	76.9	3	23	0.62
200	10	3	a	478132	212.9	3	46	476684	77.2	3	47	-0.30
200	10	3	b	364834	272.0	3	22	365250	73.3	3	22	0.11
	Moyenne				76.7	2.8	16.9		17.5	2.7	17.1	-0.55

NB : un écart négatif indique une amélioration par rapport à la solution prise en référence.

TAB. 7.1 – *Résultats du LRGTS - Instances I1 avec capacité sur les dépôts*

				MAPM				*LRGTS*				
n	m	β	type	Coût	Tps	dep	tr	Coût	Tps	dep	tr	E_{MAPM}
100	10	0	0.75	1493.92	31.5	3	11	1490.82	3.3	3	11	-0.21
100	20	0	0.75	1471.36	35.6	2	11	1471.76	6.5	3	11	0.03
100	10	0	1	1418.83	36.2	3	10	1412.04	4.2	3	10	-0.48
100	20	0	1	1492.46	36.4	2	11	1443.06	7.4	2	11	-3.31
100	10	3	0.75	1173.22	31.9	2	11	1187.63	6.9	2	11	1.23
100	20	3	0.75	1115.37	42.7	3	11	1115.95	6.8	3	11	0.05
100	10	3	1	793.97	38.0	2	11	813.28	5.2	3	12	2.43
100	20	3	1	730.51	49.3	2	11	742.96	5.9	3	11	1.70
100	10	5	0.75	1262.32	36.8	3	11	1267.93	4.3	3	11	0.45
100	20	5	0.75	1251.32	47.7	3	11	1256.12	6.3	3	11	0.38
100	10	5	1	903.82	35.1	3	12	913.06	4.0	3	12	1.02
100	20	5	1	1022.93	62.6	4	11	1025.51	4.9	3	11	0.25
150	10	0	0.75	1959.39	128.5	3	16	1946.01	12.5	3	16	-0.68
150	20	0	0.75	1881.67	144.2	3	16	1875.79	18.5	3	16	-0.31
150	10	0	1	1984.25	111.0	3	16	2010.53	11.1	3	16	1.32
150	20	0	1	1855.25	144.1	3	16	1819.89	15.8	3	16	-1.91
150	10	3	0.75	1448.27	166.6	2	16	1448.65	22.0	2	16	0.03
150	20	3	0.75	1459.83	154.8	2	16	1492.86	28.0	3	16	2.26
150	10	3	1	1207.41	161.4	3	17	1211.07	14.6	3	17	0.30
150	20	3	1	934.79	196.1	3	17	936.93	13.7	3	17	0.23
150	10	5	0.75	1720.30	143.8	3	16	1729.31	17.9	3	16	0.52
150	20	5	0.75	1429.34	155.7	4	16	1424.59	18.5	3	16	-0.33
150	10	5	1	1203.44	153.8	3	17	1216.32	14.5	3	16	1.07
150	20	5	1	1158.54	223.0	3	17	1162.16	14.3	3	17	0.31
200	10	0	0.75	2293.99	418.3	3	21	2296.52	32.6	3	22	0.11
200	20	0	0.75	2277.39	458.4	3	21	2207.50	39.6	4	22	-3.07
200	10	0	1	2274.57	376.8	3	21	2260.87	32.8	4	21	-0.60
200	20	0	1	2376.25	436.3	3	21	2259.52	40.2	3	21	-4.91
200	10	3	0.75	2106.26	350.5	3	21	2120.76	47.2	3	21	0.69
200	20	3	0.75	1771.53	377.0	2	20	1737.81	59.3	3	21	-1.90
200	10	3	1	1467.54	322.2	2	21	1488.55	36.7	2	21	1.43
200	20	3	1	1088.00	505.0	3	22	1090.59	38.7	3	22	0.24
200	10	5	0.75	1973.28	412.8	4	22	1984.06	41.6	4	22	0.55
200	20	5	0.75	1979.05	406.1	5	21	1986.49	51.8	4	22	0.38
200	10	5	1	1782.23	352.8	3	22	1786.79	34.0	3	22	0.26
200	20	5	1	1396.24	529.8	5	22	1401.16	43.2	5	22	0.35
	Moyenne				203.1	2.9	16.2		21.2	3.1	16.3	0.00

NB : un écart négatif indique une amélioration par rapport à la solution prise en référence.

TAB. 7.2 – *Résultats du LRGTS - Instances I2 sans capacité sur les dépôts*

7.5.5 Résultats pour le troisième groupe d'instances - *I3*

Dans le tableau 7.3, *LRGTS* est encore comparé avec *MA|PM*, mais sur les instances *I3*. *LRGTS* est cette fois-ci compétitif avec *MA|PM*, et obtient des résultats de coût inférieur en moyenne (de 0.34%). Ce gain est un peu moins important que celui sur le groupe d'*I1*, instances comportant également des capacités sur les dépôts. Cependant, en regardant plus en détails, il trouve des meilleures solutions sur un plus grand nombre d'instances (50% contre 30%), sans qu'une taille ne paraisse plus avantagée qu'une autre.

Les conclusions sur les temps de calcul sont encore similaires à celles concernant les ensembles *I1* et *I2*.

			MAPM				*LRGTS*				
n	m	type	Coût	Tps	dep	tr	Coût	Tps	dep	tr	E_{MAPM}
21	5	6000	424.9	0.3	2	4	424.9	0.0	2	4	0.00
22	5	4500	611.8	0.3	2	3	587.4	0.0	1	3	-3.99
27	5	2500	3062.0	1.0	2	4	3065.2	0.0	2	4	0.11
29	5	4500	512.1	0.8	2	4	512.1	0.1	2	4	0.00
32	5	8000	571.9	0.8	2	4	584.6	0.1	2	4	2.21
32	5	11000	534.7	1.0	2	3	504.8	0.1	1	3	-5.60
36	5	250	485.4	1.4	2	4	476.5	0.1	1	4	-1.84
50	5	160	565.6	3.8	2	6	586.4	0.4	2	6	3.68
75	10	140	866.1	9.4	3	9	863.5	1.0	3	10	-0.29
88	8	9000000	355.8	34.2	2	7	368.7	2.7	2	6	3.63
100	10	200	850.1	44.5	2	8	842.9	5.8	2	8	-0.86
134	8	850	5950.1	110.5	4	10	5809.0	13.3	3	11	-2.37
150	10	8000000	44011.7	255.0	3	11	44386.3	30.3	3	14	0.85
Moyenne				35.6	2.3	5.9		4.2	2.0	6.2	-0.34

NB : un écart négatif indique une amélioration par rapport à la solution prise en référence.

TAB. 7.3 – *Résultats du LRGTS - Instances I3 avec capacité sur les dépôts*

7.6 Impact des composants de la méthode

Les résultats donnés en section 7.5 montrent que *LRGTS* est compétitif avec *MA|PM*. Il donne des résultats de qualité similaire en moyenne, mais apporte un gain de 0.27% sur l'ensemble des instances. Son principal avantage est sa vélocité puisqu'il est près de 10 fois plus rapide que *MA|PM*. De plus, il semble plus stable, en particulier sur les instances avec capacité sur les dépôts. En effet, il n'est jamais très éloigné des résultats de *MA|PM* quand ce dernier fournit les meilleurs résultats, mais propose des gains bien plus importants dans le cas inverse.

Le but de ce paragraphe est de regarder plus en détail le rôle des différents éléments de la méthode, en particulier la granularité utilisée dans la recherche taboue de la phase de routage, et la coopération relative aux régénérations de tournées. Les résultats sont fournis dans les tableaux 7.5, 7.6 et 7.7, où cette fois-ci, les colonnes E_{LRGTS} présentent des écarts relatifs en prenant le *LRGTS* complet comme référence.

7.6.1 Impact de la granularité

Le but de la granularité dans la recherche taboue est de réduire le voisinage visité en retirant les attributs non prometteurs. Comme dans toutes les techniques de réduction de voisinage, l'objectif est la diminution des temps de calcul, mais elle se fait généralement au prix d'une baisse de la qualité des solutions. Ici, l'exploration s'effectue par insertion d'arêtes dans la solution et la granularité est implémentée sur un graphe réduit. Les arêtes de plus fort coût sont retirées en partant du principe qu'ils relient des clients distants et n'ont que très peu de chance d'appartenir à une solution optimale. Cette technique peut malheureusement empêcher l'accès à certaines solutions intéressantes.

Pour évaluer ce risque, deux algorithmes sont comparés pour la phase de routage : le *GTS* présenté dans le paragraphe 7.3, et la même méthode mais sans la granularité, notée *TS*. Les résultats sont donnés dans le tableau 7.4. Pour chaque groupe d'instances, ce tableau donne le

n	m	E_{GTS}	Tps	E_{TS}	Tps
Premier ensemble - *I1* :					
20	5	-9.1	0.0	-10.4	0.0
50	5	-7.8	0.1	-8.8	0.9
100	5	-4.1	0.8	-4.7	14.8
100	10	-2.6	1.1	-3.3	23.2
200	10	-1.6	6.0	-2.5	316.5
Moyenne		-5.0	1.6	-5.9	71.1
Deuxième ensemble - *I2* :					
100	10	-6.1	1.3	-7.4	22.6
100	20	-5.6	1.2	-6.6	16.0
150	10	-5.0	5.3	-6.2	97.4
150	20	-4.4	4.3	-5.6	100.1
200	10	-3.9	10.8	-5.2	354.1
200	20	-4.3	13.5	-5.7	412.0
Moyenne		-4.9	6.1	-6.1	167.0
Troisième ensemble - *I3* :					
≤ 50	5	-8.5	0.0	-10.1	0.2
> 50 et ≤ 150	≤ 10	-5.7	2.5	-7.1	41.8
Moyenne		-7.4	1.0	-9.0	16.2
Moyenne		**-5.3**	**3.5**	**-6.5**	**105.8**

NB : un écart négatif indique une amélioration.

TAB. 7.4 – *Impact de la granularité sur la construction des tournées*

temps moyen passé dans ces phases de routage (Tps) et la variation moyenne en pourcentage du coût entre la solution donnée en entrée et récupérée en fin du GTS (E_{GTS}) ou du TS (E_{TS}). Les résultats montrent que la granularité permet une réduction d'un facteur 30 des temps de calcul dans la partie routage, alors que l'exploration du voisinage complet dans TS améliore un peu plus la solution de départ en passant d'un gain de 5.3% à 6.5% en moyenne.

Lorsque ce ne sont plus les performances de la phase de routage qui sont observées, mais celles de l'algorithme complet, les conclusions sont différentes, comme l'indiquent les résultats du tableau 7.5. *LRTS* correspond à la méthode coopérative dans laquelle le GTS est remplacé par un TS. Les valeurs rapportées pour chaque groupe d'instances sont les coûts et temps de calcul moyens de chaque algorithme et l'écart en pourcentage des solutions avec la version utilisant un voisinage réduit dans la phase de routage (E_{LRGTS}). Les temps de calcul sont toujours fortement réduits avec l'utilisation de la granularité (division par un facteur 7), par contre, les coûts sont également inférieurs, avec un gain de 0.5%. Une explication possible est la convergence prématurée de *LRTS*. L'algorithme utilisant TS alterne alors moins souvent entre des phases de localisation et routage, ce qui a pour conséquence un appel plus fréquent à la régénération de tournées. Il n'a pas le temps de tirer profit de la synergie entre les deux phases et explore ainsi de façon moins judicieuse l'espace des solutions.

7.6.2 Impact de la diversification

L'échange naturel d'informations entre la localisation et le routage intervient sans hiérarchisation. Cependant, l'algorithme peut converger prématurément vers un optimum local et il faut alors perturber la solution pour y échapper. Cette perturbation se fait par reconstruction de nouvelles tournées en utilisant des informations récoltées au cours des itérations. Les arêtes

		LRGTS		*LRTS*		
n	m	Coût	Tps	Coût	Tps	E_{LRGTS}
Premier ensemble - *I1* :						
20	5	45171	0.3	45428	0.3	0.5
50	5	74254	1.1	75264	2.1	1.5
100	5	200636	4.6	200949	18.1	0.1
100	10	240517	12.8	242350	35.5	0.8
200	10	422488	68.3	421862	373.2	-0.2
Moyenne			17.5		85.9	0.6
Deuxième ensemble - *I2* :						
100	10	1180.79	4.6	1190.38	25.3	0.7
100	20	1175.89	6.3	1190.31	20.4	1.1
150	10	1593.65	15.4	1602.11	104.5	0.6
150	20	1452.03	18.1	1447.82	111.6	-0.3
200	10	1989.59	37.5	1980.62	373.2	-0.4
200	20	1780.51	45.5	1767.34	435.0	-0.7
Moyenne			21.2		178.3	0.2
Troisième ensemble - *I3* :						
≤ 50	5	842.7	0.1	850.34	0.3	1.5
> 50 et ≤ 150	≤ 10	10454.1	10.6	10505.08	48.4	1.1
Moyenne			4.2		18.8	1.4
Moyenne			**17.0**		**117.0**	**0.5**

NB : un écart négatif indique une amélioration.

TAB. 7.5 – *Impact de la granularité sur la méthode coopérative*

		LRGTS		No Restart		
n	m	Coût	Tps	Coût	Tps	E_{LRGTS}
Premier ensemble - *I1* :						
20	5	45171	0.3	45217	0.2	0.1
50	5	74254	1.1	76704	0.3	3.5
100	5	200636	4.6	203786	1.2	1.6
100	10	240517	12.8	261413	3.6	8.5
200	10	422488	68.3	442456	15.4	4.7
Moyenne			17.5		4.1	3.9
Deuxième ensemble - *I2* :						
100	10	1180.79	4.6	1205.65	1.1	2.1
100	20	1175.89	6.3	1221.00	1.3	3.6
150	10	1593.65	15.4	1669.42	3.0	4.9
150	20	1452.03	18.1	1485.45	3.8	2.2
200	10	1989.59	37.5	2045.73	5.7	2.9
200	20	1780.51	45.5	1825.56	6.5	2.3
Moyenne			21.2		3.6	3.0
Troisième ensemble - *I3* :						
≤ 50	5	842.74	0.1	867.54	0.0	4.7
> 50 et ≤ 150	≤ 10	10454.08	10.6	10803.46	1.8	4.6
Moyenne			4.2		0.7	4.6
Moyenne			**17.0**		**3.3**	**3.6**

NB : un écart négatif indique une amélioration.

TAB. 7.6 – *Impact de la diversification*

		LRGTS		No Coop		
n	m	Coût	Tps	Coût	Tps	E_{LRGTS}
Premier ensemble - *I1* :						
20	5	45171	0.3	45171	0.4	0.0
50	5	74254	1.1	74445	1.1	0.3
100	5	200636	4.6	200922	4.2	0.1
100	10	240517	12.8	240539	10.8	0.0
200	10	422488	68.3	423421	56.2	0.2
Moyenne			17.5		14.6	0.1
Deuxième ensemble - *I1* :						
100	10	1180.79	4.6	1190.35	4.3	0.8
100	20	1175.89	6.3	1183.38	6.5	0.5
150	10	1593.65	15.4	1594.86	12.8	0.1
150	20	1452.03	18.1	1452.65	16.5	0.1
200	10	1989.59	37.5	1991.32	31.9	0.1
200	20	1780.51	45.5	1786.44	35.2	0.3
Moyenne			21.2		17.9	0.3
Troisième ensemble - *I1* :						
≤ 50	5	842.74	0.1	849.27	0.1	1.3
> 50 et ≤ 150	≤ 10	10454.0	10.6	10378.31	9.2	0.5
Moyenne			4.2		3.6	1.0
Moyenne			**17.0**		**14.3**	**0.4**

NB : un écart négatif indique une amélioration.

TAB. 7.7 – *Impact de l'utilisation coopérative d'informations*

reliant les clients les plus utilisées en fin de chaque itération globale sont ainsi favorisées.

Le tableau 7.6 compare les résultats obtenus avec et sans reconstruction de tournées, en utilisant les mêmes indicateurs que dans le tableau 7.5. Bien évidemment, sans technique permettant d'échapper à un optimum local, les temps de calcul sont divisés par 5. Par contre, l'espace des solutions étant bien moins exploré, les coûts augmentent de 3.6% en moyenne. L'introduction de perturbations est donc bien nécessaire.

Le tableau 7.7, utilisant toujours les mêmes indicateurs, compare les résultats obtenus par le *LRGTS* avec une version n'utilisant pas l'information basée sur les fréquences d'arêtes. La perturbation consiste alors à reconstruire chaque tournée en choisissant d'abord aléatoirement un client puis en la complétant par l'utilisation d'une heuristique de type Plus Proche Voisin (PPV). Cette intervention *aveugle* sur les tournées est suffisante pour améliorer les résultats de l'algorithme sans régénérations de tournées (en comparant les résultats avec ceux fournis dans le tableau 7.6), mais une coopération progressive, pouvant être vue comme un branchement dans un arbre de recherche sur les arêtes les plus prometteuses, apporte un gain supplémentaire de 0.4%.

7.7 Conclusion

Dans ce chapitre, une nouvelle métaheuristique pour le problème de localisation-routage généralisé est présentée. La méthode est une métaheuristique coopérative appelée *LRGTS*, échangeant un certain nombre d'informations lors d'alternance entre deux principales phases de l'algorithme. La première phase porte sur la localisation des dépôts et est abordée en agré-

geant les clients d'une même tournée en un super-client, afin de n'avoir plus qu'un problème de localisation de dépôts à traiter. Ce dernier est résolu avec une relaxation lagrangienne des contraintes d'affectation. Ensuite, l'ensemble de dépôts sélectionné lors de la première phase est fourni à la phase de routage, qui va alors traiter un *VRP* multi-dépôt à l'aide d'une recherche de type tabou granulaire. De nouveaux super-clients pour la phase de localisation suivante vont alors découler des tournées obtenues. De plus, afin d'échapper à des extrema locaux, un mécanisme de perturbation de la solution propose de nouvelles tournées à la prochaine phase de localisation lors de convergence prématurée. Des statistiques sont collectées sur les fréquences d'utilisation des arêtes reliant des clients, et la création de nouvelles tournées introduit en nombre restreint les arêtes les plus fréquentes. L'étroite coopération entre les phases permet de renforcer la méthode.

LRGTS est testée sur les trois ensembles d'instances disponibles. Les résultats montrent que la méthode est capable d'obtenir de très bons résultats en moyenne par rapport aux métaheuristiques présentées dans les chapitres précédents. Par contre, sur les solutions prises individuellement, le $MA|PM$ peut obtenir des coûts plus faibles, en particulier sur les instances de l'ensemble *I2*, sans capacité sur les dépôts. Ceci peut s'expliquer par le fait que la phase de localisation utilise une technique de résolution s'appuyant sur les capacités, et qu'une valeur fictive doit être utilisée pour contraindre des dépôts quand ces derniers sont sans limitation de charge. Cependant, le *LRGTS* procure des résultats moins dispersés. Il fournit des coûts de solutions jamais très éloignés des meilleurs lorsqu'ils sont apportés par les autres méthodes, et peut procurer des gains allant jusqu'à 12.85% par rapport à $MA|PM$. L'utilisation de méthodes basées sur les connaissances spécifiques des sous-problèmes du *LRP* et de la coopération entre les phases de localisation et de routage semble donc rendre *LRGTS* plus stable que les autres approches de structure générique pouvant s'adapter à tout type de problème.

Enfin, le *LRGTS* est capable de trouver des solutions de grande qualité en des temps de calcul inférieurs à ceux des autres algorithmes avec lesquels il est comparé. Cette faculté montre encore une fois l'intérêt d'exploiter les caractéristiques du problème. En effet, la décomposition en deux phases moins complexes que sont la localisation et le *VRP* multi-dépôt, permet le recours à des méthodes spécifiques, performantes et rapides. Le problème de localisation est de taille très restreinte grâce à l'agrégation des tournées en super-clients. Le problème de tournée quant à lui n'est pas réduit, mais le mécanisme de granularité dans la recherche taboue permet une diminution de nombre de mouvements testés tout en conservant de bons résultats.

Ce travail a été développé en partie lors d'un séjour de 6 semaines début 2005 au Centre de Recherche sur les Transports (CRT) de Montréal, en collaboration avec P. Soriano et A. Ruiz. Une première version du *LRGTS* a été présentée en conférences internationales : 6^{th} *Metaheuristics International Conference* - MIC 2005 (Prins *et al.*, 2005a) et *XXXVI° Annual Conference of the Italian Operations Research Society* - AIRO 2005 (Prins *et al.*, 2005b). La version finale est en révision favorable pour la revue *Transportation Science* (Prins *et al.*, En révision).

Dans ce chapitre et les précédents, différentes méthodes de résolution approchées du problème de localisation-routage ont été proposées. Les résultats sont comparés entre-eux pour permettre une évaluation relative des performances des méthodes proposées. Cependant, une résolution de type exact fournissant des solutions optimales ou au minimum des bornes inférieures sur les plus grosses instances serait intéressante pour mieux juger les résultats obtenus,

tout comme une comparaison avec des résultats de la littérature. L'objectif du chapitre suivant est donc de proposer de telles approches exactes et confrontations.

