Conception et évaluation d'outils décisionnels pour des systèmes réactifs d'aide à la mobilité

Libo Ren

N° d'ordre : 2275
EDSPIC : 574

Université Blaise Pascal – Clermont-Ferrand II

ÉCOLE DOCTORALE
SCIENCES POUR L'INGÉNIEUR DE CLERMONT-FERRAND

THÈSE

présentée par

Libo REN

pour obtenir le grade de

DOCTEUR D'UNIVERSITÉ

Spécialité : INFORMATIQUE

Conception et évaluation d'outils décisionnels pour des systèmes réactifs d'aide à la mobilité

Soutenue publiquement le 05 octobre 2012
Devant le jury composé de :

Rapporteurs :

Caroline PRODHON	Maître de conférences (HDR) à l'Université de Technologie de Troyes
Aziz MOUKRIM	Professeur à l'Université de Technologie de Compiègne

Examinateurs :

Alexandre DOLGUI	Professeur à l'École des Mines de Saint-Étienne
Philippe LACOMME	Maître de conférences (HDR) à l'Université Blaise Pascal

Directeurs de thèse :

Alain QUILLIOT	Professeur à l'Université Blaise Pascal
Christophe DUHAMEL	Maître de conférences à l'Université Blaise Pascal

Thèse préparée au sein du LIMOS - UMR 6158, CNRS.

Remerciements

Tout d'abord, je tiens à remercier mes deux directeurs de thèse, M. Alain Quilliot et M. Christophe Duhamel, pour m'avoir permis d'effectuer cette thèse dans les meilleures conditions au sein du LIMOS. Je leur témoigne toute ma reconnaissance pour leur disponibilité, leur gentillesse, leurs idées et leurs encouragements tout au long de ces quatre années de thèse.

Je remercie M. Alexandre Dolgui pour avoir accepté d'être président du jury. Je remercie également Mme. Caroline Prodhon et M. Aziz Moukrim, d'avoir accepté d'être rapporteurs. Je les remercie pour le temps qu'ils ont consacré et pour leurs remarques constructives qui m'ont permis d'améliorer le manuscrit.

Je remercie tout particulièrement M. Philippe Lacomme pour son aide précieuse et ses conseils apportés pendant toutes ces années de thèse.

Je remercie également mes collègues, en particulier Andréa, Heitor, Hélène, Zhao, Raksmey, Djelloul, Nardjes, Olivier, Jérémy, Jonathan et Diyé, pour m'avoir accueilli dans un environnement de travail convivial, pour leurs idées et pour leurs aides.

Je remercie aussi mes amis : Datta, Huijuan, Rui, Jocelyn, Michelle, Sun et Xing, pour leurs soutiens et pour les moments de détente qu'on a partagés. Plus particulièrement, je remercie M. Jacques Barnouin, pour ses conseils et ses idées philosophiques.

Enfin et surtout, je tiens à remercier mes parents qui m'ont supporté durant toutes ces années, aussi pour leur confiance et leurs encouragements. Je remercie particulièrement mon épouse, pour le soutien moral qu'elle m'a apporté depuis le début de ma thèse.

Résumé

Dans le cadre de cette thèse, nous nous intéressons au traitement des problèmes d'optimisation combinatoire liés à la conception d'outils de gestion des systèmes de véhicules partagés. Ces problèmes sont proches des problèmes de collecte et de livraison. Après avoir réalisé une étude théorique sur des problèmes d'optimisation combinatoire autour du transport et des méthodes de résolutions, nous nous sommes intéressés ici à trois problèmes particuliers : le PPRV, le PPRV-PM et le PPRV-T.

Le premier problème est le Problème de Planification du Redéploiement de Véhicules partagés (PPRV). C'est une extension du *One-commodity Pickup-and-Delivery Problem* (1-PDP) car les véhicules partagés sont indifférenciés. Nous avons proposé un modèle linéaire et une heuristique utilisant le schéma hybride ILS/VND. L'approche développée repose sur la stratégie « *route-first, cluster-second* » : on commence par construire une tournée géante, puis on l'améliore par une procédure de perturbation et une recherche locale. Pendant la recherche locale, la contrainte de capacité des véhicules est momentanément relaxée et progressivement restaurée ; la tournée géante obtenue est ensuite transformée en plusieurs tournées à l'aide de la procédure Split.