Chapitre 8

Méthode de coupes

8.1 Introduction

Dans les chapitres précédents, le *LRP* est résolu par des approches de type heuristique. Ceci permet d'obtenir des solutions réalisables en des temps de calcul raisonnables, compte-tenu du fait qu'il s'agit d'un problème combinant des décisions classiquement considérées de niveaux différents et bien plus combinatoire que les sous-problèmes le composant. Cependant, il est difficile de juger de la qualité des résultats obtenus. Le moyen utilisé est alors de comparer entre-eux les coûts procurés par les diverses méthodes développées, la meilleure solution représentant une borne supérieure du problème (puisque l'objectif est une minimisation). Néanmoins, il est impossible de savoir si cette référence est éloignée ou non de l'optimalité. L'idéal serait de connaître cette valeur minimale ou à défaut de disposer d'une borne inférieure de bonne qualité. Pour cela, on peut avoir recours aux approches exactes. Ces dernières sont cependant très gourmandes en temps de calcul pour les problèmes *NP*-difficiles.

L'état de l'art réalisé au chapitre 2 montre que les travaux sur ce genre de techniques pour le *LRP* sont assez sporadiques. Albareda-Sambola *et al.* (2005) proposent une borne inférieure pour un problème sans capacité sur les tournées. Pour le cas généralisé (capacités sur les véhicules et les dépôts), les deux seules méthodes exactes à notre connaissances viennent de Laporte *et al.* (1986) et de Barreto (2004b). Chacun ne considère qu'une flotte limitée par dépôt et utilise une approche par un algorithme de coupes.

Le but dans ce chapitre est de proposer de nouvelles bornes inférieures pour le problème de localisation-routage généralisé. Différentes alternatives sont possibles, comme évoqué au chapitre 2. Il peut s'agir par exemple d'une énumération implicite des solutions ou de relaxations de contraintes du programme linéaire. C'est ici la deuxième approche qui est envisagée car une méthode par séparation et évaluation serait trop coûteuse en temps en calcul, même pour des petites instances. La relaxation adoptée se base sur les formulations du problème en variables à 2 indices présentées au chapitre 3 et rappelées ici en section 8.2. L'idée est de réduire au maximum la formulation initiale, puis d'appliquer une méthode de coupes dont le principe général est également rappelé en section 8.2. Afin d'améliorer les bornes issues de la littérature, de nouvelles familles de contraintes valides sont développées (voir section 8.3). Pour identifier les contraintes non respectées dans une solution du problème relaxé, des

procédures dites de séparation sont nécessaires et sont présentées dans la section 8.4. Les résultats obtenus par l'approche proposée sont donnés dans la section 8.5. Nous tirons profit de la proposition de bornes pour évaluer les solutions provenant de l'ensemble des chapitres. Ainsi, en section 8.6, la qualité des résultats est mesurée avec les différentes bornes disponibles et les coûts fournis par les autres méthodes de la littérature. Enfin, la section 8.7 conclut ce chapitre.

8.2 Présentation de la méthode

8.2.1 Principe des algorithmes de coupes

Les algorithmes de coupes sont une approche de résolution de type polyédrale. Ce genre de méthode apporte généralement de bons résultats sur les problèmes de tournées (voir par exemple les articles de Padberg et Rinaldi (1991) pour le *TSP* et de Augerat *et al.* (1999) ou Lysgaard *et al.* (2004) pour le *VRP*). Le principe est donné au chapitre 2. Nous allons tout de même en rappeler les grandes lignes ici.

Les itérations d'un algorithme de coupes consistent à résoudre un programme linéaire. Au départ, certaines contraintes gênantes du problème traité sont relaxées. Ensuite, pour chaque solution optimale du problème obtenu, on recherche une ou plusieurs contraintes violées pour le problème d'origine. Elles sont alors ajoutées au programme linéaire à résoudre. Pour un problème de minimisation par exemple, ce procédé se poursuit soit jusqu'à l'accession d'une solution réalisable ou bien de coût égal à une borne supérieure, soit jusqu'a ce que l'on ne trouve plus de contraintes violées. Ce dernier cas fournit une borne inférieure. Si cette borne est calculée à chaque nœud d'un algorithme de séparation et évaluation, la méthode s'appelle *branchement et coupes* (*Branch and Cut*).

L'idée ici est de résoudre le *LRP* par des méthodes de coupes dans le but de proposer de nouvelles bornes inférieures au problème. Pour cela, il est nécessaire de se baser sur des formulations mathématiques appropriées. C'est pourquoi, les deux nouveaux programmes linéaires utilisant des variables à 2 indices et proposés dans le chapitre 3 sont utilisées et rappelées dans la suite.

8.2.2 Formulations mathématiques utilisées

Une approche polyédrale est appliquée sur les modèles mathématiques à deux indices proposés au chapitre 3. Ces derniers ont été choisis pour leur formulation se rapprochant de celle utilisée par Lysgaard *et al.* (2004) pour le *VRP*. Ils sont rappelés ici. Les notations nécessaires sont rappelées dans le glossaire en début de mémoire.

8.2.2.1 Formulation 1 :

Dans cette première formulation, les variables de décision suivantes sont utilisées : $y_i = 1$ si et seulement si le dépôt i est ouvert, $x_{ij} = 1$ (i,$j \in V$) si et seulement si un véhicule utilise

l'arête (i,j) une et une seule fois, $x_{ij} = 2$ $(j \in J,\ i \in I)$ si et seulement si un véhicule utilise une même arête (i,j) deux fois (si un véhicule ne sert qu'un seul client j à partir du dépôt i). Les variables x_{ij} avec $(i,j) \in I$ peuvent être exclues.

$$\text{(LRP2)} \quad \min \sum_{(ij)\in E} c_{ij}x_{ij} + \frac{F}{2}\sum_{i\in I}\sum_{j\in J} x_{ij} + \sum_{i\in I} O_i y_i \tag{3.12}$$

sous les contraintes :

$$x(\delta(j)) = 2 \qquad \forall j \in J \qquad 3.13$$

$$x(\delta(S)) \geq 2\lceil D(S)/Q \rceil \qquad \forall S \subseteq J \tag{3.14}$$

$$x_{ij} \leq 2y_i \qquad \forall i \in I \quad \forall j \in J \tag{3.15}$$

$$x(\delta(S \cup \{i\})) \geq 2 \qquad \forall i \in I \quad \forall S \subseteq J \quad | \quad D(S) > W_i \tag{3.16}$$

$$2x(\gamma(S \cup \{j\})) + x_{ij} - x(E(S\ :\ ((J\backslash(S \cup \{j\}))) \cup \{i\}))\ \leq\ 2|S|$$

$$S \subseteq J\backslash\{j\} \qquad \forall i \in I \qquad S \neq \emptyset \tag{3.17}$$

$$x_{ij} \in \{0,1,2\} \qquad \forall i \in I \quad \forall j \in J \tag{3.18}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in E \tag{3.19}$$

$$y_i \in \{0,1\} \qquad \forall i \in I \tag{3.20}$$

Dans ce programme linéaire, la fonction-objectif (3.12) minimise les coûts d'ouverture de tournées et de dépôts ainsi que les coûts pour visiter les clients. Les contraintes (3.13) concernent le degré des variables. Les contraintes (3.14) correspondent au respect des capacités des véhicules. Les contraintes (3.15) imposent que les arêtes ne soient reliées aux dépôts que si ces derniers sont ouverts. Les contraintes (3.16) obligent à respecter les capacités des dépôts. (3.17) sont une nouvelle formulation de contraintes généralisant les propositions faites par Laporte *et al.* (1986) (voir chapitre 2). Elles restreignent l'affectation d'un véhicule à un seul dépôt et sont appelées contraintes *path-f1* car elles interdisent ainsi les chemins entre deux dépôts distincts. (3.18 - 3.20) sont les contraintes d'intégrité des variables.

8.2.2.2 Formulation 2 :

Dans la seconde formulation, les mêmes variables y_i sont utilisées concernant les dépôts, ainsi que les même variables x_{ij} $(i,j \in V)$ pour les arêtes (i,j) utilisées une unique fois. Par contre, de nouvelles variables binaires sont introduites : $w_{ij} = 1$ $(j \in J,\ i \in I)$ pour les arêtes utilisées deux fois (véhicules ne servant qu'un seul client j à partir d'un dépôt i). Les variables x_{ij} avec $(i,j) \in I$ peuvent ici encore être exclues du modèle.

$$\text{(LRP3)} \quad \min \sum_{(ij)\in E} c_{ij}x_{ij} + \sum_{i\in I}\sum_{j\in J} 2c_{ij}w_{ij} + \frac{F}{2}\sum_{i\in I}\sum_{j\in J}(x_{ij} + 2w_{ij}) + \sum_{i\in I} O_i y_i \tag{3.21}$$

sous les contraintes :

$$\sum_{i \in I} 2w_{ij} + x(\delta(j)) = 2 \qquad \forall j \in J \tag{3.22}$$

$$\sum_{i \in I} \sum_{j \in S} 2w_{ij} + x(\delta(S)) \geq 2\lceil D(S)/Q \rceil \qquad \forall S \subseteq J \tag{3.23}$$

$$x_{ij} + w_{ij} \leq y_i \qquad \forall i \in I \quad \forall j \in J \tag{3.24}$$

$$\sum_{t \in I \setminus \{i\}} \sum_{j \in S} 2w_{tj} + x(\delta(S \cup \{i\})) \geq 2 \qquad \forall i \in I \quad \forall S \subseteq J \quad | \quad D(S) > W_i \tag{3.25}$$

$$x(\gamma(S \cup \{j\})) + x_{ij} - x(E(S : (J \setminus (S \cup \{j\})) \cup \{i\})) \leq |S| \qquad \forall j \in J \quad \forall S \subseteq J \setminus \{j\} \quad \forall i \in I \tag{3.26}$$

$$x_{ij} \in \{0,1\} \qquad \forall i,j \in E \tag{3.27}$$

$$w_{ij} \in \{0,1\} \qquad \forall i \in I, \forall j \in J \tag{3.28}$$

$$y_i \in \{0,1\} \qquad \forall i \in I \tag{3.29}$$

Comme précédemment, la fonction-objectif (3.21) minimise l'ensemble des coûts sur les tournées et les dépôts. (3.22) sont les contraintes concernant le degré des variables, (3.23) correspondent au respect des capacités des véhicules, (3.24) imposent que les arêtes ne soient reliées aux dépôts que si ces derniers sont ouverts, (3.25) concernent les capacités des dépôts, dans le même esprit que pour les contraintes (3.23). Les contraintes (3.26) restreignent l'affectation d'un véhicule à un seul dépôt, elles sont appelées cette fois-ci contraintes *path-f2* car tout comme dans (LRP2), elles interdisent en même temps les chemins entre deux dépôts distincts, et (3.27) - (3.29) sont les contraintes d'intégrité des variables.

A partir de ces formulations, il est possible d'utiliser une approche de résolution par un algorithme de coupes, mais pour rendre efficace cette méthode, il est important de définir des familles de contraintes valides pour problème initial.

8.3 Familles d'inégalités valides pour le *LRP*

Le principe d'un algorithme de coupes est présenté dans la section précédente. L'idée est de partir d'une formulation relaxée du problème de départ et d'ajouter les contraintes violées dans la solution de la formulation courante. L'idéal est d'introduire des contraintes représentant des facettes du polyèdre de solutions réalisables, car celles-ci restreignent au maximum l'espace des solutions du programme linéaire à résoudre. Cependant, il est difficile de trouver de telles contraintes.

Afin d'augmenter les chances d'identifier des contraintes violées et améliorer la formulation d'une itération à la suivante, il est intéressant de développer des familles d'inégalités valides (c'est-à-dire qui respectent des contraintes du problème de départ) pour lesquelles des procédures d'identification, permettant de détecter si certaines de ces inégalités ne sont pas respectées, peuvent être développées.

La plupart des contraintes classiques pour le *VRP* avec capacités (par exemples celles de Lysgaard *et al.* (2004)) sont également valides pour le problème traité ici. Mais cette section en présente de nouvelles développées spécifiquement pour le *LRP*.

8.3.1 Amélioration des contraintes de capacité

Le premier type de contraintes, souvent non respecté lors de la relaxation des inégalités proposées, concerne les capacités. Afin de renforcer le programme linéaire, il est intéressant d'essayer de les identifier. Le recours à de nouvelles familles de contraintes est alors intéressant. Deux types sont proposés :

- *y-capacité*, impliquant les tournées reliées à un dépôt i (les dépôts étant représentés par les variables y) ;
- *w-capacité*, utilisé pour la seconde formulation et nécessitant les variables w.

Ces dernières sont développées ci-après.

8.3.1.1 Contraintes de type *y-capacité*

Les contraintes de type *y-capacité* sont utilisées pour la première formulation du problème. Soit $k(S) = \lceil D(S)/Q \rceil$, représentant le nombre minimum de véhicules nécessaire pour servir S, et I_j les dépôts auxquels est relié le client j. On a alors :

$$\sum_{j \in S'} \left(\sum_{i \in I_j} (x_{ij} - y_i) \right) + x\left(\gamma(S)\right) \leq |\, S \,| - 1 \qquad S \subseteq J, S' \subset S \;/\; |\, S' \,| \leq |\, S \,| - 1,$$
$$k(S) = 1,\, I_j \subseteq I \text{ pour } j \in S' \qquad (8.1)$$

Ces contraintes peuvent se généraliser pour toute valeur $k(S) \geq 1$ de la manière suivante :

$$\sum_{j \in S'} \left(\sum_{i \in I_j} (x_{ij} - y_i) \right) + x\left(\gamma(S)\right) \leq |\, S \,| - k(S) \qquad S \subseteq J, S' \subseteq S \;/\; |\, S' \,| \leq |\, S \,| - k(S),$$
$$k(S \backslash S') = k(S), I_j \subseteq I \text{ pour } j \in S' \qquad (8.2)$$

Elles caractérisent le fait qu'un dépôt ne peut être ouvert partiellement comme l'illustre la figure 8.1.

Preuve de validité

Soit (x,y) une solution réalisable du problème. Notons que $\sum_{i \in I_j} (x_{ij} - y_i) \leq 1$ pour tout $j \in S'$ et que les contraintes de capacité assurent le fait que $x\left(\gamma(S)\right) \leq |S| - k(S)$. Ainsi, la seule possibilité de violation de (8.2) est quand $\sum_{i \in I_j} (x_{ij} - y_i) = 1$ pour des clients $j \in S'$, mais alors $x_{ij} = 2$ pour un dépôt $i \in I_j$, donc le client j est servi seul dans une tournée du dépôt i.

$$\sum_{j\in S'}\left(\sum_{i\in I_j}(x_{ij}-y_i)\right)+x\left(\gamma(S)\right)\leq|S|-k(S) \qquad S\subseteq J,\ S'\subseteq S\ /\ |S'|\leq|S|-k(S),$$
$$k(S\backslash S')=k(S),\ I_j\subseteq I \text{ for } j\in S'$$

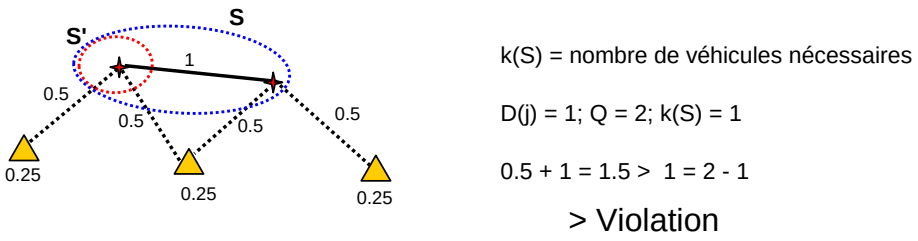


FIG. 8.1 – *Exemple de violation des contraintes de type y-capacité*

Supposons que dans la solution réalisable, $\sum_{j\in S'}\left(\sum_{i\in I_j}(x_{ij}-y_i)\right)=k\leq|S'|$. Ceci implique qu'il existe $S''\subseteq S'$ avec $|S''|=k$ tel que chaque client de S'' est servi seul dans une tournée. De ce fait, on a

$$x\left(\gamma(S)\right)=x\left(\gamma(S\backslash S'')\right)\leq\underbrace{|S\backslash S''|}_{=|S|-k}-\underbrace{k(S\backslash S'')}_{=k(S\backslash S')=k(S)}=|S|-k-k(S) \qquad \diamondsuit$$

8.3.1.2 Contraintes de type *w-capacité*

Les contraintes de type *w-capacité* sont similaires aux précédentes mais sont utilisées pour la seconde formulation du problème et peuvent s'écrire de la manière suivante :

$$\sum_{i\in I}\sum_{j\in S'}w_{ij}+x(\gamma(S))\leq|S|-1 \quad \forall S\subseteq J,\quad (D(S)\leq Q),\quad S\subseteq S',\quad |S'|=|S|-1$$

Preuve de validité

Soit $\sum_{i\in I}\sum_{j\in S'}w_{ij}=p\leq|S'|=|S|-1$, alors p clients appartenant à S sont servis individuellement par des véhicules et aucune autre arête n'est incidente à eux. Ainsi, $x(\gamma(S))\leq|S|-p-1$ et l'inégalité est respectée. $\diamondsuit$

$$\sum_{i\in I'} x_{ij} + 2x\left(\gamma(S\cup\{j,l\})\right) + \sum_{k\in I\setminus I'} x_{kl} \leq 2|S| + 3$$
$$j,l \in J,\ S \subseteq J\setminus\{j,l\},\ S \neq \emptyset, D(S) \leq Q, I' \subset I$$

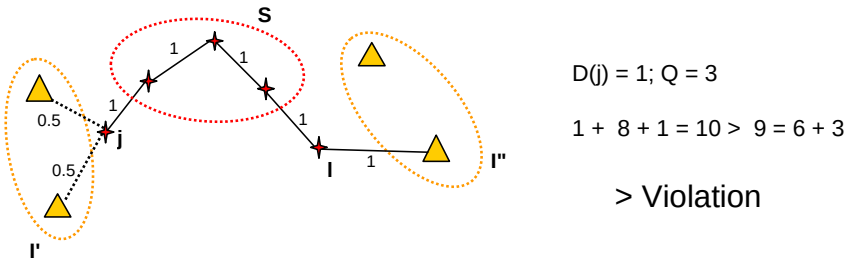


FIG. 8.2 – *Exemple de violation des contraintes de type chain2-f1*

8.3.2 Alternatives aux contraintes de type *path*

Laporte *et al.* (1986) ont proposé les inégalités appelées *chain barring* (voir chapitre 2). Ces dernières peuvent cependant facilement être améliorées, et adaptées pour les deux formulations appliquées ici, par les contraintes appelées path-f1 (3.17) et path-f2 (3.26) ou les nouvelles proposées dans la suite.

8.3.2.1 Formulation 1

Pour la formulation 1, les adaptations proposées peuvent être appelée *chain2-f1* :

$$\sum_{i\in I'} x_{ij} + 2x\left(\gamma(S\cup\{j,l\})\right) + \sum_{k\in I\setminus I'} x_{kl} \leq 2|S| + 3$$
$$j,l \in J, \quad S \subseteq J\setminus\{j,l\}, \quad S \neq \emptyset, \quad D(S) \leq Q, \quad I' \subset I \quad (8.3)$$

et si $S = \emptyset$:

$$\sum_{i\in I'} x_{ij} + 3x_{lj} + \sum_{k\in I\setminus I'} x_{kl} \leq 4 \quad j,l \in J, \quad I' \subset I \quad (8.4)$$

Ces dernières ont pour caractéristique de travailler non pas sur le fait qu'une tournée doivent être affectée à un unique dépôt mais, de manière plus générale, à un unique sous-ensemble de dépôts comme l'illustre la figure 8.2.

Preuve de validité

Pour (8.3), $S \neq \emptyset$ et $x(\gamma(S \cup \{j,l\})) \leq |S| + 1$. De plus :

- si $x(\gamma(S \cup \{j,l\})) = |S| + 1$
  alors, tous les clients de $(S \cup \{j,l\})$ sont connectés et appartiennent à une même tournée qui doit être reliée soit à un dépôt de I', soit à un dépôt de $I \backslash I'$. Donc, $\sum_{i \in I'} x_{ij} = 1$ ou $\sum_{k \in I \backslash I'} x_{kl} = 1$, mais pas les deux sinon il existerait un chemin entre les deux sous-ensembles de dépôts, ce qui n'est pas permis.
  De plus, on a :
  $$\sum_{i \in I'} (x_{ij}) \leq 1 \quad \text{et} \quad \sum_{k \in I \backslash I'} (x_{kl}) \leq 1$$
  La contrainte est donc bien respectée.
- si $x(\gamma(S \cup \{j,l\})) = |S|$
  alors, tous les clients de $(S \cup \{j,l\})$ sauf un sont connectés et appartiennent à une même tournée. Si ce client n'est ni j ni l, alors $\sum_{i \in I'} x_{ij} \leq 1$ et $\sum_{k \in I \backslash I'} x_{kl} \leq 1$ et l'inégalité est respectée. Sinon, soit $\sum_{i \in I'} x_{ij} \leq 2$ et $\sum_{k \in I \backslash I'} x_{kl} \leq 1$, soit $\sum_{i \in I'} x_{ij} \leq 1$ et $\sum_{k \in I \backslash I'} x_{kl} \leq 2$, et la contrainte est encore une fois bien respectée.
- si $x(\gamma(S \cup \{j,l\})) < |S|$, la contrainte est toujours respectée.