Les deux problèmes suivants sont considérés comme des extensions du PPRV en autorisant des livraisons partielles : PPRV avec Passage Multiple (PPRV-PM) et PPRV avec Transfert d'objets (PPRV-T). Nous proposons une approche de type « *divide-first, route-second* » pour la résolution du PPRV-PM. Elle consiste à effectuer d'abord un fractionnement de la demande, puis la résoudre à l'aide d'un schéma hybride de type GRASP/VND. Le PPRV-T étend le PPRV-PM au transfert d'objets entre les transporteurs lors du passage sur un sommet. Nous avons reformulé le PPRV-T comme un problème de multi-flots couplés sur un réseau dynamique. Nous avons proposé une méthode d'insertion basée sur cette modélisation.

Mots-clés : véhicule partagé, optimisation combinatoire, transport, collecte et livraison, heuristique.

Abstract

In this thesis, we are interested to deal with combinatorial optimization problems related to design management tools for vehicle-sharing systems. These problems are close to the Pickup-and-Delivery Problems (PDP) in the literature. After performing a survey on the problems area and on the resolution methods, we focused on three specific problems and we proposed one approach for each problem.

The first one is the sharing Vehicles Redeployment Planning Problem (VRPP), which is considered as a multi-vehicles extension of the One-commodity Pickup-and-Delivery Problem (1-PDP). We proposed a linear model and a hybrid heuristic which combines the ILS and VND. The proposed approach uses the *rout-first, cluster-second* strategy: we construct a Hamiltonian route, and then improve it using a procedure combines a shacking step and a VND local search. The used neighborhoods are adapted to the relaxation of capacity; the obtained route would be then split into several vehicles tours in the clustering phase.

The two following problems are considered as extensions of VRPP introducing the split demand constraint: VRPP with Multi-Passage (VRPP-MP) and VRPP with Transferring objects (VRPP-T). We proposed an approach with the *divide-first, route-second* strategy for VRPP-MP. It consists of dividing in advance the demand, and then solves it using a hybrid scheme of GRASP/VND. In the VRPP-T, the objects carried could be exchanged between carriers when crossing on the sites. The VRPP-T is modeled here as a multi-flows problem on a dynamic network. We proposed an insertion method based on this modeling.

Keywords: vehicle-sharing, combinatorial optimization, transport, pickup-and-delivery, heuristic.

Table des matières

Introduction **1**

1 État de l'art : problèmes et méthodes autour des tournées de véhicules **15**

1.1 Problèmes associés à la mobilité et au transport 15
1.1.1 Problèmes de tournées de type VRP 15
1.1.1.1 Définition du CVRP 16
1.1.1.2 Modèle linéaire du CVRP 16
1.1.2 Problèmes de tournées de type PDP 17
1.1.2.1 Définition du GPDP 18
1.1.2.2 Modèle linéaire du GPDP 18
1.1.3 Classification des problèmes de Pickup and Delivery 19
1.1.3.1 Caractérisation des requêtes 19
1.1.3.2 Critères de classification dans les problèmes de Pickup and Delivery 21
1.1.3.3 Classification des problèmes de Pickup and Delivery 22
1.1.4 Principaux problèmes impliquant le « *Pickup and Delivery* » 23
1.1.4.1 Problème de tournées de véhicules avec « *Backhaul* » (VRPB) 23
1.1.4.2 Problème de tournées de véhicules avec livraison et collecte (VRPPD) 24
1.1.5 Variantes dynamiques des problèmes de transport 26
1.1.5.1 Problème de tournées de véhicules dynamiques 26
1.1.5.2 Problèmes de *Pickup and Delivry* dynamiques 28
1.2 Méthodes d'optimisation 29
1.2.1 Méthodes exactes pour les problèmes de tournées de véhicules 29
1.2.1.1 Méthodes de type branchement 30
1.2.1.2 Programmation dynamique 33
1.2.1.3 Programmation linéaire en nombres entiers 35
1.2.2 Méthodes approchées pour les problèmes de tournées de véhicules 37
1.2.2.1 Heuristiques 37
1.2.2.2 Méta-heuristiques 44
1.3 Application des problèmes de flots sur le transport 52
1.3.1 Problèmes de « synthèse de réseaux » sur le transport 52
1.3.1.1 Principaux problèmes de flot 52
1.3.1.2 Problème du flot impliquant le problème de synthèse de réseaux 55
1.3.2 Principaux outils pour les problèmes de flots 55
1.3.2.1 Méthodes de décomposition en programmation linéaire et polyèdres 55
1.3.2.2 Réseaux de files d'attente et modèles de la théorie des jeux 61
1.4 Conclusion 62

2 Problème de la Planification du Redéploiement de Véhicules partagés : définition et modèle 63

2.1 Contexte du problème 63
2.1.1 Problème du Plan de Redéploiement (PPR) 64
2.1.2 PPRV : la version de base des problèmes de type PPR 67
2.1.3 Autres problèmes de type PPR 67
2.2 Définition du PPRV 69
2.3 Aperçu des problèmes proches du PPRV 70
2.4 Modélisation mathématique du PPRV 72
2.4.1 Données d'entrée et paramètres 72
2.4.2 Variables 72
2.4.3 Contraintes 72
2.4.4 Fonction objectif 73
2.4.5 Modèle mathématique du PPRV 73
2.4.6 Contraintes d'élimination de cycles 74
2.5 Expérimentations 77
2.5.1 Environnements de l'expérimentation et instances utilisées 78
2.5.2 Comparaison des contraintes d'élimination de cycles 78
2.5.3 Résultat pour le PPRV 79
2.6 Conclusion 82

3 Résolution approchée du PPRV 83

3.1 Étude des méthodes approchées 83
3.1.1 Méthodes approchées pour les problèmes de type 1-PDP 84
3.1.2 Relaxations de contrainte pour la résolution du PPRV 86
3.1.2.1 Relaxation de contrainte de capacité 86
3.1.2.2 Relaxation de contrainte mono-accès 87
3.2 Présentation de la méthode approchée 88
3.2.1 Schéma ILS/VND 89
3.2.2 Schéma général de la méthode proposée 91
3.2.3 Procédure de construction d'une tournée géante 92
3.2.3.1 Heuristique *Nearest Neighbour* randomisée 92
3.2.3.2 Modification de l'heuristique de Zhao *et al.* 93
3.2.3.3 Schéma de la procédure de construction d'une tournée géante 94
3.2.4 Procédure de perturbation 94
3.2.5 Transformation d'une tournée géante en une solution du PPRV 96
3.2.5.1 Split 96
3.2.5.2 Adaptation de Split au PPRV 99
3.2.5.3 Procédure de transformation 100
3.2.6 Recherches locales 101
3.2.6.1 Structures de voisinage et opérateurs d'amélioration locale 101
3.2.6.2 Opérateurs utilisés 102
3.2.6.3 Recherche locale pour la tournée géante (VND adaptée) 104
3.2.6.4 Recherche locale pour l'amélioration d'une solution PPRV 106
3.3 Expérimentations 106

3.3.1 Qualité des tournées géantes 107
3.3.2 Qualité des solutions du PPRV en multi-agents....111
3.4 Conclusion 112

4 PPRV avec passage multiple : définition et résolution 115

4.1 Introduction....115
4.2 Définition du PPRV avec Passage Multiple 117
4.3 Problèmes proches du PPRV-PM 119
4.4 Proposition d'une approche heuristique....121
4.4.1 Création d'une instance auxiliaire....121
4.4.1.1 Algorithme de création des couples « offre-demande » 122
4.4.1.2 Stratégies de la division de demande.... 126
4.4.2 Heuristique de résolution 128
4.4.2.1 Schéma de l'heuristique proposée 128
4.4.2.2 Procédure de construction d'une solution initiale 130
4.4.2.3 Opérateurs d'amélioration locale et Recherche locale 131
4.4.2.4 Projection d'une solution auxiliaire sur l'instance initiale 134
4.5 Expérimentations 134
4.5.1 Taille des instances auxiliaires....134
4.5.2 Opérateurs d'amélioration et la recherche locale 135
4.5.3 Résultats avec un agent 136
4.5.4 Résultats avec multi-agents....139
4.6 Conclusion 142