Dans le cas où $S = \emptyset$ (contrainte 8.4) :

- si $x_{lj} = 1$
  alors, une tournée relie les clients l et j et on a :
  $$\sum_{i \in I'} (x_{ij}) \leq 1 \quad \text{et} \quad \sum_{k \in I \backslash I'} (x_{kl}) \leq 1$$
  Si $\sum_{i \in I'} (x_{ij}) = 1$ alors la tournée est reliée à I' et $\sum_{k \in I \backslash I'} (x_{kl}) = 0$. Si $\sum_{k \in I \backslash I'} (x_{kl}) = 1$ alors la tournée est reliée à $I \backslash I'$ et $\sum_{i \in I'} (x_{ij}) = 0$.
  La contrainte est donc toujours respectée.
- si $x_{lj} = 0$
  au plus $\sum_{i \in I'} (x_{ij}) = 2$ et $\sum_{k \in I \backslash I'} (x_{kl}) = 2$ et la contrainte est naturellement respectée.
  $\diamondsuit$

Une autre alternative pour les contraintes de type *path*, qui peuvent être nommée *path2-f1*, est la suivante :

$$2x\left(\gamma(S \cup \{j\})\right) + x_{ij} + \frac{1}{2} x\left(E(S \cup \{j\} : I \backslash \{i\})\right) \leq 2|S| + 1 \qquad i \in I,\ S \subseteq J,\ j \in J \backslash S \quad (8.5)$$

L'intérêt ici est que ce type de contraintes peut n'impliquer qu'un seul client (si $S = \emptyset$)

Preuve de validité

Soit (x,y) une solution réalisable du problème. Il est possible de considérer différents cas, selon la valeur de $x\left(\gamma(S \cup \{j\})\right) = A$ et $x\left(E(S \cup \{j\} : I \backslash \{i\})\right) = B$:

- D'abord, il est impossible d'avoir simultanément $A = |\,S\,|$ et $x_{ij} = 2$, car la dernière condition implique que le client j est l'unique client d'une tournée partant du dépôt i et donc $x\left(\gamma(S \cup \{j\})\right) = x\left(\gamma(S)\right) \leq |\,S\,| - 1$.
- Si $A = |\,S\,|$ et $x_{ij} = 1$, alors il existe un chemin partant du dépôt i et visitant tous les clients de $S \cup j$. Ainsi, ce chemin ne peut être affecté à aucun autre dépôt et $B = 0$.
- Si $A = |\,S\,|$ et $x_{ij} = 0$, alors il existe un chemin passant par tous les clients de $S \cup \{j\}$ mais ne reliant pas directement le clients j au dépôt i, et ce chemin peut donc être affecté à un autre dépôt, impliquant que $B \leq 2$.
- Si $A = |\,S\,| - 1$ et $x_{ij} = 2$, alors le client j est l'unique client d'une tournée partant du dépôt i et il existe un chemin reliant tous les clients de S. Ce chemin peut être affecté au dépôt i ou non, donc $B \leq 2$.
- Si $A = |\,S\,| - 1$ et $x_{ij} = 1$, alors il existe deux chemins, l'un partant du dépôt i et servant le client j en premier, éventuellement reliant j à d'autres clients de S, et l'autre reliant les clients restant de S. Le premier devant finir au dépôt i, on obtient $B \leq 2$.
- Si $A = |\,S\,| - 1$ et $x_{ij} = 0$, alors il existe deux chemins, l'un servant le client j, éventuellement reliant j à d'autres clients de S, et l'autre reliant les clients restants de S. Chacun peut être affecté à un autre dépôt que i, impliquant que $B \leq 4$.
- Si $A = |\,S\,| - 2$ et $x_{ij} = 2$, alors le client j est l'unique client d'une tournée partant du dépôt i et il existe deux chemins reliants tous les clients de S, donc $B \leq 4$.
- Si $A = |\,S\,| - 2$ et $x_{ij} = 1$, alors il existe trois chemins, l'un partant du dépôt i, servant d'abord le client j, éventuellement reliant j à d'autres clients de S, et les deux autres reliants les clients restants de S. Le premier devant retourner au dépôt i, cela implique que $B \leq 4$.
- Si $A = |\,S\,| - 2$ et $x_{ij} = 0$, alors il existe trois chemins, l'un servant le client j, éventuellement reliant j à d'autres clients de S, et les deux autres reliants les clients restants de S. Ces chemins peuvent être affectés à tout dépôt différent de i, donc $B \leq 6$.
- Et ainsi de suite, si $A = |\,S\,| - k$ avec $3 \leq k \leq |\,S\,|$, alors $B \leq 2k$ (quand $x_{ij} = 1$ ou 2) ou $B \leq 2k + 2$ (quand $x_{ij} = 0$).

♢

8.3.2.2 Formulation 2

Pour la formulation 2, les adaptations des contraintes de Laporte *et al.* (1986) peuvent être appelée *chain2-f2* :

$$\sum_{i \in I'} (w_{ij} + x_{ij}) + x\left(\gamma(S \cup \{j,l\})\right) + \sum_{k \in I"} (w_{kl} + x_{kl}) \leq |\,S\,| + 2$$
$$\{j,l\} \in J, \quad S \subseteq J \backslash \{j,l\}, \quad S \neq \emptyset, \quad I' \cap I" = \emptyset \quad (8.6)$$

Tout comme dans la formulation 1, les contraintes *chain2-f2* ont pour caractéristique de

travailler non pas sur le fait qu'une tournée doivent être affectée à un unique dépôt mais, de manière plus générale, sur un unique sous-ensemble de dépôts.

Preuve de validité

On a :

$$\sum_{i\in I'}(x_{ij}+w_{ij})\leq 1, \qquad \sum_{k\in I"}(x_{kl}+w_{kl})\leq 1, \quad \text{et} \quad x(\gamma(S\cup\{j,l\}))\leq |S|+1$$

Si $x(\gamma(S\cup\{j,l\}))=|S|+1$, et $\sum_{i\in I'}(x_{ij}+w_{ij})=1$, alors un véhicule peut servir tous les clients de $S\cup\{j,l\}$ à partir d'un dépôt de I'; ainsi, ce véhicule devant retourner au dépôt de départ, $\sum_{k\in I"}(x_{kl}+w_{kl})=0$ est respectée. Si $x(\gamma(S\cup\{j,l\}))\leq|S|$, l'inégalité est naturellement respectée. De plus, l'inégalité est valide quand $S=\emptyset$. ♢

L'intérêt ici est que ce type de contraintes peut n'impliquer qu'un seul client (si $S=\emptyset$) Comme précédemment, une autre alternative est possible tout en n'impliquant qu'un seul client (si $S=\emptyset$). Les inégalités concernées sont appelées contraintes *path2-f2* :

$$x\left(\gamma(S\cup\{j\})\right)+x_{ij}+w_{ij} + \frac{1}{2}(x(E(S\cup\{j\}:I\backslash\{i\}))+2w(E(S\cup\{j\}:I\backslash\{i\})))\leq |\,S\,|+1 \qquad i\in I,\quad S\subseteq J,\quad j\in J\backslash S \quad (8.7)$$

Preuve de validité

Soit (x,w,y) une solution réalisable du problème, différents cas de figures sont envisageables, selon les valeurs de $x\left(\gamma(S\cup\{j\})\right)=A$,
et de $x\left(E(S\cup\{j\}:I\backslash\{i\})\right)+2w\left(E(S\cup j:I\backslash\{i\})\right)=B$:

- Si $x_{ij}=1$ et $w_{ij}=0$ alors la contrainte est du même type que précédemment avec *path2-f1* et est donc valide. Sinon, si $x_{ij}=0$ et $w_{ij}=1$, alors le client j est relié au dépôt i et est le seul client de la tournée. Dans le cas où $x_{ij}=0$ et $w_{ij}=0$, alors j n'est pas relié au dépôt.
- Si $A=|\,S\,|$, alors tous les clients appartenant à $S\cup\{j\}$ sont servis par le même véhicule et $w_{ij}=1$ est impossible.
- Si $A=|\,S\,|$, $x_{ij}=0$ et $w_{ij}=0$, alors tous les clients de $S\cup j$ sont servis par le même véhicule affecté à un dépôt différent de i, donc $B=2$.
- Si $A=|\,S\,|-1$, $x_{ij}=0$ et $w_{ij}=1$, alors le client j est relié au dépôt i et est le seul client de la tournée, et il existe un chemin joignant tous les clients de $S\cup\{j\}$, ce dernier pouvant être relié au dépôt i ou non, donc $B\leq 2$.
- Si $A=|\,S\,|-1$, $x_{ij}=0$ et $w_{ij}=0$, alors soit tous les clients de S appartiennent à la même tournée et ne sont pas reliés à j, soit il existe deux tournées et l'une passe par le client j. Ces tournées peuvent être affectées au dépôt i ou non, donc $B\leq 4$.

- Si $A = |S| - 2$, $x_{ij} = 0$ et $w_{ij} = 1$, alors le client j est relié au dépôt i et est le seul client de la tournée, et il existe au plus deux chemins dans S, partant du dépôt i ou non. Si chacun de ces chemins débutent à un autre dépôt que i, il doit finir à ce même dépôt et $B \leq 4$.
- Si $A = |S| - 2$, $x_{ij} = 0$ et $w_{ij} = 0$, alors il y a trois tournées dont une servant le client j seul ou non. Ces tournées peuvent être affectées au dépôt i ou non, donc $B \leq 6$.
- Et ainsi de suite pour les autres cas. ♢

8.3.3 Inégalités de recouvrement pour les dépôts

La contrainte suivante, de type sac-à-dos, est valide puisqu'il s'agit de couvrir la demande des clients par les dépôts :

$$\sum_{i \in I} W_i y_i \geq D(J)$$

Il est alors possible d'introduire des inégalités de recouvrement (*Cover inequalities*) pour les dépôts :

$$\sum_{i \in C} y_i \geq 1$$

avec C une couverture de la capacité résiduelle des dépôts (égale à $\sum_{i \in I} W_i - D(J)$), c'est-à-dire un ensemble de dépôts C tel que $\sum_{i \in C} W_i > \sum_{i \in I} W_i - D(J)$ ou encore :

$$\sum_{i \in I \backslash C} W_i \leq D(J)$$

(on recherche C tel que les dépôts de $I \backslash C$ ne sont pas suffisant pour répondre à la demande totale des clients, impliquant $\sum_{i \in C} y_i \geq 1$)

Preuve de validité

Une inégalité forte et valide pour

$$S = \{x \in B^n : \sum_{j \in N} a_j x_j \leq b\}$$

est

$$\sum_{j \in C} x_j \leq |C| - 1$$

avec $C \subseteq N$ une couverture si $\sum_{j \in C} a_j > b$

Dans notre cas, concernant les dépôts, il faut

$$S = \{y : \sum_{i \in I} W_i y_i \geq D(J)\}$$

donc

$$S = \{y : \sum_{i \in I} W_i(1 - y_i) \leq \sum_{i \in I} W_i - D(J)\}$$

Ainsi, les inégalités obtenues sont de la forme :

$$\sum_{i \in C}(1 - y_i) \leq |C| - 1$$

ou

$$\sum_{i \in C} y_i \geq 1$$

avec C une couverture de la capacité restante des dépôts (égale à $\sum_{i \in I} W_i - D(J)$). ♢

8.4 Algorithmes de coupes

8.4.1 Formulations initiales

Au départ, dans nos algorithmes de coupes, seules quelques contraintes sont introduites dans le programme linéaire du modèle. Pour la formulation 1, ce sont :

1. Les contraintes de degré : $x(\delta(j)) = 2 \qquad \forall j \in J$
2. Les contraintes d'ouverture de dépôts : $x_{ij} \leq 2y_i \qquad \forall i \in I \quad \forall j \in J$,
3. Le nombre minimum de véhicules : $\sum_{i \in I} x(\delta(i)) \geq 2\lceil D(J)/Q \rceil$
4. Une capacité suffisante sur les dépôts ouverts : $\sum_{i \in I} W_i y_i \geq D(J)$
5. Le nombre minimum de dépôts : $\sum_{i \in I} y_i \geq Mindep$ où $Mindep = \min\{k : \sum_{i=1..k} W_i \geq D(J)\}$, en supposant que $W_1 \geq W_2 \geq \ldots \geq W_m$
6. Les bornes : $0 \leq x_{ij} \leq 1$, $\forall (i,j) \in E \backslash \delta(I)$; $0 \leq x_{ij} \leq 2$, $\forall i \in I$, $\forall j \in J$.
7. Les bornes : $0 \leq y_i \leq 1$, $\forall i \in I$ (version 1 de l'algorithme) ou $y_i \in \{0,1\}$, $\forall i \in I$ (version 2 de l'algorithme), voir la version utilisée dans la section 8.5.

et pour la formulation 2 :

1. Les contraintes de degré : $\sum_{i \in I} 2w_{ij} + x(\delta(j)) = 2 \qquad \forall j \in J$
2. Les contraintes d'ouverture de dépôts : $x_{ij} + w_{ij} \leq y_i \qquad \forall i \in I \qquad \forall j \in J$
3. Le nombre minimum de véhicules : $\sum_{i \in I}(x(\delta(i)) + 2w(\delta(i))) \geq 2\lceil D(J)/Q \rceil$
4. Une capacité suffisante sur les dépôts ouverts : $\sum_{i \in I} W_i y_i \geq D(J)$
5. Le nombre minimum de dépôts : $\sum_{i \in I} y_i \geq Mindep$ où $Mindep = \min\{k : \sum_{i=1..k} W_i \geq D(J)\}$, en supposant que $W_1 \geq W_2 \geq \ldots \geq W_m$
6. Les bornes : $0 \leq x_{ij} \leq 1$, $\forall (i,j) \in E$; $w_{ij} \leq 1$, $\forall i \in I$, $\forall j \in J$.
7. Les bornes : $0 \leq y_i \leq 1$, $\forall i \in I$ (version 1 de l'algorithme) ou $y_i \in \{0,1\}$, $\forall i \in I$ (version 2 de l'algorithme), voir la version utilisée dans la section 8.5.

Pour réaliser un algorithme de coupes (version 1 et 2), le principe est de partir de ces formulations. Ensuite, à chaque itération, un nombre maximum *MaxConst* de contraintes par famille d'inégalités valides est ajouté au programme linéaire. Pour cela, il faut avoir recours à des procédures de séparation, comme celles décrites dans les paragraphes suivants. Le résultat donne en général une borne inférieure au problème à moins d'obtenir une solution réalisable ou égale à une borne supérieure.

Mais une telle borne peut également être calculée au sein d'un algorithme exact de branchement et coupes (version 3). Le principe général d'une telle approche peut être trouvée dans Padberg et Rinaldi (1991) par exemple. Le branchement consiste à séparer en sous-problèmes l'arbre de recherche. La borne inférieure à chaque nœud de cet arbre est calculée en faisant appel à l'algorithme de coupes dans lequel toutes les contraintes d'intégrité sont relaxées.

8.4.2 Procédures de séparation

Soit $(\bar{x},\bar{y})$ (ou $(\bar{x},\bar{w},\bar{y})$ pour la formulation 2) une solution optimale du modèle relaxé, composé d'un sous-ensemble de contraintes. On peut définir un graphe valué $G(\bar{x}) = (V,\bar{E},\bar{X})$ correspondant à la solution, avec $\bar{E}$ les arêtes $(i,j) \in E$ telles que $\overline{x_{ij}} > 0$, et $\overline{X}$ la valuation des arêtes correspondant à la valeur prise par les variables non nulles $(\bar{x},\bar{y})$ (ou $(\bar{x},\bar{w},\bar{y})$) dans la solution. $G(\bar{x})$ est appelé *graphe-support*.

8.4.2.1 Séparation pour les contraintes de capacité

Les premières coupes identifiables concernent les contraintes de capacité : (3.14) et (3.23). Les procédures de séparation utilisées par Augerat *et al.* (1999) et Lysgaard *et al.* (2004) sont ici valides et peuvent être utilisées. Des adaptations sont cependant nécessaires en particulier pour identifier les formulations de type *y-capacité* et *w-capacité*.

A partir d'une solution optimale $\bar{x}$ du programme linéaire relaxé, l'idée est de construire le graphe-support $G(\bar{x})$ et un sous-ensemble de clients S prometteur pour vérifier la validité des contraintes de capacité : $\bar{x}(\delta(S))$, $\sum_{i\in I}\sum_{j\in S} 2w_{ij} + x(\delta(S))$, ou les versions améliorées, selon les formulations. Un sous-ensemble de clients S prometteur est soit :

- très petit puisque si $\bar{x}(\delta(S)) < 2$ (ou $\sum_{i\in I}\sum_{j\in S} 2w_{ij} + x(\delta(S)) < 2$), la contrainte est violée.
- de telle sorte que $D(S)$ soit aussi proche que possible d'un multiple de Q.

Pour trouver S, des heuristiques sont utilisées. La première recherche les composantes connexes dans le graphe de la solution. S'il en existe non reliées à des dépôts, alors il y a une violation. Les composantes connexes dans le graphe sans dépôts sont également identifiées et leur demande est vérifiée pour s'assurer qu'elle ne dépasse pas à la capacité d'un véhicule.

La seconde heuristique est basée sur un *graphe contracté*. Les clients (i,j) reliés par une arête $\bar{x}_{ij} = 1$ sont fusionnés et la demande donnée au nœud résultant k (*super-nœud*) est égale à la somme des demandes des deux nœuds originaux. Les arêtes reliant des nœuds disparus par agrégation à un autre nœud l sont remplacés par une arête (k,l) de valeur $x_{kl} = x_{il} + x_{jl}$.

Pour la seconde formulation, avec les variables w, le même principe est appliqué si l est un dépôt. Si le super-nœud obtenu a une demande supérieure à Q ou si une arête a une valeur plus grande que 1 alors des contraintes sont violées. Quand toutes les arêtes du graphes ont une valeur supérieure ou égale à 1 dans le graphe contracté, il peut être prouvé qu'il existe au moins une violation de capacité dans le graphe supporté initial de la solution.

Enfin, la dernière heuristique est une méthode taboue qui commence avec un sous-ensemble initial $S = \{v\}$,$\{v\} \in J$, pour alterner entre des phases d'expansion et d'échanges. En phase d'expansion, un nœud v est récursivement ajouté à S de telle sorte que $v \in J \backslash S$ et $\bar{x}(E(S : \{v\}))$ soit différent de zéro et maximum (ou $\bar{x}(E(S \cup \{v\} : \overline{S \cup \{v\}}))$ minimum), et ceci jusqu'à ce que plus aucun nœud v ne répondant à ces conditions n'existe. En cas d'égalité, le nœud v de demande maximale est sélectionné. Une liste limitée de candidats est utilisée pour limiter les possibilités. La phase d'échanges commence quand $D(S)$ est proche d'un multiple de Q ($D(S) < Q$). Un nœud v est alors ajouté ou retiré de S. Ce dernier est choisi de telle sorte que $\bar{x}(E(S : \{v\}))$ soit minimisé. Le nœud ajouté est choisi afin de garder une valeur de $D(S)$ restant proche d'un multiple de Q. Un principe de mise en mémoire des derniers nœuds impliqués dans les mouvements est mis en place afin de les rendre tabous.

En ce qui concerne les capacités des dépôts, il est possible de vérifier dans les heuristiques précédentes les contraintes correspondantes, $x(\delta(S \cup \{i\})) \geq 2$, pour tout ensemble S ou tout ensemble connexe dépassant la capacité d'un dépôt qu'il contient.

8.4.2.2 Séparation pour les contraintes de type Path

Dans la formulation 1, cela concerne les contraintes (3.17), (8.3), (8.4) et (8.5). Dans la seconde, il s'agit de (3.26), (8.6) et (8.7). Elles peuvent être vérifiées facilement dans les heuristiques précédentes pour les contraintes de capacité, dès qu'un ensemble S est identifié.

Plus spécifiquement, d'autres procédures sont proposées comme pour les contraintes *chain2-f1* :

$$\sum_{i \in I'} x_{ij} + 2x\left(\gamma(S \cup \{j,l\})\right) + \sum_{k \in I \backslash I'} x_{kl} \leq 2|S| + 3$$

$$j,l \in J, \quad S \subseteq J \backslash \{j,l\}, \quad S \neq \emptyset, \quad D(S) \leq Q, \quad I' \subset I$$

et

$$\sum_{i \in I'} x_{ij} + 3x_{lj} + \sum_{k \in I \backslash I'} x_{kl} \leq 4 \quad j,l \in J, \quad I' \subset I$$

En utilisant le graphe contracté, si un nouveau super-nœud représentant un ensemble T de nœuds du graphe-support initial est incident à plus d'un dépôt et vérifie $x(E(T : I)) > 1$, alors une procédure spécifique est appelée. Celle-ci tente d'identifier deux nœuds l et j dans T, avec $S = T \backslash \{j,l\}$, et un sous-ensemble de dépôts I', pour lesquels une contrainte de type *chain2-f1* est violée.

Une autre procédure, exacte celle-ci, est également proposée pour les contraintes *chain2-f1*. Soit une solution satisfaisant toutes les contraintes initiales et les contraintes de capacité

du problème. Comme ces dernières induisent les contraintes d'élimination des sous-tours, il en découle que pour tout ensemble $T \subseteq J$, $x(\gamma(T)) \leq |T| - 1$.

La procédure exacte peut alors s'appliquer à chaque paire de clients j et l contenus dans une même composante connexe du graphe-support et connectés directement à au moins un dépôt (excluant les clients servis seuls dans une tournée). A l'évidence, les composantes connexes incluant zéro ou un seul dépôt ne nécessitent pas de traitement. Il serait également possible de restreindre l'étude aux paires de clients les plus prometteuses.

Lorsqu'une paire de clients j et l est choisie, les différentes étapes à suivre sont les suivantes :

1. Si $\sum_{i \in I} x_{ij} = 0$ ou $\sum_{i \in I} x_{il} = 0$, l'inégalité (8.3) est forcément satisfaite. L'étude peut s'arrêter pour cette paire de clients.
2. Soit $A(j,l,I') = \sum_{i \in I'} x_{ij} + \sum_{k \in I \setminus I'} x_{kl}$. Il est alors intéressant de choisir I' de telle sorte que $A(j,l,I')$ soit maximisé, ce qui est indépendant du choix de l'ensemble de clients S. Soit $I(j,l) = \{i \in I : x_{ij} \geq x_{il}\}$, alors $A(j,l,I')$ est maximum quand $I' = I(j,l)$, valeur donnée dans la suite à I'.
3. De plus, quand $I' = I$, ou $I' = \emptyset$, alors (8.3) n'est pas violée quelque soit S. En effet, si $I' = I$, comme $x(\gamma(S \cup \{j,l\})) = x(E(j : S \cup \{l\})) + x(\gamma(S \cup \{l\}))$, l'inégalité devient :

$$\sum_{i \in I} x_{ij} + 2x(\gamma(S \cup \{j,l\})) = \left(\sum_{i \in I} x_{ij} + x(E(j : S \cup \{l\})) \right) + x(\gamma(S \cup \{l\})) +$$

$$+ x(\gamma(S \cup \{j,l\})) \leq 2 + |S| + (|S| + 1) = 2|S| + 3$$

   Cette dernière est respectée car la partie entre parenthèses correspond au degré du nœud j, et les contraintes de sous-tours dans $S \cup \{l\}$ et $S \cup \{j,l\}$ sont respectées.
   La validation pour le cas où $I' = \emptyset$ est similaire.
   Il reste maintenant à regarder ce qui se passe dans les configurations suivantes : $I' = I(j,l)$, avec $I' \neq I$ et $I' \neq \emptyset$.
4. A ce niveau, il est facile de vérifier le cas particulier où $S = \emptyset$:
   $\sum_{i \in I'} x_{ij} + \sum_{k \in I \setminus I'} x_{kl} + 3x_{jl} \leq 4$.
   Si la contrainte est violée, la procédure peut s'arrêter. Sinon, il faut s'attarder sur le cas plus général : $S \neq \emptyset$.

Remarques :
Comme $\sum_{i \in I'} x_{ij} \leq 2$ et $\sum_{k \in I \setminus I'} x_{kl} \leq 2$, alors $A(j,l,I') \leq 4$.
D'autre part, $x(\gamma(S \cup \{j,l\})) = x(\gamma(S)) + x_{jl} + x(E(S : \{j,l\}))$, et $2x_{jl} + x(E(S : \{j,l\}) + A(j,l,I') \leq 4$ puisque les degrés de j et l sont égaux à 2.
Ainsi, pour certaines valeurs de $A(j,l,I')$, la contrainte (8.3) ne sera pas violée, comme dans les trois cas suivants :

- Si $A(j,l,I') = 4$, car alors, de par les contraintes de degré sur j et l, $x(\gamma(S \cup \{j,l\})) = x(\gamma(S)) \leq |S| - 1$.
- Si $3 \leq A(j,l,I') < 4$, soit $A(j,l,I') = 3 + \varepsilon$ avec $\varepsilon \geq 0$. Sachant que $x_{jl} + x(E(S : \{j,l\}) \leq 4 - A(j,l,I')$, alors :
  $x(\gamma(S \cup \{j,l\})) = x(\gamma(S)) + x_{jl} + x(E(S : \{j,l\}) \leq x(\gamma(S)) + 4 - A(j,l,I')$.

Ainsi, $2x\left(\gamma(S \cup \{j,l\})\right)+A(j,l,I') \leq 2x\left(\gamma(S)\right)+8-2A(j,l,I')+A(j,l,I') = 2x\left(\gamma(S)\right)+8-A(j,l,I') \leq 2|S|-2+8-3-\varepsilon = 2|S|+3-\varepsilon$, impliquant le respect de la contrainte (8.3).

- Si $A(j,l,I') \leq 1$, car les contraintes d'élimination des sous-tours $S \cup \{j,l\}$ sont satisfaites.

De ce qui précède, il en découle que les contraintes (8.3) ne peuvent être violées que quand $1 < A(j,l,I') < 3$.

5. Si $1 < A(j,l,I') < 3$ n'est pas respecté, la procédure peut s'arrêter.

Remarques :

Il faut trouver un ensemble $S \subseteq J$ tel que $2x\left(\gamma(S \cup \{j,l\})\right) + A(j,l,I') > 2|S|+3$ ou montrer qu'aucun n'existe. Ceci est équivalent à $x\left(\delta(S \cup \{j,l\})\right) < A(j,l,I')+1$, puisque $2|S|+4 = 2x\left(\gamma(S \cup \{j,l\})\right) + x\left(\delta(S \cup \{j,l\})\right)$.

Il faut donc chercher $T \subseteq J$ de telle sorte que $j,l \in T$ et $x(\delta(T))$ soit aussi petit que possible. Pour cela, une idée est de trouver une coupe minimale par le biais d'un algorithme de flot maximum, en s'assurant que l'ensemble T définissant la coupe contienne bien les nœuds j et l mais aucun dépôt. Pour l'algorithme de flot, on peut utiliser celui de Ford-Fulkerson et partir du graphe-support en remplaçant les arêtes par deux arcs opposés de capacité égale à la valuation de l'arête correspondante.

6. Sinon, il est nécessaire de construire un graphe auxiliaire obtenu à partir du graphe-support, en y ajoutant deux super-nœuds s (source) et t (destination). Le nœud s est relié à j et l par des arêtes de capacité M (suffisamment grosse), et t est connecté à tous les dépôts avec également des arêtes de capacité M. Toutes les autres arêtes du graphe-support sont conservées. Dans ce graphe auxiliaire, un algorithme de flot maximum (Ford-Fulkerson) est utilisé entre s et t et la coupe minimale correspondante séparant s et t est recherchée. Cette coupe va définir un ensemble $\{s\} \cup T$ dans lequel, de part les grandes capacités M définies, T va contenir j et l mais aucun dépôt.

   Soit T^* l'ensemble de clients induit par la coupe minimale. Deux cas sont possibles :

   - $x(\delta(T^*)) \geq A(j,l,I') + 1$. Alors, les contraintes (8.3) sont satisfaites pour tout S (vu que $x\left(\delta(S \cup \{j,l\})\right) \geq x(\delta(T^*))$ par définition de T^*).
   - $x(\delta(T^*)) < A(j,l,I') + 1$. Alors, deux alternatives peuvent se produire selon l'ensemble T^*.
     Si $T^* \neq \{j,l\}$, la contrainte (8.3) correspondant à $S = T^* \backslash \{j,l\}$ est violée.
     Si $T^* = \{j,l\}$, une étape supplémentaire est nécessaire pour conclure.

7. Si $T^* = \{j,l\}$, l'ensemble correspondant est $S = \emptyset$, mais dans ce cas, la contrainte correspondante à déjà été vérifiée. L'idée ici est de trouver un nouvel ensemble $S \neq \emptyset$. Pour cela, il faut forcer T^* à inclure au moins un client de $J \backslash \{j,l\}$.

   Une telle introduction d'un client $h \in J \backslash \{j,l\}$ dans T^* se fait en changeant la capacité de l'arête reliant s à h à la valeur M, et en réinitialisant l'algorithme de flot maximum. Cette démarche est à faire pour tout client $h \in J \backslash \{j,l\}$ appartenant à la même composante connexe que j et l dans le graphe-support. Ensuite, pour chaque ensemble $T^* \neq \{j,l\}$ ainsi généré, si $x(\delta(T^*)) < A(j,l,I') + 1$ alors une violation de la contrainte (8.3) est identifiée.

Une procédure de séparation est proposée pour les contraintes *path2-f1* :

$$2x\left(\gamma(S\cup\{j\})\right)+x_{ij}+\frac{1}{2}x\left(E(S\cup\{j\}:I\backslash\{i\})\right)\leq 2|\,S\,|+1 \qquad i\in I,\, S\subseteq J,\, j\in J\backslash S$$

La méthode suivante est appliquée à chaque client j contenu dans une composante connexe du graphe-support et relié directement à un dépôt. Parmi les dépôts directement connectés à j, on prend pour i celui relié à j par une arête de coût maximal. Ensuite, l'ensemble S est soit composé de tous les autres éléments de la composante connexe considérée, soit d'un sous-ensemble de cette composante. Dans ce dernier cas, S comprend tous les clients (sauf j) traversés par le flot maximum entre j et tout client de la composante connexe. Ainsi, la contrainte 8.5 peut être vérifiée.

Une autre procédure est nécessaire pour les contraintes *chain2-f1* de la seconde formulation :

$$\sum_{i\in I'}(w_{ij}+x_{ij})+x\left(\gamma(S\cup\{j,l\})\right)+\sum_{k\in I"}(w_{kl}+x_{kl})\leq|\,S\,|+2$$

$\{j,l\}\in J,\, S\subseteq J\backslash\{j,l\},\, S\neq\emptyset,\, I'\cap I"=\emptyset$

Elle est appliquée à chaque paire de dépôts i et k. La première étape consiste à vérifier si une chaîne reliant ces deux dépôts existe dans le graphe-support, en vérifiant par exemple si un flot non nul existe entre ces nœuds. Si c'est le cas, l'algorithme recherche le client j le plus fortement et directement connecté au dépôt i dans le graphe partiel des arêtes de flot non nul entre ces deux dépôts. S est alors l'ensemble de clients induit dans le flot entre les dépôts excluant j et l. l est le client le plus fortement et directement connecté au dépôt k dans le flot résultant.

I' est ensuite initialisé avec les dépôts incidents au client j uniquement et non à l. Si des dépôts sont incidents aux deux clients j et l, ou à aucun d'entre-eux, ils s'ajoutent à l'ensemble I' s'ils sont plus proches de j que de l. Les dépôts restants composent l'ensemble $I"$.

Ainsi, la contrainte (8.7) peut être vérifiée.

Enfin, il est possible d'identifier les contraintes *path2-f2* :

$$x\left(\gamma(S\cup j)\right)+x_{ij}+w_{ij}+\frac{1}{2}x\left(E(S\cup j:I\backslash\{i\})\right)+2w(E(S\cup j:I\backslash\{i\})\leq|\,S\,|+1$$

$i\in I,\, S\subseteq J,\, j\in J\backslash S$

Pour cela, la méthode consiste, pour chaque paire de dépôts i et k, à d'abord vérifier si une chaîne les reliant existe dans le graphe-support. Si c'est le cas, l'algorithme vérifie la contrainte pour chaque client j traversé par un flot non nul entre i et k adjacent à l'un de ces deux dépôts i. S est l'ensemble des clients traversés par ce flot, excepté j.

8.4.2.3 Séparation pour les contraintes de couverture

Dans les deux formulations, cela concerne les contraintes :

$$\sum_{i \in C} y_i \geq 1$$

avec C une couverture de la capacité non utilisée sur des dépôts (égale à $\sum_{i \in I} W_i - D(J)$).

La procédure utilisée consiste à trouver une couverture C de la demande, en sélectionnant itérativement les dépôts dont la valeur de la variable y_i correspondante est la plus grande jusqu'à ce que la somme de leur capacité couvre la demande totale des clients. Si $\sum_{i \in C} y_i < 1$, alors la contrainte est violée.

D'autres contraintes et procédures de séparations peuvent ajoutées à la méthode, comme celles implémentées en C++ par Lysgaard *et al.* (2004) pour le *VRP* (contraintes de capacité, d'hypotours, de peigne...). Pour le moment, nous n'avons inclus que des contraintes de capacité et de capacité généralisée.

8.5 Évaluation Numérique

8.5.1 Implémentation et instances

Les algorithmes sont ici codés en C++ et les programmes linéaires sont résolus en faisant appel à Cplex 8.0 par le biais de la DLL (*Dynamic Link Library* - bibliothèque du logiciel) du solveur. Cette technique, qui requiert des connaissances informatiques spécifiques, permet de récupérer directement les résultats du programme linéaire dans l'application C++.

De plus, une telle méthode de coupes n'est pas triviale à implémenter. Elle fait appel à de nombreuses procédures pour les séparations des contraintes. Ces dernières utilisent des algorithmes spécifiques, comme pour calculer des flots maximums ou les graphes réduits, dans lesquels des super-nœuds représentent des sous-ensembles de clients. Le codage implique donc des listes chaînées pour retrouver les contraintes violées à partir des procédures utilisant de tels graphes.

La méthode est exécutée sur un PC Dell Optiplex GX260, avec un Pentium 4 à 2.4 GHz, 512 Mo de RAM et Windows XP.

Plusieurs options de résolutions sont envisagées, comme décrit dans le paragraphe 8.4 : réaliser un algorithme de coupes en se basant sur un programme linéaire dont entre-autres les contraintes d'intégrité des variables sont toutes relaxées (version 1), ou un algorithme de coupes dans lequel l'intégrité des variables y, qui concernent le statut des dépôts, est conservée (version 2) de manière à obtenir un programme linéaire en nombres mixtes relaxé, ou encore un algorithme de branchement et coupes (version 3). Selon le choix réalisé, la qualité de la borne, mais aussi les temps de calcul peuvent fortement varier. C'est pourquoi, les différentes techniques ne sont pas appliquées à l'ensemble des instances présentées en Annexe, et en

particulier, à l'ensemble *I2* composé uniquement d'instances de grande taille. Pour chacune des versions, des précisions sont données dans la suite et la résolution est réalisée sur chacune des formulations avec variables à 2 indices proposées précédemment pour le *LRP* et utilisant les contraintes additionnelles présentées.

Les performances des différentes versions sont données ci-dessous. Dans les tableaux, les colonnes *Coût* donnent la valeur de la borne trouvée par une méthode, les colonnes *Tps* sont les temps de calculs, *V1-F1*, *V2-F1*, *V3-F1*, *V1-F2*, *V2-F2* et *V3-F2* correspondent respectivement aux versions 1, 2 et 3 de l'algorithme sur les formulations 1 et 2. Les colonnes n et m indiquent respectivement le nombre de clients et de dépôts de l'instance. β est le nombre de clusters, *type* est soit relatif à la capacité des tournées (instances *I1* et *I3*), soit correspond au ratio des clients appartenant à des clusters sur n (instances $I2$) (voir Annexe).

Enfin, les colonnes E_{Vi} donnent l'écart en pourcentage de la version considérée par rapport à la version i du paragraphe précédent sur le même type de formulation du problème, et la colonne *Ecart* donne l'écart relatif de la formulation 1 par rapport à la formulation 2. Les résultats présentés sont arrondis à la première décimale en fin d'algorithme afin d'être présentés dans les tableaux.

8.5.2 Paramétrage de l'algorithme

Comme précisé en introduction de ce chapitre, dans les approches par coupes, le problème initial est relaxé en éliminant de la formulation mathématique un ensemble de contraintes. Le but est alors d'obtenir un problème plus simple à résoudre itérativement en introduisant une à une les contraintes violées repérées à chaque résolution, afin de trouver des solutions en des temps de calcul bien inférieurs à ceux nécessaires pour aborder un programme linéaire en nombres entiers complet pour le *LRP*. Pour ne pas alourdir inutilement les formulations d'une itération à l'autre, le nombre de contraintes introduites par famille d'inégalités est limité à $MaxConst = 100$.

Le solveur utilisé pour résoudre les programmes linéaires est Cplex 8.0. Les paramètres utilisés sont ceux fournis par défaut dans ce solveur. Seule la version 3, de branchement et coupes, utilise un paramétrage spécifique. La sélection des variables séparées est basée sur une résolution partielle d'un nombre de sous-problèmes avec tentative de branchement pour sélectionner le plus prometteur (stratégie appelée *strong branching* par le solveur). La borne supérieure utilisée pour évaluer la pérennité des nœuds dans l'arborescence est fournie en initialisation. Elle correspond à la meilleure solution trouvée par les métaheuristiques décrites dans les chapitres précédents.

8.5.3 Résultats de la version 1

Dans la première version, l'algorithme de coupes pur est utilisé. Il s'agit de résoudre un programme linéaire relaxé à chaque itération, sans branchement sur les variables, et d'ajouter des coupes comme décrit en section 8.4. Les résultats obtenus sont présentés dans le tableau 8.1. Toutes les instances de l'ensemble *I3* sont ici résolues, plus celles de *I1* allant de $n = 20$ et $m = 5$ à $n = 100$ et $m = 5$ (description en Annexe).

				$V1-F1$		$V1-F2$		
n	m	β	type	Coût	Tps	Coût	Tps	Ecart
Ensemble *I1* :								
20	5	0	a	47736.9	1.8	48585.7	1.1	1.78
20	5	0	b	39104.0	0.4	39104.0	0.2	0.00
20	5	2	a	46103.3	2.4	46397.0	1.4	0.64
20	5	2	b	36921.2	0.9	37542.0	0.2	1.68
50	5	0	a	74875.6	75.5	75258.7	136.3	0.51
50	5	0	b	54766.7	18.2	55016.2	24.7	0.46
50	5	2	a	80119.7	32.7	80467.6	29.0	0.43
50	5	2	b	62673.6	9.8	62735.9	6.7	0.10
50	5	2'	a	80172.7	110.2	80897.5	155.1	0.90
50	5	2'	b	49189.4	19.1	50775.6	8.2	3.22
50	5	3	a	70329.4	45.9	71991.7	103.3	2.36
50	5	3	b	52952.6	14.4	53460.3	15.4	0.96
100	5	0	a	259703.4	4463.3	260186.3	5191.2	0.19
100	5	0	b	205619.9	452.3	205648.9	193.7	0.01
100	5	2	a	176176.8	587.7	177354.1	561.6	0.67
100	5	2	b	143308.0	156.5	143419.8	72.4	0.08
100	5	3	a	173896.7	217.8	175024.5	174.5	0.65
100	5	3	b	141184.8	91.8	141334.9	27.5	0.11
100	10	0	a	253980.8	1714.9	258242.6	1789.6	1.68
100	10	0	b	218580.1	132.9	218826.0	79.9	0.11
100	10	2	a	226289.7	1635.9	226905.0	1309.0	0.27
100	10	2	b	193726.6	98.3	194627.7	50.0	0.47
100	10	3	a	220595.8	737.4	222353.2	865.3	0.80
100	10	3	b	189191.4	160.1	189308.5	66.2	0.06
Moyenne				129050	449.2	129811	452.6	0.76
Ensemble *I3* :								
21	5		6000	396.2	3.0	403.7	0.9	1.90
22	5		4500	562.7	0.5	577.5	0.9	2.64
27	5		2500	2657.1	1.5	2746.3	1.4	3.36
29	5		4500	446.3	0.9	445.9	0.2	-0.10
32	5		8000	509.5	2.8	518.2	2.1	1.69
32	5		11000	475.3	2.9	477.8	0.7	0.54
36	5		250	430.0	5.6	433.2	5.7	0.73
50	5		160	479.2	9.6	487.9	4.9	1.82
75	10		140	718.4	83.8	718.7	65.9	0.05
88	8		9000000	319.8	15.2	320.6	19.2	0.27
100	10		200	762.3	90.4	765.3	40.8	0.39
134	8		850	5166.1	438.5	5189.4	70.1	0.45
150	10		8000000	39318.7	281.0	39470.5	61.0	0.39
Moyenne				4018.6	72.0	4042.7	21.1	1.09
Moyenne				85120.0	316.6	85622.1	301.0	0.87

NB : un écart négatif indique une meilleure borne par la formulation 1

TAB. 8.1 – *Résultats de l'algorithme de coupes* - V1

Il en ressort que la formulation 2 permet d'obtenir de meilleures bornes inférieures sur l'ensemble des instances sauf une, avec un gain moyen de 0.9%, pouvant même dépasser 3%. L'instance sur laquelle la formulation 1 fournit un meilleur résultat est obtenue en un temps de calcul quadruplé par rapport à la première formulation, pour un gain de 0.1% seulement.

Plus généralement, les temps d'exécution peuvent fortement varier d'une instance à l'autre, même si sa taille joue un rôle. Les problèmes les plus petits sont résolus en quelques secondes contre quelques minutes voire plus d'une heure pour les plus gros. Mais, dans l'ensemble *I1* par exemple, il est possible d'avoir des résultats sur des instances avec $n = 100$ et $m = 5$ entre 27.5 s et 5191.2 s.

La formulation permettant d'obtenir les résultats le plus rapidement n'est pas toujours la même. En moyenne, sur l'ensemble *I1*, la formulation 1 a un léger avantage. Par contre, sur l'ensemble *I3*, celle n'utilisant que des variables binaires apporte des bornes en des temps moins importants, particulièrement sur les deux dernières instances.

8.5.4 Résultats de la version 2

Cette fois-ci, l'idée est de procéder au même type de résolution que dans la version 1, c'est-à-dire à l'aide d'un algorithme de coupes, sauf que les variables de type y sont gardées entières dans la formulation initiale. C'est donc un programme linéaire en nombres mixtes qui est résolu à chaque itération de la méthode. L'avantage est de pouvoir améliorer la borne inférieure proposée tout en restant dans des temps de calcul raisonnables. En effet, les variables y sont relativement peu nombreuses dans les problèmes (entre 5 et 10 sur les instances traitées). Malgré tout, les instances de l'ensemble *I1* avec $n = 100$ et $m = 10$ sont ici exclues, ainsi que la plus grosse de *I3* ($n = 150$ et $m = 10$). Les résultats obtenus sont présentés dans le tableau 8.2.

Cette fois-ci, les coûts à partir de l'une ou l'autre des formulations mathématiques sont en moyenne équivalents. Ce n'est donc pas sur ce critère qu'il est possible de bien juger leur intérêt. Mais en regardant les temps de calcul, il apparaît clairement que la seconde formulation permet une résolution plus rapide.

En ce qui concerne l'amélioration de la borne inférieure proposée dans la version 1, elle est de plus de 7% en moyenne. Comme il était prévisible, il est très avantageux d'utiliser un branchement sur certaines variables pour obtenir de meilleurs résultats. Le choix porté sur les variables y est judicieux car ces variables sont peu nombreuses et jouent un grand rôle dans la solution comme le montrent les résultats. Par contre, bien que le nombre de dépôts contenus dans les instances soit relativement petit, n'impliquant ainsi que très peu de branchement dans la procédure de séparation et évaluation du programme linéaire en nombre mixte résultant, les temps de calcul par rapport à la version 1 sont bien plus importants. En effet, une procédure de séparation et évaluation, faisant appel à un solveur commercial, est utilisée à chaque itération pour résoudre le programme linéaire en nombres mixtes. Ainsi, en regardant la valeur moyenne dans les colonnes *Tps* sur l'ensemble des instances, par rapport au tableau 8.1, celle-ci a doublé alors que les plus grosses instances ne sont plus incluses. En fait, en ne regardant que les instances communes, les temps de calcul ont presque triplé dans la version 2.

				$V2-F1$			$V2-F2$			
n	m	β	type	Coût	Tps	E_{V1}	Coût	Tps	E_{V1}	Ecart
Ensemble *I1* :										
20	5	0	a	52419.7	6.6	9.81	52419.7	1.1	7.89	0.00
20	5	0	b	39104.0	0.2	0.00	39104.0	0.2	0.00	0.00
20	5	2	a	47458.8	4.3	2.94	47726.4	1.4	2.87	0.56
20	5	2	b	37440.5	0.7	1.41	37542.0	0.2	0.00	0.27
50	5	0	a	84750.6	449.4	13.19	84746.9	136.3	12.61	0.00
50	5	0	b	59574.9	53.5	8.78	59567.4	24.7	8.27	-0.01
50	5	2	a	82057.1	46.7	2.42	82051.0	29.0	1.97	-0.01
50	5	2	b	63836.0	23.1	1.85	63841.3	6.7	1.76	0.01
50	5	2'	a	82345.8	160.5	2.71	82356.6	155.1	1.80	0.01
50	5	2'	b	50360.0	25.4	2.38	51085.3	8.2	0.61	1.44
50	5	3	a	82703.8	365.8	17.59	82686.3	103.3	14.86	-0.02
50	5	3	b	59334.8	35.8	12.05	59473.8	15.4	11.25	0.23
100	5	0	a	272082.4	10534.3	4.77	263355.2	13107.0	1.22	-3.21
100	5	0	b	207037.4	933.8	0.69	207011.6	193.7	0.66	-0.01
100	5	2	a	186916.6	2578.6	6.10	186912.4	561.6	5.39	0.00
100	5	2	b	153827.1	471.0	7.34	153820.7	72.4	7.25	0.00
100	5	3	a	194196.8	1257.5	11.67	194202.0	174.5	10.96	0.00
100	5	3	b	149980.6	490.5	6.23	149985.6	27.5	6.12	0.00
Moyenne				105857	968.8	6.22	105438	983.9	5.30	-0.04
Ensemble *I3* :										
21	5		6000	417.3	3.5	5.33	417.4	5.1	3.38	0.01
22	5		4500	584.5	1.4	3.88	585.1	1.5	1.32	0.10
27	5		2500	3046.7	5.5	14.66	3046.7	3.4	10.94	0.00
29	5		4500	500.2	1.7	12.09	500.5	2.5	12.26	0.05
32	5		8000	555.3	6.9	8.97	555.3	9.7	7.16	0.00
32	5		11000	504.3	4.4	6.11	504.3	1.7	5.54	0.00
36	5		250	460.4	20.3	7.06	460.4	28.9	6.28	0.00
50	5		160	551.1	143.0	15.00	551.0	141.0	12.92	-0.03
75	10		140	791.4	1325.1	10.17	791.3	1018.0	10.10	-0.02
88	8		9000000	346.9	122.6	8.48	347.0	122.1	8.23	0.04
100	10		200	818.1	3274.2	7.32	818.1	2570.8	6.90	-0.01
134	8		850	5423.0	1892.4	4.97	5419.5	521.4	4.43	-0.07
Moyenne				1166.6	566.8	8.67	1166.4	368.8	7.45	0.01
Moyenne				63980.9	808.0	7.20	63729.5	737.8	6.16	-0.02

NB : $Ecart < 0 \Rightarrow$ meilleure borne par la formulation 1; $E_{V1} < 0 \Rightarrow$ meilleure borne par la version 1

TAB. 8.2 – *Résultats de l'algorithme de coupes avec les variables y entières* - V2

				$V3-F1$				$V3-F2$			
n	m	β	type	Coût	Tps	Noeuds	E_{V2}	Coût	Tps	Noeuds	E_{V2}
Ensemble *I1* :											
20	5	0	a	54793	1968.4	2395	4.53	54793	2014.5	1825.0	4.53
20	5	0	b	39104	0.2	0	0.00	39104	0.2	0.0	0.00
20	5	2	a	48908	102.5	198	3.05	48908	59.0	153.0	2.48
20	5	2	b	37542	0.9	1	0.27	37542	0.2	0.0	0.00
Moyenne				45086.8	518.01	648.50	1.96	45086.8	518.5	494.5	1.75
Ensemble *I3* :											
21	5		6000	424.9	7.4	33	1.81	424.9	8.6	27	1.80
22	5		4500	585.1	1.2	4	0.10	585.1	1.5	3	0.00
27	5		2500	3062.0	4.1	14	0.50	3062.0	3.9	14	0.50
29	5		4500	512.1	28.4	11	2.37	512.1	31.0	260	2.32
32	5		8000	562.2	14.0	396	1.26	562.2	26.2	36	1.26
32	5		11000	504.3	3.9	23	0.01	504.3	1.5	5	0.01
36	5		250	460.4	20.0	5	0.00	460.4	25.1	14	0.00
Moyenne				873.0	11.3	69.4	0.87	873.0	14.0	51.3	0.84
Moyenne				16950.7	195.5	280.0	1.26	16950.7	197.4	212.5	1.17

NB : $E_{V2} < 0 \Rightarrow$ meilleure borne par la version 2

TAB. 8.3 – *Résultats de l'algorithme de branchement et coupes* - V3

8.5.5 Résultats de la version 3

La dernière version proposée est une méthode de branchement et coupes. Cette approche exacte permet d'obtenir des solutions optimales pour les instances résolues. Par contre, même si les temps de calcul sont certainement moins importants que dans une résolution par séparation et évaluation, ils deviennent vite excessifs. Pour ces raisons, seules les instances de petites tailles sont abordés par cette méthode ($n < 50$). Le tableau 8.3 donne les résultats obtenus. Les valeurs étant optimales, la colonne *Ecart* est ici sans intérêt. Par contre, des colonnes *Nœuds* sont ajoutées. Elles représentent pour chaque formulation, le nombre de nœuds visités

sont abordées par cette dernière approche. Il est donc intéressant de confronter les résultats avec d'autres références.

De plus, jusqu'à maintenant, les résultats heuristiques obtenus n'ont été comparés qu'avec ceux fournis par des méthodes présentées plus tôt dans les chapitres. Il est cependant indispensable de mesurer la qualité des solutions avec des bornes et autres résultats de la littérature. C'est pourquoi, comme annoncé dans l'introduction, nous profitons de ce chapitre pour évaluer la qualité des résultats obtenus par les diverses approches développées au cours de cette thèse.

Pour chaque ensemble d'instances, les colonnes E_{LB} et E_{MRC} correspondent respectivement aux écarts relatifs des résultats concernés avec les meilleures bornes inférieures proposées LB et les meilleurs résultats connus MRC (borne supérieure). Ces derniers proviennent des coûts les plus faibles atteints lors du développement des différentes techniques de résolution heuristique durant cette thèse, avec l'usage de divers paramétrages.

8.6.1 Analyse sur les instances de l'ensemble *I1*

Le premier ensemble d'instances a été développé pour les besoins de cette thèse afin de traiter des cas avec capacité sur les dépôts. Aucune autre méthode à part celles présentées dans les chapitres précédents n'a été publiée sur de telles instances à ce jour.

L'idée est donc de comparer les résultats heuristiques avec les bornes connues (voir tableau 8.4). Les meilleures bornes inférieures, dénotées ici LB, donnent une première vision de la qualité des solutions proposées. Nos algorithmes atteignent l'optimalité sur les plus petites instances, sauf 20-5-1-a pour laquelle seul le $MA|PM$ trouve cette solution.

Cependant, sur les instances à plus de 20 clients, aucune valeur optimale n'est connue et aucune information sur la pertinence des bornes n'est disponible, sans compter que pour les plus grosses instances, aucune comparaison à LB n'est possible. Il est donc plus difficile de conclure sur la qualité des résultats. La borne inférieure peut être éloignée de la borne supérieure de plus de 20%. Les solutions obtenues par les approches heuristiques sont par contre toutes en moyenne relativement proches de MRC (3.85% pour le $GRASP$, 1.31% pour le $MA|PM$, et 0.55% pour le $LRGTS$) alors qu'elles tournent à plus de 6% de déviation moyenne par rapport à LB.

Il n'y a ici pas de surprise à voir que le $LRGTS$ est le plus proche des bornes proposées. Une nouvelle fois, il ressort que le $MA|PM$ obtient des résultats de très bonne qualité mais peut ponctuellement se révéler plus faible, comme sur les instances 100-10-1-a et 100-10-1-b. Par contre, le $LRGTS$ confirme son côté stable avec une déviation à MRC toujours relativement faible (en moyenne à 0.5%), et ne dépassant pas les 2.10%.

8.6.2 Analyse sur les instances de l'ensemble *I2*

Pour le deuxième ensemble d'instances (*I2*), composé uniquement de problèmes de grande taille, aucune borne inférieure n'est disponible. Il est cependant possible de mesurer les performances des heuristiques par rapport à la borne supérieure MRC. De plus, cet ensemble

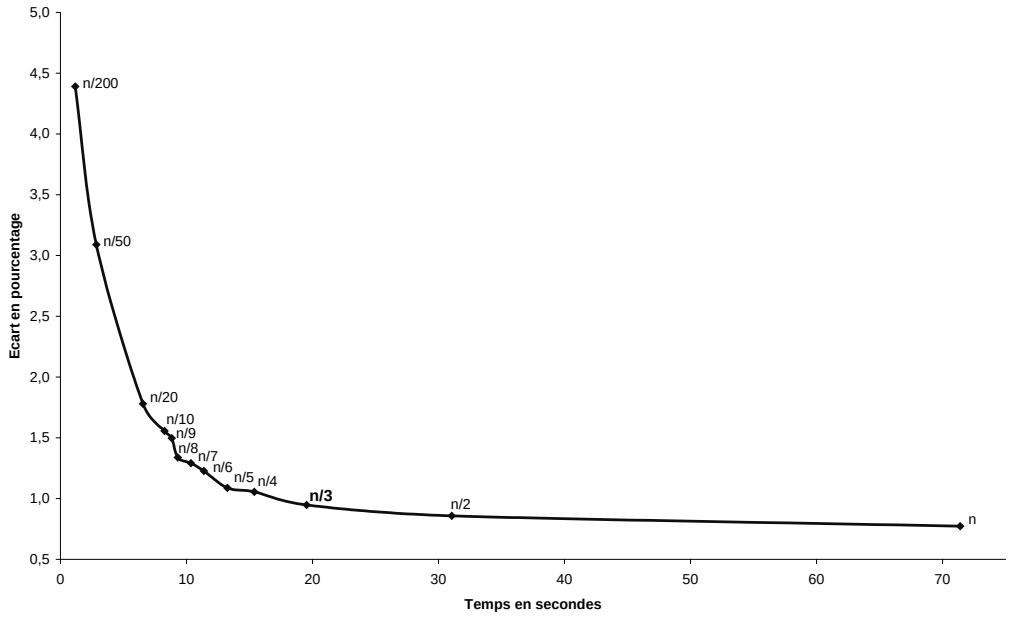


FIG. 8.3 – *Impact du nombre de perturbations autorisé - (MaxPertub)*

vient de la littérature et une comparaison de nos résultats avec ceux proposés par Tuzun et Burke (1999) est réalisée.

Leur méthode est une métaheuristique basée sur un principe tabou et est appelée *TS* dans le tableau 8.5. Il s'agit d'une approche dans laquelle, à chaque itération, la cardinalité de l'ensemble des dépôts ouverts est fixée. Elle alterne alors entre une phase de localisation de ces dépôts et une phase de routage. La cardinalité est d'abord égale à 1 puis augmente progressivement jusqu'à ce que cela ne permette plus de trouver de solution de meilleur coût. Ce principe n'est possible que parce que cet ensemble d'instances est sans capacité sur les dépôts. Une telle approche permet une résolution en des temps de calcul relativement faibles. En effet, en utilisant un ratio de conversion des temps selon les ordinateurs utilisés (Dongarra, 1985, revisé en 2006), il se révèle que la méthode de Tuzun et Burke (1999) obtient des solutions en 1 s environ en moyenne quand les métaheuristiques proposées dans cette thèse nécessitent entre 16 et 160 s.

Par contre, l'écart à *MRC* est bien plus grand qu'avec les algorithmes développés ici (3.7% contre 2.9% pour le GRASP et 1.3% pour le $MA|PM$ et le *LRGTS*). La figure 8.3 montre de plus que *LRGTS* est capable de fournir des résultats de même qualité en des temps de calculs similaires. *LRGTS* est la méthode coopérative du chapitre 7, échangeant un certain nombre d'informations lors d'alternance entre des phases de localisation des dépôts, abordée avec une relaxation lagrangienne, et de routage, traitée à l'aide d'une recherche de type tabou granulaire.

Cette figure illustre l'impact du paramètre *MaxPertub*, représentant le nombre de fois où la reconstruction coopérative de tournées est effectuée au maximum. Un point du graphe

synthétise les performances moyenne de $LRGTS$ sur les instances de type *I1* et *I2* pour une valeur donnée de *MaxPertub*. Le temps de calcul moyen nécessaire pour résoudre les problèmes est reporté en abscisse. L'écart moyen obtenu entre le coût des solutions obtenu par le $LRGTS$ et MRC est donné en ordonnée. Les valeurs de *MaxPertub* sont prises entre $n/200$ et n. Outre le fait que cette figure mette en évidence le besoin de diversification coopérative pour obtenir des solutions de bonne qualité (en référence avec MRC), selon les exigences retenues, le $LRGTS$ peut fournir des solutions de diverses valeurs au prix des temps de calcul. Mais la courbe montre aussi qu'il n'est pas forcément nécessaire d'augmenter excessivement le nombre de diversifications. Un bon compromis se situe autour de *MaxPertub*, paramètre servant de critère d'arrêt de l'algorithme, pris entre $n/10$ et $n/2$, d'où le choix fixé à $n/3$ dans les résultats proposés (voir chapitre 7). Cependant, il est possible de réduire les temps de calcul à quelques secondes, pour un écart à MRC de 3% environ.

Encore une fois, au regard simplement du coût des solutions, le $MA|PM$ et le $LRGTS$ sont de qualité semblable, mais $LRGTS$ est plus stable. Quoiqu'il en soit, l'écart à MRC est ici de 1.3% en moyenne, et toujours inférieur à 2% sauf pour 5 instances sur les 36 proposées et inférieur à 1% sur près de 40% des cas.

				LB	*MRC*		*GRASP*			*MAPM*			*LRGTS*		
n	m	β	type	Coût	Coût	E_{LB}	Coût	E_{LB}	E_{MRC}	Coût	E_{LB}	E_{MRC}	Coût	E_{LB}	E_{MRC}
20	5	0	a	54793.00	54793	0.00	55021	0.42	0.42	54793	0.00	0.00	55131	0.62	0.62
20	5	0	b	39104.00	39104	0.00	39104	0.00	0.00	39104	0.00	0.00	39104	0.00	0.00
20	5	2	a	48908.00	48908	0.00	48908	0.00	0.00	48908	0.00	0.00	48908	0.00	0.00
20	5	2	b	37542.00	37542	0.00	37542	0.00	0.00	37542	0.00	0.00	37542	0.00	0.00
50	5	0	a	84750.65	90111	6.32	90632	6.94	0.58	90160	6.38	0.05	90160	6.38	0.05
50	5	0	b	59574.89	63242	6.16	64761	8.71	2.40	63242	6.16	0.00	63256	6.18	0.02
50	5	2	a	82057.13	88298	7.61	88786	8.20	0.55	88298	7.61	0.00	88715	8.11	0.47
50	5	2	b	63841.35	67340	5.48	68042	6.58	1.04	67893	6.35	0.82	67698	6.04	0.53
50	5	2’	a	82356.61	84055	2.06	84055	2.06	0.00	84055	2.06	0.00	84181	2.22	0.15
50	5	2’	b	51085.29	51822	1.44	52059	1.91	0.46	51822	1.44	0.00	51992	1.77	0.33
50	5	3	a	82703.76	86203	4.23	87380	5.65	1.37	86203	4.23	0.00	86203	4.23	0.00
50	5	3	b	59473.83	61830	3.96	61890	4.06	0.10	61830	3.96	0.00	61830	3.96	0.00
100	5	0	a	272082.37	275993	1.44	279437	2.70	1.25	281944	3.62	2.16	277935	2.15	0.70
100	5	0	b	207037.38	214392	3.55	216159	4.41	0.82	216656	4.65	1.06	214885	3.79	0.23
100	5	2	a	186916.59	194598	4.11	199520	6.74	2.53	195568	4.63	0.50	196545	5.15	1.00
100	5	2	b	153827.05	157173	2.18	159550	3.72	1.51	157325	2.27	0.10	157792	2.58	0.39
100	5	3	a	194202.03	200246	3.11	203999	5.04	1.87	201749	3.89	0.75	201952	3.99	0.85
100	5	3	b	149985.58	152586	1.73	154596	3.07	1.32	153322	2.22	0.48	154709	3.15	1.39
100	10	0	a	258242.64	290429	12.46	323171	25.14	11.27	316575	22.59	9.00	291887	13.03	0.50
100	10	0	b	218825.96	234641	7.23	271477	24.06	15.70	270251	23.50	15.18	235532	7.63	0.38
100	10	2	a	226904.99	244265	7.65	254087	11.98	4.02	245123	8.03	0.35	246708	8.73	1.00
100	10	2	b	194627.72	203988	4.81	206555	6.13	1.26	205052	5.36	0.52	204435	5.04	0.22
100	10	3	a	222353.23	253344	13.94	270826	21.80	6.90	253669	14.08	0.13	258656	16.33	2.10
100	10	3	b	189308.50	204597	8.08	216173	14.19	5.66	204815	8.19	0.11	205883	8.76	0.63
200	10	0	a	–	479425	–	490820	–	2.38	483497	–	0.85	481676	–	0.47
200	10	0	b	–	378773	–	416753	–	10.03	380044	–	0.34	380613	–	0.49
200	10	2	a	–	450468	–	512679	–	13.81	451840	–	0.30	453353	–	0.64
200	10	2	b	–	374435	–	379980	–	1.48	375019	–	0.16	377351	–	0.78
200	10	3	a	–	472898	–	496694	–	5.03	478132	–	1.11	476684	–	0.80
200	10	3	b	–	364178	–	389016	–	6.82	364834	–	0.18	365250	–	0.29
	Moyenne				197323	4.48	207322	7.2	3.4	200309	5.9	1.1	198552	5.0	0.5

TAB. 8.4 – *Comparaison des résultats sur les instances de type I1 avec capacité sur les dépôts*

				MRC	*TS*		*GRASP*		*MAPM*		*LRGTS*	
n	m	β	type	Coût	Coût	E_{MRC}	Coût	E_{MRC}	Coût	E_{MRC}	Coût	E_{MRC}
100	10	0	0.75	1468.40	1556.64	6.01	1525.25	3.87	1493.92	1.74	1490.82	1.53
100	20	0	0.75	1449.20	1531.88	5.71	1526.90	5.36	1471.36	1.53	1471.76	1.56
100	10	0	1	1396.46	1443.43	3.36	1423.54	1.94	1418.83	1.60	1412.04	1.12
100	20	0	1	1432.29	1511.39	5.52	1482.29	3.49	1492.46	4.20	1443.06	0.75
100	10	3	0.75	1167.53	1231.11	5.45	1200.24	2.80	1173.22	0.49	1187.63	1.72
100	20	3	0.75	1102.70	1132.02	2.66	1123.64	1.90	1115.37	1.15	1115.95	1.20
100	10	3	1	793.97	825.12	3.92	814.00	2.52	793.97	0.00	813.28	2.43
100	20	3	1	728.30	740.54	1.68	747.84	2.68	730.51	0.30	742.96	2.01
100	10	5	0.75	1238.49	1316.98	6.34	1273.10	2.79	1262.32	1.92	1267.93	2.38
100	20	5	0.75	1246.34	1274.50	2.26	1272.94	2.13	1251.32	0.40	1256.12	0.78
100	10	5	1	902.38	920.75	2.04	912.19	1.09	903.82	0.16	913.06	1.18
100	20	5	1	1021.31	1042.21	2.05	1022.51	0.12	1022.93	0.16	1025.51	0.41
150	10	0	0.75	1866.75	2000.97	7.19	2006.70	7.50	1959.39	4.96	1946.01	4.25
150	20	0	0.75	1841.86	1892.84	2.77	1888.90	2.55	1881.67	2.16	1875.79	1.84
150	10	0	1	1981.37	2022.11	2.06	2033.93	2.65	1984.25	0.15	2010.53	1.47
150	20	0	1	1809.25	1854.97	2.53	1856.07	2.59	1855.25	2.54	1819.89	0.59
150	10	3	0.75	1448.27	1555.82	7.43	1508.33	4.15	1448.27	0.00	1448.65	0.03
150	20	3	0.75	1444.25	1478.80	2.39	1456.82	0.87	1459.83	1.08	1492.86	3.37
150	10	3	1	1206.73	1231.34	2.04	1240.40	2.79	1207.41	0.06	1211.07	0.36
150	20	3	1	931.94	948.28	1.75	940.80	0.95	934.79	0.31	936.93	0.54
150	10	5	0.75	1699.92	1762.45	3.68	1736.90	2.18	1720.30	1.20	1729.31	1.73
150	20	5	0.75	1401.82	1488.34	6.17	1425.74	1.71	1429.34	1.96	1424.59	1.62
150	10	5	1	1199.51	1264.63	5.43	1223.70	2.02	1203.44	0.33	1216.32	1.40
150	20	5	1	1152.86	1182.28	2.55	1231.33	6.81	1158.54	0.49	1162.16	0.81
200	10	0	0.75	2281.78	2379.47	4.28	2384.01	4.48	2293.99	0.54	2296.52	0.65
200	20	0	0.75	2185.55	2211.74	1.20	2288.09	4.69	2277.39	4.20	2207.50	1.00
200	10	0	1	2234.78	2288.17	2.39	2273.19	1.72	2274.57	1.78	2260.87	1.17
200	20	0	1	2259.52	2355.81	3.79	2345.10	3.79	2376.25	5.17	2259.52	0.00
200	10	3	0.75	2101.90	2158.60	2.70	2137.08	1.67	2106.26	0.21	2120.76	0.90
200	20	3	0.75	1709.56	1787.02	4.53	1807.29	5.72	1771.53	3.62	1737.81	1.65
200	10	3	1	1467.54	1549.79	5.60	1496.75	1.99	1467.54	0.00	1488.55	1.43
200	20	3	1	1084.78	1112.96	2.60	1095.92	1.03	1088.00	0.30	1090.59	0.54
200	10	5	0.75	1973.28	2056.11	4.20	2044.66	3.62	1973.28	0.00	1984.06	0.55
200	20	5	0.75	1957.23	2002.42	2.31	2090.95	6.83	1979.05	1.11	1986.49	1.50
200	10	5	1	1771.06	1877.30	6.00	1788.70	1.00	1782.23	0.63	1786.79	0.89
200	20	5	1	1393.62	1414.83	1.52	1408.63	1.08	1396.24	0.19	1401.16	0.54
	Moyenne			1510.14	1566.77	3.7	1556.51	2.9	1532.19	1.3	1528.75	1.3

TAB. 8.5 – *Comparaison des résultats sur les instances de type I2 sans capacité sur les dépôts*

			LB	*LBB*		*HB*		*GRASP*		*MAPM*		*LRGTS*	
n	m	type	Coût	Coût	E_{LB}	Coût	E_{LB}	Coût	E_{LB}	Coût	E_{LB}	Coût	E_{LB}
21	5	6000	424.9	424.9	0.0	435.9	2.6	429.6	1.1	424.9	0.0	424.9	0.0
22	5	4500	585.1	585.1	0.0	591.5	1.1	585.1	0.0	611.8	4.6	587.4	0.4
27	5	2500	3062.0	3062	0.0	3062.0	0.0	3062.0	0.0	3062.0	0.0	3065.2	0.1
29	5	4500	512.1	512.1	0.0	512.1	0.0	515.1	0.6	512.1	0.0	512.1	0.0
32	5	8000	562.2	556.5	-1.0	571.7	1.7	571.9	1.7	571.9	1.7	584.6	4.0
32	5	11000	504.3	504.3	0.0	511.4	1.4	504.3	0.0	534.7	6.0	504.8	0.1
36	5	250	460.4	460.4	0.0	470.7	2.2	460.4	0.0	485.4	5.4	476.5	3.5
50	5	160	551.1	549.4	-0.3	582.7	5.7	599.1	8.7	565.6	2.6	586.4	6.4
75	10	140	791.4	744.7	-5.9	886.3	12.0	861.6	8.9	866.1	9.4	863.5	9.1
88	8	9000000	818.1	356.4	-3.6	384.9	8.7	356.9	5.3	355.8	3.9	368.7	3.0
100	10	200	347.0	788.6	2.7	889.4	10.9	861.6	2.8	850.1	2.5	842.9	6.2
134	8	850	5423.0	–		6238.0	15.0	5965.1	10.0	5950.1	9.7	5809.0	7.1
150	10	8000000	39470.5	43406	10.0	46642.7	18.2	44625.2	13.1	44011.7	11.5	44386.3	12.5
Moyenne			4116.3	4329.2	0.2	4752.3	6.1	4569.1	4.0	4523.3	4.4	4539.4	4.0

TAB. 8.6 – *Comparaison des résultats sur les instances de type I3 avec capacité sur les dépôts*

8.6.3 Analyse sur les instances de l'ensemble *I3*

L'ensemble *I3* est utilisé par Barreto (2004b) dans sa thèse pour tester une heuristique (dénotées *HB* dans la suite) et une proposition de borne inférieure (*LBB*). *HB* est en fait une méthode composite basée sur un algorithme de clusters (regroupement des clients) en quatre phases. Elle est exécutée 24 fois, en utilisant 4 techniques pour les clusters et 6 mesures de proximité. Les quatre phases de la méthode sont les suivantes : d'abord la construction de groupes de clients avec une limite de capacité égale à Q; ensuite résolution d'un *TSP* à l'intérieur de chaque groupe; amélioration des tournées; et enfin, une phase de localisation de dépôts et d'affectation des tournées.

LBB est une borne inférieure obtenue avec CPLEX 7.1 sur un algorithme de coupes basé sur un programme linéaire en nombres entiers avec des variables à 2 indices mais aucun détail sur l'implémentation n'est disponible. Par rapport au problème étudié dans cette thèse, une contrainte supplémentaire est ajoutée. Elle concerne le nombre maximum de véhicules utilisé par dépôt, égal à W_i/Q. Théoriquement, la comparaison des bornes obtenues par Barreto et les nôtres n'est pas valide. Cependant, notre formulation ignorant cette contrainte, notre valeur optimale sera toujours égale ou inférieure à celle de la formulation utilisée par Barreto. Ainsi, une borne améliorée par notre résolution est sans conteste valide pour les deux modèles. Seules nos bornes de coût inférieur doivent être jugées avec précaution. De plus, comme le montre les résultats sur les instances résolues optimalement par les deux approches, en pratique, les coûts résultants sont souvent identiques. Néanmoins, il est possible que ponctuellement, la comparaison soit inopportune.

Cette borne *LBB* étant la seule de la littérature sur laquelle nous pouvons nous comparer, il est donc tout de même intéressant de l'utiliser comme point de comparaison au pire cas (voir tableau 8.6). Il apparaît que nos résultats améliorent les valeurs connues sur 1/3 des instances de *I3*, et fournissent une borne inférieure sur une instance n'en disposant pas pour le moment. Les performances de nos algorithmes de coupes sont donc très encourageants. Le tableau 8.6 montre également que les métaheuristiques proposées dans cette thèse obtiennent de bien meilleurs résultats que les heuristiques *HB*. Nos méthodes sont capables d'atteindre l'optimalité sur quelques petites instances et sont en moyenne à environ 4% des bornes inférieures.

8.7 Conclusion

Dans ce chapitre, une méthode de résolution exacte et des bornes inférieures sont proposées pour le problème de localisation-routage. Le principe est basé sur un algorithme de coupes. Une formulation judicieuse du problème est alors nécessaire. Deux sont proposées, il s'agit des formulations à deux indices du chapitre 3. Elles ont été choisies car le nombre de variables impliquées est plus faible que dans le modèle à trois indices. De plus, elles se rapprochent de celle utilisée dans la plupart des approches polyédrales pour le *VRP*.

Au départ, certaines contraintes du modèle sont relaxées, dont les contraintes d'intégrité des variables, et la solution du programme linéaire obtenu est calculée. Des procédures de séparation identifient alors des inégalités violées dans la formulation initiale. Ces dernières sont ajoutées au problème relaxé qui est à nouveau résolu. L'avantage est la simplification

apportée, permettant d'obtenir des solutions en des temps de calcul raisonnables pour des instances de petites et moyenne taille. De nouvelles familles de contraintes valides ont été proposées, ainsi que des procédures permettant de les identifier. Ces dernières nécessitent le développement d'algorithmes exacts et la méthode requiert l'appel à un solveur à chaque itération.

Pour obtenir des bornes plus pertinentes, une version de l'algorithme de coupes ne relaxant pas les contraintes d'intégrité sur les variables y, concernant le statut des dépôts, est utilisée. Cette dernière permet d'améliorer fortement les résultats tout en gardant des temps de calcul raisonnables pour les instances allant jusqu'à $n = 134$ et $m = 8$. Malheureusement, les solutions obtenues ne fournissent pas en général des résultats optimaux. L'idée est donc d'utiliser un algorithme de coupes au sein d'une procédure de séparation et évaluation (branchement et coupes). Le branchement devant se faire sur un grand ensemble de variables, seules les plus petites instances peuvent alors être abordées.

Ces algorithmes sont capables de résoudre de manière exacte des petites instances du *LRP* et fournissent une bonne borne inférieure pour les plus grandes instances. La qualité de ces bornes est mesurée avec les meilleurs résultats heuristiques connus ou avec d'autres bornes de la littérature. Il apparaît que, comparé à des bornes supérieures, l'écart peut être relativement important sur les instances de moyenne et grande taille. Cependant, les méthodes de coupes proposées, en particulier celle basée sur un programme linéaire en nombres mixtes, apportent des résultats intéressants et permettent d'améliorer des bornes de la littérature et d'en proposer de nouvelles.

Nous avons également profité de l'introduction de ces bornes pour valider la qualité des solutions obtenues par les méthodes approchées exposées dans les chapitres précédents. Il apparaît que les résultats sur les petites instances sont souvent optimaux. De manière plus générale, en moyenne, ils se situent à environs 10% de la borne inférieure (quand elle est disponible) mais les écarts se réduisent à 3% pour le *GRASP* et à près de 1% pour le *MA|PM* et le *LRGTS* par rapport aux meilleures solutions connues. Ces bons résultats améliorent de plus ceux de la littérature.

Ce travail sur les bornes inférieures a été mené en collaboration avec J.M. Belenguer et E. Benavent de l'Université de Valence (Espagne). Il a fait l'objet de deux présentations en congrès. La première est ODYSSEUS 2006 (Belenguer *et al.*, 2006a), troisième conférence internationale sur transport fret et la logistique, avec actes publiés. La seconde est EURO XXI, 21^{ime} conférence européenne en recherche opérationnelle (Belenguer *et al.*, 2006b).

Les bornes proposées peuvent encore être améliorées en travaillant à la recherche d'inégalités valides supplémentaires et de procédures de séparations efficaces. De plus, le paramétrage de l'algorithme de branchement et coupes peut être affiné pour tenter d'en augmenter les performances. D'autres approches exactes sont également possibles pour obtenir des bonnes bornes inférieures et solutions optimales au *LRP*. On peut citer par exemple l'utilisation de la quatrième formulation mathématique (PL4) du chapitre 3 afin d'appliquer l'approche de décomposition de Benders comme alors expliqué. Une autre possibilité, en se basant sur une formulation similaire basées sur un ensemble de tournées, est l'application de relaxations lagrangiennes afin d'employer une approche basée sur les idées développées par Baldacci et Mingozzi (2006) pour le *VRP*. Enfin, une autre méthode pour obtenir des bornes, dans le cas où le nombre k de dépôts à ouvrir est connu, consiste à appliquer une résolution d'arbre

recouvrant à k-racines.

Chapitre 9

Conclusion générale

L'intérêt pour les problèmes de logistique du transport est grandissant dans la vie actuelle. De nombreux domaines sont impliqués comme la distribution et la collecte de produits, ou l'agencement de réseaux de communication. Avec la mondialisation, l'efficacité des réseaux logistiques est devenue capitale pour assurer les liaisons entre les centres de production et les marchés. De plus, les exigences des clients et les problèmes environnementaux donnent une importance non négligeable aux coûts logistiques. Des efforts sont donc nécessaires afin de développer des réseaux de transport flexibles et efficaces.

L'approche classique, majoritairement adoptée dans le passé pour l'optimisation des systèmes logistiques, est la décomposition des problèmes selon les niveaux de décision, en commençant par l'échelon stratégique. L'avantage d'une telle démarche est la simplification apportée. Cependant, de cette conduite résulte bien souvent une perte de l'optimalité globale car l'intégralité du problème n'est pas prise en compte. Il est donc intéressant d'adopter une vision plus large considérant plusieurs niveaux de décision.

Le but de cette thèse a été de traiter le problème de localisation-routage (*Location-Routing Problem* - *LRP*), combinant des choix stratégiques avec une localisation de dépôts à définir et des choix tactiques relatifs aux tournées de véhicules à effectuer afin de visiter l'ensemble des clients, en respectant les capacités des dépôts ouverts et des véhicules. L'objectif est la minimisation des coûts résultants de l'ensemble des décisions prises.

L'étude a d'abord porté sur une synthèse de la littérature concernant les problèmes logistiques en général, et plus spécifiquement le *LRP*. Les différentes approches de résolution de ce type de problèmes combinatoires ont été passées en revue. Le principal constat qui en a découlé est que le problème abordé dans cette thèse reste encore relativement peu étudié, en particulier le cas généralisé combinant des capacités limitées à la fois sur les dépôts et les véhicules, ce qui a encouragé une recherche plus approfondie dans ce sens.

Les premiers algorithmes que nous avons développés sont basés sur des heuristiques constructives et de fusion. Ils permettent d'obtenir assez rapidement des solutions réalisables et nous ont donné une première appréciation des particularités du problème. Soulignons que ces heuristiques utilisent des recherches locales basées sur divers types de mouvements, dont certains relativement complexes. De plus, ces algorithmes sont ré-utilisés par la suite pour initialiser des méthodes plus agressives.

Des méthodes plus puissantes et plus performantes basées sur des métaheuristiques ont été proposées par la suite. Tout d'abord, un *GRASP* a été élaboré. Il s'agit d'une méthode itérative basée sur une heuristique gloutonne et randomisée. L'algorithme de Clarke et Wright a été adapté au cas multi-dépôt et utilisé comme heuristique gloutonne. Nous avons agrémenté la méthode classique du *GRASP* d'une mémorisation de dépôts de manière à acquérir des informations permettant d'alterner judicieusement entre des phases de diversification et d'intensification. Il est suivi ensuite par une post-optimisation réalisée par le biais d'un *path relinking* entre quelques solutions diversifiées et de bonne qualité sélectionnées durant les itérations du *GRASP*. Une mesure de dissimilarité pour le *LRP* a donc été proposée pour garantir la diversité de ces solutions. Les résultats des heuristiques ont été nettement améliorés par cette méthode.

La seconde métaheuristique que nous avons appliquée au *LRP* est un $MA|PM$, une forme très récente d'algorithme génétique intégrant une recherche locale et une gestion de la population grâce à une mesure de distance entre les solutions. L'avantage de cette approche est de travailler sur un ensemble de solutions suffisamment diversifiées et de bonne qualité. Ce processus permet une meilleure exploration de l'espace des solutions en combinant les deux aspects déterminants d'une recherche : la diversification grâce une mesure de dissimilarité et l'intensification en utilisant une recherche locale. Bien que le principe de cette métaheuristique semble relativement simple, sont implémentation a nécessité certains efforts. Il a fallu construire un bon codage des solutions. Celui proposé est particulièrement efficace car il permet de représenter chaque solution par deux sous-séquences de taille fixe caractérisant la localisation et le routage. Le $MA|PM$ est une méthode à population plus sophistiquée que le *GRASP*, et les résultats obtenus améliorent ceux fournis précédemment.

Les temps de calculs de ces métaheuristiques vont de quelques secondes à quelques minutes. Ceci est tout à fait raisonnable puisque des décisions stratégiques sont impliquées, mais une réflexion plus approfondie, basée sur la structure du problème, a semblé opportune.

Une méthode dite coopérative a fait suite aux premiers résultats métaheuristiques. Il s'agit de tirer parti des particularités du problème. Une décomposition hiérarchique en niveaux de décision n'est pas adéquate. Par contre, un échange coopératif entre la localisation et le routage apporte une synergie et permet une résolution de sous-problèmes de taille plus restreinte, conduisant à une réduction des temps de calcul. Ainsi, une phase de localisation est résolue en agrégeant les clients d'une même tournée en un super-client. Ceci aboutit à un problème de localisation, auquel est appliqué une relaxation lagrangienne des contraintes d'affectation. La sélection de dépôts ouverts permet de ne traiter ensuite qu'un *VRP* multi-dépôt pour la partie routage, prise en charge par un *GTS*. L'algorithme alterne entre ces deux phases. Une

résultats de l'ensemble de nos approches est mise en évidence par une comparaison avec les meilleures solutions connues sur les instances utilisées, ainsi qu'avec d'autres résultats de la littérature. Il en résulte particulièrement que nos approches améliorent les résultats publiés par d'autres auteurs.

De plus, soulignons que pour tester l'ensemble de nos méthodes et fournir les résultats proposés, un développement informatique pointu a été nécessaire comme l'analyse d'algorithmes et de leurs complexités ou l'utilisation de structures de données complexes. L'implémentation des approches exposées n'est pas triviale et requiert le développement de nombreuses procédures, parfois exactes. Celles-ci ont été réalisées en langage C et C++ (de l'ordre de 18000 lignes en tout) et des appels à un solveur commercial à partir du programme informatique en C++ ont parfois été nécessaire. Pour valider les méthodes, la conception de jeux d'essai, ainsi qu'un travail de débogage, réglage des paramètres et dépouillement des résultats ont également dû être réalisés.

L'ensemble de ces recherches ont déjà donné lieux à deux articles acceptés dans des revues internationales (*4OR - A Quarterly Journal of Operations Research* et *Lecture Notes in Computer Science*) et dans 8 conférences nationales et internationales. Notons de plus qu'un troisième article est en cours de révision pour la revue internationale *Transportation Science*.

Diverses conclusions résultent de l'ensemble de ces travaux. La première est que des études restent à faire concernant le *LRP*, en particulier afin d'apporter des bornes inférieures plus serrées au problème, voire des solutions optimales sur les nombreuses instances encore ouvertes. Pour cela, notre prochain objectif est d'améliorer encore les bornes proposées en recherchant des inégalités valides supplémentaires ou en affinant le paramétrage de l'algorithme de séparation et branchement. D'autres approches exactes sont également envisageables comme en se basant sur d'autres modélisations du problème et l'application de relaxations.

Il pourrait également être intéressant de développer de nouveaux jeux d'instances avec capacité sur les dépôts et les véhicules, et un rapport coût de localisation/coût de routage plus faible.

L'intégration de nouvelles contraintes à notre problème semble pertinente pour des applications pratiques. L'introduction de fenêtres de temps pour la visite des clients faciliterait la tâche de réception de marchandises pour ces derniers, surtout s'il y a des mesures spécifiques à prendre (réception de certains produits chimiques par exemple). La deuxième extension qui nous parait pertinente est la considération de l'aspect périodique des demandes des clients. Notons finalement que le *LRP* généralisé étant déjà *NP*-difficile, l'extension vers une version plus générale n'est pas triviale. Cependant, ce type d'études n'a encore jamais été proposé et de tels axes de recherche sont importants pour se rapprocher au mieux de problèmes réalistes.

Annexe

Les instances utilisées pour tester les méthodes développées durant cette thèse sont expliquées dans cette annexe. Elles sont de trois types, dénotées *I1*, *I2* et *I3*.

Ensemble d'instances *I1*

L'ensemble *I1* a été créé pour les besoins de cette thèse. En effet, dans la littérature, il n'existe que très peu d'instances pour le *LRP*, et la plupart sont conçues pour des cas plus spécifiques comme les configurations sans capacité limitée (Albareda-Sambola *et al.*, 2005; Tuzun et Burke, 1999) ou avec une flotte hétérogène (Wu *et al.*, 2002).

30 nouvelles instances sont donc proposées pour des cas de petite-moyenne et grande taille. Toutes les données, y compris les coûts ont des valeurs entières. Plus précisément, ci dessous sont donné les paramètres utilisés :

- nombre de dépôts : $m \in \{5,10\}$
- nombre de clients : $n \in \{20,50,100,200\}$
- capacité fixe des véhicules : $Q \in \{75,150\}$, le type a dans les tableau correspondant à $Q = 75$, le type b à $Q = 150$.
- nombre de clusters (regroupement de clients) : $\beta \in \{0,2,3\}$, 0 représentant une répartition uniforme dans l'espace Euclidien. Deux instances, marquées d'un apostrophe, ont des clusters plus fortement éloignés.

m est ici limité arbitrairement à 10. Le coût d'ouverture des dépôts varie d'un site à l'autre, par contre, le coût fixe d'utilisation d'un véhicule est identique pour tous. La capacité par dépôt peut varier mais est déterminée afin d'avoir en moyenne 2 ou 3 dépôts ouverts au minimum pour couvrir la demande des clients. Cette dernière est donnée aléatoirement à partir d'une distribution uniforme allant de 11 à 20.

Les coûts des arêtes c_{ij} correspondent aux distances Euclidiennes, arrondies à l'entier supérieur après les avoir multipliées par 100.

Ensemble d'instances *I2*

Afin de tester nos méthodes sur divers type d'instances, mais aussi de pouvoir comparer nos résultats avec ceux de la littérature, un autre ensemble, *I2*, est utilisé au cours de cette thèse. Il a été développé par Tuzun et Burke (1999) afin d'évaluer leur méthode de type tabou pour le LRP sans capacité sur les dépôts.

Il est composé de 36 instances avec :

- nombre de dépôts : $m \in \{10,20\}$
- nombre de clients : $n \in \{100,150,200\}$
- capacité fixe des véhicules : $Q = 150$
- nombre de clusters (regroupement de clients) : $\beta \in \{0,3,5\}$, 0 représentant une répartition uniforme dans l'espace Euclidien. La répartition dans l'espace est également controlée par un pourcentage (75% ou 100%) correspondant au nombre de clients appartenant aux clusters.

Le coût d'ouverture des dépôts est ici est identique pour tous, comme le coût fixe d'utilisation d'un véhicule. Les coûts des arêtes c_{ij} correspondent aux distances Euclidiennes, non arrondies.

Ensemble d'instances *I3*

Le dernier jeux d'instances vient de fichiers utilisés par Barreto (2004b) pendant sa thèse pour tester ses heuristiques et bornes inférieures au problème de localisation-routage. Cet ensemble, *I3*, est composé de 13 instances avec :

- nombre de dépôts : $m \in \{5,8,10\}$
- nombre de clients : $n \in [21,150]$

Le coût d'ouverture des dépôts est ici encore est identique pour tous (sauf sur l'instance avec $n = 88$ et $m = 8$). Le coût fixe d'utilisation d'un véhicule est nul. Les coûts des arêtes c_{ij} correspondent aux distances Euclidiennes, non arrondies.

Les instances sont issues de la littérature sur les problèmes de tournées de véhicules et adaptées au besoin. Ils sont composés de cas avec capacité sur les dépôts et les véhicules. Sur certaines instances, la demande des clients peut être très dispersée allant de 1 à 273 par exemple pour une capacité de véhicule de 850 (instance à 134 clients et 8 dépôts) ou au contraire égale pour chaque client (par exemple l'instance avec 36 clients et 5 dépôt). Les fichiers sont disponibles sur internet Barreto (2004a).

Bibliographie

Aardal, K., Pochet, Y., et Wolsey, L. (1995). Capacitated facilty location: valid inequalities and facets. *Mathematics of Operations Research*, *20*, 562–582.

Ahlander, F. (1994). *The capacitated facility location problem with single sourcing*. Ph.D. thesis, Linköping University, Sweden.

Albareda-Sambola, M. (2003). *Models and Algorithms for Location-Routing and Related Problems*. Ph.D. thesis, Universitat Politècnica de Catalunya.

Albareda-Sambola, M., Díaz, J., et Fernández, E. (2005). A compact model and tight bounds for a combined location-routing problem. *Computers and Operations Research*, *32*, 407–428.

Altinel, K., Aras, N., et Oommen, J. (2000). Fast, efficient and accurate solutions to the hamiltonian path problem using neural approaches. *Computers and Operations Research*, *27*(5), 461–494.

Alumur, S. et Kara, B. (.à paraître). A new model for the hazardous waste location-routing problem. *Computers and Operations Research*.

Applegate, D., Bixby, R., Chvatal, V., et Cook, W. (2005). http://www.tsp.gatech.edu/concorde.html.

Augerat, P., Belenguer, J.-M., Benavent, E., Corberan, A., et Naddef, D. (1999). Separating capacity constraints in the CVRP using tabu search. *European Journal of Operational Research*, *106*, 546–557.

Balas, E. (1980). Cutting planes from conditional bounds: A new approach to set covering. *MPS*, *12*.

Balas, E. et Carrera, M.-C. (1996). A dynamic subgradient-based branch-and-bound procedure for set covering. *Operations Research*, *44*, 875–890.

Balas, E. et Ho, A. (1980). Set covering using cutting planes, heuristics and subgradient optimisation: A computational study. *Mathematical Programming Study*, *12*, 37–60.

Balas, E. et Ng, S. (1989). On the set covering polytope: All facets with coefficients in 0,1,2. *Mathematical Programming*, *43*, 57–69.

Balas, E. et Toth, P. (1985). Branch and bounds methods. In E. Lawler, J. Lenstra, A. R. Kan, et D. Shmoys (Eds.), *The traveling Salesman Problem*. Chichester: John Wiley & Sons, 361–401.

Baldacci, R., Hadjiconstantinou, E., Maniezzo, V., et Mingozzi, A. (2002). A new method for solving capacitated location problems based on a set partitioning approach. *Computers and Operations Research*, *29*, 365–386.

Baldacci, R. et Mingozzi, A. (2006). Lower bounds and an exact method for the capacitated vehicle routing problem. In *Odysseus 2006 - Third International Workshop on Freight Transportation and Logistics*. Altea, Espagne, 42–44.

Balinski, M. et Quandt, R. (1964). On an integer program for a delivery program. *Operations Research*, *12*, 300–304.

Barcelo, J. et Casanovas, J. (1984). A heuristic lagrangean algorithm for the capacitated plant location problem. *European Journal of Operational Research*, *15*, 212–226.

Barreto, S. (2004a). http://sweet.ua.pt/ iscf143.

Barreto, S. (2004b). *Análise e Modelização de Problemas de localização-distribuição [Analysis and modelling of location-routing problems]*. Ph.D. thesis, University of Aveiro, campus universitário de Santiago, 3810-193 Aveiro, Portugal. [In Portuguese].

Bean, J. (1994). Genetic algorithms and random keys for sequencing and optimization. *ORSA Journal on Computing*, *6*, 154–160.

Beasley, J. (1983). Route-first cluster-second methods for vehicle routing. *Omega*, *11*, 403–408.

Beasley, J. (1987). An algorithm for set covering problem. *European Journal of Operational Research*, *31*, 85–93.

Beasley, J. (1993a). Lagrangean heuristics for location problems. *European Journal of Operational Research*, *65*, 383–399.

Beasley, J. (1993b). Lagrangean relaxation. In C. Reeves (Ed.), *Modern heuristic techniques for combinatorial problems*. Blackwell Scientific Publications, 243–303.

Belenguer, J., Benavent, E., Prins, C., Prodhon, C., et Wolfler-Calvo, R. (2006a). A cutting plane method for the capacitated location-routing problem. In *Odysseus 2006 - Third International Workshop on Freight Transportation and Logistics*. Altea, Espagne, 50–52.

Belenguer, J., Benavent, E., Prins, C., Prodhon, C., et Wolfler-Calvo, R. (2006b). A cutting plane method for the capacitated location-routing problem. In *EURO'06*. Iceland.

Berger, J. et Barkaoui, M. (2003). A new hybrid genetic algorithm for the capacitated vehicle routing problem. *Journal of the Operational Research Society*, *54*, 1254–1262.

Berman, O., Jaillet, P., et Simchi-Levi, D. (1995). Location-routing problems with uncertainty. In Z. Drezner (Ed.), *Facility Location: a survey of applications and methods*. Springer, 427–452.

Binato, S., Hery, W., Loewenstern, D., et Resende, M. (2002). A GRASP for job shop scheduling. In C. Ribeiro et P. Hansen (Eds.), *Essays and Surveys in Metaheuristics*, Kluwer Academic Publishers. 59–79.

Boudia, M., Louly, M., et Prins, C. (.à paraître). A reactive GRASP and path relinking for a combined production-distribution problem. *Computers and Operations Research*.

Boudia, M., Louly, M., et Prins, C. (2006). A memetic algorithm with population management for a production-distribution problem. In A. Dolgui, G. Morel, et C. Pereira (Eds.), *Preprints of the 12th IFAC Symposium on Information Control Problems in Manufacturing INCOM'06*. Saint Etienne-France, vol. 3, 541–546.

Boulanger, C., Semet, F., et Vaca-Arellano, P. (2006). An integrated bicriteria model and a heuristic approach for a hazardous wastes transportation problem. In *Odysseus 2006 - Third International Workshop on Freight Transportation and Logistics*. Altea, Espagne, 66–69.

Bruns, A. et Klose, A. (1995). An iterative heuristic for location-routing problems based on clustering. In *Proceedings of the second International Workshop on Distribution Logistics*. Oegstgeest, The Netherlands.

Bruns, A. et Klose, A. (1996). A "locate first - route second" heuristic for a combined location-routeing problem. In U. Zimmermann, U. Derigs, W. Gaul, R. Mohring, et K. Schuster (Eds.), *Operations Research Proceedings*. Springer.

Caballero, R., González, M., Guerrero, F., Molina, J., et Paralera, C. (.à paraître). Solving a multiobjective location routing problem with a metaheuristic based on tabu search. Application to a real case in Andalusia. *European Journal of Operational Research*.

Campos, V., Marti, R., et Laguna, M. (2005). Context-independent scatter search and tabu search for permutation problems. *INFORMS Journal on Computing*, *17*, 111–122.

Cappanera, P., Gallo, G., et Maffioli, F. (2003). Discrete facility location and routing of obnoxious activities. *Discrete Applied Mathematics*, *133*(1-3), 3–28.

Caprara, A., Fischetti, M., et Toth, P. (1999). A heuristic method for the set covering problem. *Operations Research*, *47*, 730–743.

Caprara, A., Toth, P., et Fischetti, M. (2000). Algorithms for the set covering problem. *Annals of Operations Research*, *98*, 353–371.

Carpaneto, G., Fischetti, M., et Toth, P. (1989). New lower bounds for the symmetric travelling salesman problem. *Mathematical Programming*, *45*, 233 – 254.

Carpaneto, G. et Toth, P. (1980). Some new branching and bounding criteria for the asymmetric travelling salesman problem. *Management Science*, *26*(7), 736–743.

Carreto, C. et Baker, B. (2002). A GRASP interactive approach to the vehicle routing problem with backhauls. In C. Ribeiro et P. Hansen (Eds.), *Essays and Surveys in Metaheuristics*, Kluwer Academics Publishers. 185–199.

Ceria, S., Nobili, P., et Sassano, A. (1998). A lagrangian-based heuristic for large-scale set covering problems. *Mathematical Programming*, *81*, 215–228.

Chan, Y., Carter, W., et Burnes, M. (2001). A multiple-depot, multiple-vehicle, location-routing problem with stochastically processed demands. *Computers and Operations Research*, *28*, 803–826.

Chien, T. W. (1993). Heuristic procedures for practical-sized uncapacitated location-capacitated routing problems. *Decision Science*, *24*(5), 995–1021.

Christofides, N. et Beasley, J. (1982). A tree search algorithm for the p-median problem. *European Journal of Operational Research*, *10*, 196–204.

Christofides, N. et Eilon, S. (1969). An algorithm for the vehicle dispatching problem. *Operational Research Quarterly*, *20*(3), 309–318.

Chvàtal, V. (1973). Edmonds polytopes and a hierarchy of combinatorial problems. *Discrete Mathematics*, *4*, 305–337.

Chvàtal, V. (1979). Greedy heuristic for the set-covering problem. *Mathematics of Operations Research*, *4*, 233–235.

Clarke, G. et Wright, J. (1964). Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, *12*, 568–581.

Colorni, A., Dorigo, M., Maniezzo, V., et Trubian, M. (1991). Ant system for job-shop scheduling. *Belgian Journal of Operations Research, Statistics and Computer Science*, *34*, 39–53.

Cook, W., Cunningham, W., Pulleyblank, W. R., et Schrijver, A. (1998). *Combinatorial Optimization*. New York: Wiley and Son.

Cordeau, J.-F., Gendreau, M., Hertz, A., Laporte, G., et Sormany, J. (2005). New heuristics for the vehicle routing problem. In A. Langevin et D. Riopel (Eds.), *Logistics Systems - Design and Optimization*. Springer, 279–298.

Cordeau, J.-F. et Laporte, G. (2002). Tabu search heuristics for the vehicle routing problem. Tech. Rep. G-2002-15, GERAD.

Cordeau, J.-F., Laporte, G., et Mercier, A. (2001). A unified tabu search heuristic for vehicle routing problems with time windows. *Journal of the Operational Research Society*, *52*, 928–936.

Cordone, R., Ferrandi, F., Sciuto, D., et Wolfler Calvo, R. (2001). An efficient heuristic approach to solve the unate covering problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, *20*, 1377–1388.

Cortinhal, M.-J. et Captivo, M.-E. (2003). Upper and lower bounds for the single source capacitated location problem. *European Journal of Operational Research*, *151*, 333–351.

Crainic, T. et Semet, F. (2006). *Optimisation combinatoire 3 : applications*, Hermès-Lavoisier, chap. Recherche opérationnelle et transport de marchandises. 47–116.

Dantzig, G., Fulkerson, D., et Johnson, S. (1954). Solution of a large-scale travelling-salesman problem. *Operations Research*, *2*, 393–410.

Delmaire, H., Díaz, J., Fernández, E., et Ortega, M. (1999). A reactive GRASP and tabu search based heuristics for the single source capacitated plant location problem. *INFOR*, *37*, 194–225.

DePuys, G., Moraga, R., et Whitehouse, G. (2005). Meta-RaPS: a simple and effective approach for solving the traveling salesman problem. *Transportation Research Part E: Logistics and Transportation Review*, *41*, 115–130.

Desrochers, M. et Verhoog, T. (1989). A matching based savings algorithm for the vehicle routing problem. *Les Cahiers du GERAD G-89-04 - Ecole des Hautes Etudes Commerciales de Montréal*.

Dijkstra, E. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, *1*, 269–271.

Dongarra, J. J. (1985, revisé en 2006). Performance of various computers using standard linear equations software. Technical Report CS - 89 - 85, University of Tennessee.

Dorigo, M., Maniezzo, V., et Colorni, A. (1996). The ant system: Optimization by a colony of cooperating agents. *IEEE Tansactions on Systems, Man and Cybernetics-Part B*, *26*, 29–41.

Drezner, Z. (2003). A new genetic algorithm for the quadratic assignment problem. *INFORMS Journal on Computing*, *15*, 320–330.

Ergun, O., Orlin, J., et Steele-Feldman, A. (2002). Creating very large scale neighborhoods out of smaller ones by compounding moves: A study on the vehicle routing problem. Working Paper 4393-02, MIT Sloan School of Management, Cambridge, MA, USA.

Erlenkotter, D. (1978). A dual-based procedure for uncapacitated facility location. *Operations Research*, *26*, 992–1009.

Fahrion, R. et Wrede, M. (1990). On a principle of chain-exchange for vehicle-routeing problems (1-VRP). *Journal of the Operational Research Society*, *41*(9), 821–827.

Feo, T. et González-Velarde, J. (1995). The intermodal trailer assignment problem: Models, algorithms, and heuristics. *Transportation Science*, *29*, 330–341.

Feo, T. et Resende, M. (1989). A probabilistic heuristic for a computationally difficult set covering problem. *Operations Research Letters*, *8*, 67–71.

Feo, T. et Resende, M. (1995). Greedy randomized adaptive search procedures. *Journal of Global Optimization*, *2*, 1–27.

Feo, T., Resende, M., et Smith, S. (1994). A greedy randomized adaptive search procedure for maximum independent set. *Operations Research*, *42*, 860–878.

Fischetti, M. et Toth, P. (1989). An additive bounding procedure for combinatorial optimization problems. *Operations Research*, *37*, 319–328.

Fischetti, M. et Toth, P. (1992). An additive bounding procedure for the asymmetric travelling salesman problem. *Operations Research*, *53*, 173 – 197.

Fischetti, M., Toth, P., et Vigo, D. (1994). A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs. *Operations Research*, *42*, 846–859.

Fisher, M. (1981). The lagrangian relaxation method for solving integer programming problems. *Management Science*, *27*, 1–18.

Fisher, M. et Jaikumar, R. (1981). A generalized assignement heuristic for vehicle routing. *Networks*, *11*, 109–124.

Fleurent, C. et Glover, F. (1999). Improved constructive multistart strategies for the quadratic assignment problem using adaptative memory. *INFORMS Journal on Computing*, *11*, 198–204.

Gendreau, M., Hertz, A., et Laporte, G. (1992). New insertion and post-optimization procedures for the traveling salesman problem. *Operations Research*, *40*, 1086–1094.

Gendreau, M., Hertz, A., et Laporte, G. (1994). A tabu search heuristic for the vehicle routing problem. *Management Science*, *40*, 1276–1290.

Gendreau, M., Laporte, G., et Potvin, J.-Y. (2001). *The vehicle routing problem*, Society for Industrial and Applied Mathematics, chap. Metaheuristics for the Vehicle Routing Problem. 129–154.

Gendreau, M., Laporte, G., et Vigo, D. (1999). Heuristics for the traveling salesman problem with pickup and delivery. *Computers and Operations Research*, *26*, 699–714.

Geoffrion, A. (1970a). Elements of large scale mathematical programming - part I: Concepts. *Management Science*, *15*, 652–675.

Geoffrion, A. (1970b). Elements of large scale mathematical programming - part II: Synthesis of algorithms and bibliography. *Management Science*, *16*, 676–691.

Geoffrion, A. (1974). Lagrangian relaxation for integer programming. *Mathematical Programming Study*, *2*, 82–114.

Ghaziri, H. (1996). Supervision in the self-organizing feature map: application to the vehicle routing problem. In I. Osman et J. Kelly (Eds.), *Meta-Heuristics: Theory and Applications*. Boston: Kluwer, 651–660.

Ghiani, G. et Laporte, G. (2001). Location-arc routing problems. *OPSEARCH*, *38*, 151–159.

Gibbons, A. (1985). *Algorithmic Graph Theory*. Cambridge.

Gillett, B. et Miller, L. (1974). A heuristic algorithm for the vehicle dispatch problem. *Operation Research*, *22*, 340–349.

Glover, F. (1986). Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, *5*, 533–549.

Glover, F. (1989). Tabu search. part I. *ORSA Journal on Computing*, *1*, 190–206.

Glover, F. (1990). Tabu search. part II. *ORSA Journal on Computing*, *2*, 4–32.

Glover, F. et Laguna, M. (1997). *Tabu Search*. Boston, USA: Kluwer Academic Publishers.

Glover, F., Laguna, M., et Martí, R. (2000). Fundamentals of scatter search and path relinking. *Control and Cybernetics*, *39*, 653–684.

Glover, F., Laguna, M., et Martí, R. (2003). Scatter search. In A. Ghosh et S. Tsutsui (Eds.), *Advances in Evolutionary Computation: Theory and Applications*. New York, 519–537.

Golden, B., Magnanti, T., et Nguyen, H. (1977). Implementing vehicle routing algorithms. *Networks*, *7*, 113–148.

Gomory, R. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, *64*, 275–278.

Gomory, R. (1963). An algorithm for integer solutions to linear programs. In R. Graves et P. Wolfe (Eds.), *Recent Advances in Mathematical Programming*. New York: McGraw-Hill, 269–302.

Gondran, M. et Minoux, M. (1995). *Graphes et Algorithmes*. Eyrolles.

Hadjiconstantinou, E., Christofides, N., et Mingozzi, A. (1995). A new exact algorithm for the vehicle routing problem based on q-paths and k-shortest paths relaxations. *Annals of Operations Research*, *61*, 21–43.

Hakimi, S. (1964). Optimum location of switching centers and the absolute centers and medians of a graph. *Operations Research*, *12*, 450–459.

Handler, G. (1979). Complexity and efficiency in minimax network location. In N. Christofides, A. Mingozzi, P. Toth, et C. Sandi (Eds.), *Combinatorial Optimization*. John Wiley & Sons, 281–314.

Harche, F. et Thompson, G. (1994). The column subtraction algorithm: An exact method for solving weighted set covering, packing and partitioning problemsnext term. *Computers and Operations Research*, *21*, 689–705.

Harrison, H. (1979). A planning system for facilities and resources in distribution networks. *Interfaces*, *9*(2), 6–22.

Held, M. et Karp, R. (1970). The traveling-salesman problem and minimum spanning trees. *Operations Research*, *18*, 1138–1162.

Held, M. et Karp, R. (1971). The traveling-salesman problem and minimum spanning trees: Part ii. *Mathematical Programming*, *1*, 6–25.

Helsgaun, K. (2000). An effective implementation of the lin-kernighan traveling salesman heuristic. *European Journal of Operational Research*, *126*, 106–130.

Higgins, J. (1972). On the merits of simple models in distribution planning. *International Journal of Physical Distribution*, *2*(2), 144–148.

Holland, J. (1975). Adaptation in natural and artificial systems. Tech. rep., The University of Michigan Press, Ann Arbor, MI.

Holmberg, K., Rönnqvist, M., et Yuan, D. (1999). An exact algorithm for the capacitated facility location problems with single sourcing. *European Journal of Operational Research*, *113*, 544–559.

Jacobsen, S. et Badsen, O. (1980). A comparative study of heuristics for a two-level routing-location problem. *European Journal of Operational Research*, *6*, 378–387.

Jaramillo, J., Bhadury, J., et Batta, R. (2002). On the use of genetic algorithms to solve location problems. *Computers and Operations Research*, *29*, 761–779.

Johnson, D. et Papadimitrious, C. (1985). Computational complexity. In E. Lawler, J. Lenstra, A. R. Kan, et D. Shmoys (Eds.), *The traveling Salesman Problem*. Chichester: John Wiley & Sons, 37–86.

Kawamura, H., Yamamoto, M., Mitamura, T., Suzuki, K., et Ohuchi, A. (1998). Cooperative search on pheromone communication for vehicule routing problem. *IEEE Transactions on Fundamentals*, *E81-A*, 1089–1096.

Kirkpatrick, S., C.D. Gelatt, J., et Vecchi, M. (1983). Optimization by simulated annealing. *Science*, *220*(4598), 671–680.

Klincewicz, J. (1992). Avoiding local optima in the p-hub location problem using tabu search and GRASP. *Annals of Operations Research*, *40*, 283–302.

Klincewicz, J. et Luss, H. (1986). A lagrangian relaxation heuristic for capacitated facility location with single source constraints. *Journal of the Operational Research Society*, *37*, 495–500.

Klose, A. et Drexl, A. (2005). Facility location models for distribution system design. *European Journal Of Operational Research*, *162*, 4–29.

Kontoravdis, G. et Bard, J. (1995). A GRASP for the vehicle routing problem with time windows. *ORSA Journal on Computing*, *7*, 10–23.

Labadi, N. (2003). *Problèmes tactiques et stratégiques en tournées sur arcs*. Ph.D. thesis, Université de Technologie de Troyes.

Lacomme, P., Prins, C., et Sevaux, M. (2003). *Algorithmes de graphes*. Paris: Eyrolles.

Laguna, M. et Marti, R. (1995). GRASP and path relinking for 2-layer straight line crossing minimization. *INFORMS Journal on Computing*, *11*(1), 44–52.

Laporte, G. (1988). Location-routing problems. In B. Golden et A. Assad (Eds.), *Vehicle Routing: Methods and Studies*. Amsterdam: North Holland, 163–196.

Laporte, G. et Dejax, P. (1989). Dynamic location-routeing problems. *Journal of the Operational Research Society*, *40*(5), 471–482.

Laporte, G., Gendreau, M., Potvin, J.-Y., et Semet, F. (2000). Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research*, *7*, 285–300.

Laporte, G., Louveaux, F., et Mercure, H. (1989). Models and exact solutions for a class of stochastic location-routing problems. *European Journal of Operational Research*, *39*, 71–78.

Laporte, G. et Norbert, Y. (1981). An exact algorithm for minimizing routing and operating costs in depot location. *European Journal of Operational Research*, *6*, 224–226.

Laporte, G. et Norbert, Y. (1987). Exact algorithms for the vehicle routing problem. *Annals of Discrete Mathematics*, *31*, 147–184.

Laporte, G., Norbert, Y., et Arpin, D. (1986). An exact algorithm for solving a capacitated location-routing problem. *Annals of Operations Research*, *6*, 293–310.

Laporte, G., Norbert, Y., et Taillefer, S. (1988). Solving a family of multi-depot vehicle routing and location-routing problems. *Transportation Science*, *22*, 161–167.

Lawler, E., Lenstra, J., Kan, A. R., et Shmoys, D. (1985). *The traveling Salesman Problem*. Chichester: John Wiley & Sons.

Lawrence, R. et Pengilly, P. (1969). The number and location of depots required for handling products for distribution to retail stores in south-east England. *Operational Research Quarterly*, *20*(1), 23–32.

Leung, L. et Magnanti, T. (1989). Valid inequalities and facets of the capacitated plant location problems. *Mathematical Programming*, *44*, 271–291.

Lima, C., Goldbarg, M., et Goldbarg, E. (2004). A memetic algorithm for heterogeneous fleet vehicle routing problem. *Electronic Notes in Discrete Mathematics*, *18*, 171–176.

Lin, C., Chow, C., et Chen, A. (2002). A location-routing-loading problem for bill delivery services. *Computers and Industrial Engineering*, *43*, 5–25.

Lin, C. et Kwok, R. (.à paraître). Multi-objective metaheuristics for a location-routing problem with multiple use of vehicles on real data and simulated data. *European Journal of Operational Research*.

Lin, S. et Kernighan, B. (1973). An effective heuristic algorithm for the traveling salesman problem. *Operations Research*, *21*, 498–516.

List, G. et Mirchandani, P. (1991). An integrated network/planar multiobjective model for routing and siting for hazardous materials and wastes. *Transportation Science*, *25*(2), 146–156.

Little, J. D. C., Murty, K. G., Sweeney, D. W., et Karel, C. (1963). An algorithm for the traveling salesman problem. *Operations Research*, *11*(6), 972–989.

Lorena, L. et Senne, E. (2004). A column generation approach to capacitated p-median problems. *Computers and Operations Research*, *31*, 863–876.

Lu, Q. et Dessouky, M. (2004). An exact algorithm for the multiple vehicle pickup and delivery problem. *Transportation Science*, *38*(4), 503–514.

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

Madsen, O. (1983). Methods for solving combined two level location-routing problems of realistic dimensions. *European Journal of Operational Research*, *12*, 295–301.

Maranzana, F. (1964). On the location of supply points to minimize transport costs. *Operational Research Quarterly*, *15*, 261–270.

Melechovský, J., Prins, C., et Calvo, R. W. (2005). A metaheuristic to solve a location-routing problem with non-linear costs. *Journal of Heuristics*, *11*, 375–391.

Mester, D. et Bräysy, O. (.à paraître). Active guided evolution strategies for large scale vehicle routing problems. *Computers and Operations Research*.

Min, H., Jayaraman, V., et Srivastava, R. (1998). Combined location-routing problems: A synthesis and future research directions. *European Journal of Operational Research*, *108*, 1–15.

Mingozzi, A. et Valletta, A. (2003). An exact algorithm for period and multi-depot vehicle routing problems. In *Odysseus 2003 - Second International Workshop on Freight Transportation and Logistics*. Palermo, Italy.

Mirchandani, P. et Francis, R. (1990). *Discrete Location Theory*. New York: John Wiley & Sons Inc.

Mladenović, N., Brimberg, J., Hansen, P., et Moreno-Pérez, J. (.à paraître). The p-median problem: A survey of metaheuristic approaches. *European Journal of Operational Research*.

Mladenović, N., Labbé, M., et Hansen, P. (2003). Solving the p-center problem with tabu search and variable neighborhood search. *Networks*, *42*, 48 – 64.

Moscato, P. (1989). On evolution, search, optimization, genetic algorithms, martial arts: towards memetic algorithms. Caltech Concurrent Computation Program C3P Report 826, California Institute of Technology.

Moscato, P. et Cotta, C. (2003). A gentle introduction to memetic algorithms. In F. Glover et G. Kochenberger (Eds.), *Handbook of Metaheuristics*. Boston: Kluwer Academic Publishers, 105–144.

Naddef, D. et Thienel, S. (2002a). Efficient separation routines for the symmetric traveling salesman problem i: general tools and comb separation. *Mathematical Programming*, *92*, 237 – 255.

Naddef, D. et Thienel, S. (2002b). Efficient separation routines for the symmetric traveling salesman problem ii: separating multi handle inequalities. *Mathematical Programming*, *92*, 257 – 283.

Nagy, G. et Salhi, S. (1996). Nested heuristic methods for the location-routeing problem. *Journal of the Operational Research Society*, *47*, 1166–1174.

Nambiar, J., Gelder, L., et Wassenhove, L. V. (1981). A large scale location-allocation problem in the natural rubber industry. *European Journal of Operational Research*, *6*, 183–189.

Nemhauser, G. et Wolsey, L. (1999). *Integer and Combinatorial Optimization*. New York: Wiley.

Or, I. (1976). *Traveling salesman-type combinatorial optimization problems and their relation to the logistics of regional blood banking*. Ph.D. dissertation, Northwestern University, Evanston, IL.

Or, I. et Pierskalla, W. (1979). A transportation, location-allocation model for regional blood banking. *AIIE Transactions*, *11*(2), 86–95.

Osman, I. (1993). Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, *41*, 421–451.

Padberg, M. et Rinaldi, G. (1991). A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM Review*, *33*, 60–100.

Paessens, H. (1988). The savings algorithm for the vehicle routing problem. *European Journal of Operational Research*, *34*, 336–344.

Paschos, V. (2005). *Optimisation combinatoire 1 : concepts fondamentaux*. Hermès-Lavoisier.

Perl, J. et Daskin, M. (1985). A warehouse location-routing problem. *Transportation Research*, *19B*(5), 381–396.

Piñana, E., Plana, I., Campos, V., et Martí, R. (2004). GRASP and path relinking for the matrix bandwidth minimization. *European Journal of Operational Research*, *153*, 200–210.

Pirkul, H. (1987). Efficient algorithm for the capacitated concentrator location problem. *Computers and Operations Research*, *14*, 197–208.

Pisinger, D. (1997). A minimal algorithm for the 0-1 knapsack problem. *Operations Research*, *45*, 758–767.

Potvin, J.-Y. (1996). Genetic algorithms for the traveling salesman problem. *Annals of Operations Research*, *63*, 337–370.

Potvin, J.-Y. et Rousseau, J.-M. (1995). An exchange heuristic for routeing problems with time windows. *Journal of the Operational Research Society*, *46*(12), 1433–1446.

Prais, M. et Ribeiro, C. (2000). Reactive GRASP: an application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, *12*(3), 164–176.

Preparata, F. P. et Shamos, M. I. (1985). *Computational Geometry: An Introduction*. New York: Springer.

Prins, C. (2004). A simple and effective evolutionary algorithm for the vehicle routing problem. *Computers and Operations Research*, *31*, 1985–2002.

Prins, C., Prodhon, C., Ruiz, A., Soriano, P., et Wolfler-Calvo, R. (2005a). A cooperative lagrangean relaxation-granular tabu search heuristic for the capacitated location routing problem. In *MIC'05*. Vienna, Austria.

Prins, C., Prodhon, C., Ruiz, A., Soriano, P., et Wolfler-Calvo, R. (2005b). A new metaheuristic combining a lagrangean relaxation with a tabu search approach for the location-routing problem. In *AIRO'05*. Camerino, Italy.

Prins, C., Prodhon, C., et Wolfler-Calvo, R. (2004a). Nouveaux algorithmes pour le problème de localisation et routage sous contraintes de capacité. In A. Dolgui et S. Dauzère-Pérès (Eds.), *MOSIM'04*. Ecole des Mines de Nantes: Lavoisier, vol. 2, 1115–1122.

Prins, C., Prodhon, C., et Wolfler-Calvo, R. (2005c). A reactive GRASP and path relinking algorithm for the capacitated location routing problem. In *CD proceedings of the International Conference on Industrial Engineering and Systems Management (IESM'05)*. Marrakech, Morocco: I4E2.

Prins, C., Prodhon, C., et Wolfler-Calvo, R. (2006a). A memetic algorithm with population management ($MA \mid PM$) for the capacitated location-routing problem. In J.Gottlieb et G. R. Raidl (Eds.), *Lecture Notes in Computer Science*. Proceedings of EvoCOP2006 (Evolutionary Computation in Combinatorial Optimization: 6th European Conference, Budapest, Hungary, April 10-12, 2006), Springer, vol. 3906, 183–194.

Prins, C., Prodhon, C., et Wolfler-Calvo, R. (2006b). Solving the capacitated location-routing problem by a GRASP complemented by a learning process and a path relinking. *4OR - A Quarterly Journal of Operations Research*, *4*, 221–238.

Prins, C., Prodhon, C., et Wolfler-Calvo, R. (En révision). Solving the capacitated location-routing problem by a cooperative lagrangean relaxation-granular tabu search heuristic. *Transportation Science*.

Prins, C., Sevaux, M., et Sörensen, K. (2004b). A genetic algorithm with population management ($GA \mid PM$) for the carp. In *Tristan V (5th Triennal Symposium on Transportation Analysis)*. Le Gosier, Guadeloupe.

Reeves, C. (2003). Genetic algorithms. In F. Glover et G. Kochenberger (Eds.), *Handbook of Metaheuristics*. Springer, 55–82.

Rego, C. (1998). A subpath ejection method for the vehicle routing problem. *Management Science*, *44*(10), 1447–1459.

Rego, C. et Roucairol, C. (1996). A parallel tabu search algorithm using ejection chains for the vehicle routing problem. In *Meta-Heuristics: Theory and Applications*. Boston: Kluwer, 661–675.

Reimann, M., Doerner, K., et Hartl, R. (2004). D-ants: savings based ants divide and conquer the vehicle routing problem. *Computers and Operations Research*, *31*(4), 563–591.

Renaud, J., Boctor, F., et Laporte, G. (1996a). A fast composite heuristic for the symmetric traveling salesman problem. *INFORMS Journal on Computing*, *8*(2), 134–143.

Renaud, J., Laporte, G., et Boctor, F. (1996b). A tabu search heuristic for the multi-depot vehicle routing problem. *Computers and Operations Research*, *23*(3), 229–235.

Resende, M. et Ribeiro, C. (2003). Greedy randomized adaptive search procedures. In F. Glover et G. Kochenberger (Eds.), *Handbook of Metaheuristics*, Kluwer Academic Publishers. 219–249.

Resende, M. et Ribeiro, C. (2005). GRASP with path relinking: Recent advances and applications. In T. Ibaraki, K. Nonobe, et M. Yagiura (Eds.), *Metaheuristics: Progress as Real Problem Solvers*, Kluwer Academic Publishers.

Resende, M. et Werneck, R. (2003). A fast swap-based local search procedure for location problems. Tech. rep., Internet and Network Systems Research Center, AT&T Labs Research, Florham Park, NJ. Submitted to *Discrete Applied Mathematics*.

Resende, M. et Werneck, R. (2004). A hybrid heuristic for the p-median problem. *Journal of Heuristics*, *10*, 59–88.

Rolland, E., Schilling, D., et Current, J. (1997). An efficient tabu search procedure for the p-median problem. *European Journal of Operational Research*, *96*, 329–342.

Rönnqvist, M., Tragantalerngsak, S., et J.Holt (1999). A repeated matching heuristic for the single-source capacitated facility location problem. *European Journal of Operational Research*, *116*, 51–68.

Russell, R. et Chiang, W. (2006). Scatter search for the vehicle routing problem with time windows. *European Journal of Operational Research*, *169(2)*, 606–622.

Ryan, D. M., Hjorring, C., et Glover, F. (1993). Extensions of the petal method for vehicle routeing. *Journal of the Operational Research Society*, *44*(3), 289–296.

Salhi, S. et Rand, G. K. (1989). The effect of ignoring routes when locating depots. *European Journal of Operational Research*, *39*, 150–156.

Schmitt, L. et Amini, M. (1998). Performance characteristics of alternalte genetic algorithmic approaches to the traveling salesman problem using path representation : An empirical study. *European Journal of Operational Research*, *108*(3), 551–570.

Schrijver, A. (1998). *Theory of Linear and Integer Programming*. Chichester: Wiley.

Senne, E., Lorena, L., et Pereira, M. (2005). A branch-and-price approach to p-median location problems. *Computers and Operations Research*, *32*, 1655–1664.

Shamos, M. et Hoey, D. (1976). Geometric intersection problems. In *Proceedings of the 17th Annual IEEE Symposium on Foundations of Computer Science*. Houston, TX, 208–215.

Shapiro, J. (1979). A survey of lagrangian techniques for discrete optimization. *Annals of Discrete Mathematics*, *5*, 113–138.

Sörensen, K. et Sevaux, M. (2003). *GA | PM* : genetic algorithms with population management for permutation problems. In *MIC2003*.

Sörensen, K. et Sevaux, M. (2006). *MA | PM* : memetic algorithms with population management. *Computers and Operations Research*, *33*, 1214–1225.

Sridharan, R. (1993). A lagrangian heuristic for the capacitated plant location problem with single source constraints. *European Journal of Operational Research*, *66*, 305–312.

Srivastava, R. (1993). Alternate solution procedures for the locating-routing problem. *OMEGA International Journal of Management Science*, *21*(4), 497–506.

Taillard, E. (1993). Parallel iterative search methods for vehicle routing problem. *Networks*, *23*, 661–673.

Tamir, A. (1993). Complexity results for the p-median problem with mutual communication. *Operations Research Letters*, *14*, 79–84.

Tarantilis, C. et Kiranoudis, C. (2002). Bone route: An adaptive memory-based method for effective fleet management. *Annals of Operations Research*, *115*, 227–241.

Thompson, P. et Psaraftis, H. (1993). Cyclic transfert algorithms for the multi-vehicle routing and scheduling problems. *Operations Research*, *41*, 935–946.

Toth, P. et Vigo, D. (Eds.) (2001). *The vehicle routing problem*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics.

Toth, P. et Vigo, D. (2002). Models, relaxations and exact approaches for the capacitated vehicle routing problem. *Discrete Applied Mathematics*, *123*, 487–512.

Toth, P. et Vigo, D. (2003). The granular tabu search (and its application to the vehicle routing problems. *INFORMS Journal on Computing*, *15*(4), 333–346.

Tsai, C., Tsai, C., et Tseng, C. (2004). A new hybrid heuristic approach for solving large traveling salesman problem. *Information Sciences*, *166*, 67–81.

Tsubakitani, S. et Evans, J. (1998). An empirical study of a new metaheuristic for the traveling salesman problem. *European Journal of Operational Research*, *104*, 113–128.

Tuzun, D. et Burke, L. (1999). A two-phase tabu search approach to the location routing problem. *European Journal of Operational Research*, *116*, 87–99.

Van Breedam, A. (1996). An analysis of the effect of local improvement operators in genetic algorithms and simulated annealing for the vehicle routing problem. Tech. rep., RUCA 96/14, University of Antwerp, Belgium.

Volgenant, A. et Jonker, R. (1983). The symmetric traveling salesman problem and edges exchanges in minimal 1-tree. *European Journal of Operational Research*, *12*, 394–403.

Von Boventer, E. (1961). The relationship between transportation costs and location rent in transportation problems. *Journal of Regional Science*, *3*, 27–40.

Watson-Gandy, C. et Dohrn, P. (1973). Depot location with van salesmen - a practical approach. *Omega*, *1*(3), 321–329.

Webb, M. (1968). Cost functions in the location of depots for multi-delivery journeys. *Operational Research Quarterly*, *19*(3), 311–320.

Wolsey, L. (1998). *Integer Programming*. New York: Wiley.

Wu, T., Low, C., et Bai, J. (2002). Heuristic solutions to multi-depot location-routing problems. *Computers and Operations Research*, *29*, 1393–1415.

Yellen, J. et Gross, J. (1998). *Graph theory and its applications*. CRC Press.

Yellow, P. (1970). A computational modification to the savings method of vehicle scheduling. *Operational Research Quarterly*, *21*, 281–283.

Zhang, J., Chen, B., et Ye, Y. (2004). A multi-exchange local search algorithm for the capacitated facility location problem. In D. Bienstock et G. Nemhauser (Eds.), *Integer Programming and Combinatorial Optimization: 10th International IPCO Conference*. New York, NY, USA: Springer Berlin / Heidelberg, vol. 3064 / 2004, 219–233.