5 PPRV avec transfert d'objets : définition et modélisation 145

5.1 Introduction....145
5.2 Problème des Flots Entiers Couplés (PFEC) 147
5.2.1 Définition du Problème des Flots Entiers Couplés 147
5.2.2 Traitement du PFEC Acyclique dans un cadre statique 148
5.2.2.1 Modèle mathématique du PFEC-A 149
5.2.2.2 Application de la décomposition de Benders au PFEC-A.... 150
5.2.2.3 Schéma d'insertion pour le PFEC-A 153
5.3 Modélisation du PPRV-T comme PFEC-A posé sur un réseau dynamique 156
5.3.1 Reformulation du PPRV-T en PFEC-A posé sur un réseau dynamique 156
5.3.1.1 Création d'un réseau dynamique 156
5.3.1.2 Reformulation du PPRV-T.... 157
5.3.2 Application du schéma Insertion-PFEC-A au PFEC-A-Dyn 158
5.4 Expérimentations 159
5.5 Conclusion 160

Conclusion.... 161

Bibliographie.... 163

Liste des Figures

Figure 1 : Système de location de vélo en libre service 6
Figure 2 : Véhicule *CyCab (CyberCars)* 7
Figure 3 : Schéma d'un système CyCab 8
Figure 4 : Schéma hiérarchique des problèmes liés à la construction des outils 11
Figure 5 : Schéma hiérarchique des problèmes d'optimisation combinatoire 12

Figure 1.1 : Types de requêtes désagrégées 20
Figure 1.2 : Type de requêtes agrégées (type M-M) 21
Figure 1.3 : Classification pour les problèmes de *Pickup and Delivery* 23
Figure 1.4 : Problème du type 1-M-1 24
Figure 1.5 : Problème du type 1-1 25
Figure 1.6 : Problème du type M-M 26
Figure 1.7 : Solution d'un problème de tournées de véhicules dynamique à l'instant $t + \Delta$ 27
Figure 1.8 : Opération de séparation 30
Figure 1.9 : Contrainte de coupe 32
Figure 1.10 : Schémas arborescents 33
Figure 1.11 : Structure basique de la programmation dynamique 33
Figure 1.12 : Résolution d'un problème de tournées 34
Figure 1.13 : Processus général de la génération de colonne 36
Figure 1.14 : Illustration de la méthode de Clarke et Wright 39
Figure 1.15 : Solutions avec 2-pétales 41
Figure 1.16 : Problème de plus court chemin au partitionnement 42
Figure 1.17 : Opérateurs de transformation locale 43
Figure 1.18 : Mécanisme de type recherche voisinage 44
Figure 1.19 : Processus «λ-*interchange*» 46
Figure 1.20 : Génération d'une population 47
Figure 1.21 : Deux types de « *crossover* » 49
Figure 1.22 : Application de l'algorithme de colonies de fourmis 51

Figure 2.1 : Schéma hiérarchique des problèmes liés au redéploiement de convois 64
Figure 2.2 : Estimation de la demande 65
Figure 2.3 : Deux critères dans l'objectif d'un PPR 67
Figure 2.4 : Influence sur les critères d'optimisation 69
Figure 2.5 : Exemple du PPRV 70
Figure 2.6 : Elimination de cycle par CCC 75

Figure 3.1 : Application de la RCC 87
Figure 3.2 : Application de la RCA 88
Figure 3.3 : Relaxation du point de vue de l'espace des solutions 88
Figure 3.4 : Schéma de passage entre les deux espaces de solutions 89
Figure 3.5 : Split sur un problème de tournées de véhicules 96
Figure 3.6 : Recherche du plus court chemin à l'aide des étiquettes 98

Figure 3.7 : Découpage avec Split 98
Figure 3.8 : Application de Split pour le PPRV avec 2 agents 99
Figure 3.9 : Mouvement de l'opérateur *2-opt* intra-tournée 102
Figure 3.10 : Mouvements de l'opérateur *3-opt* intra-tournée à 3 arcs 103
Figure 3.11 : Mouvement de l'opérateur *swap* 104
Figure 3.12 : Mouvement de l'opérateur *2-opt* inter-tournée 104
Figure 3.13 : Mouvement de l'opérateur *2.5-opt* inter-tournée 104

Figure 4.1 : Influence sur la division de demande 116
Figure 4.2 : Hiérarchie des problèmes liés au redéploiement de convois 117
Figure 4.3 : Illustration du PPRV-PM 118
Figure 4.4 : Division unitaire d'une instance PPRV-PM 119
Figure 4.5 : Schéma général de la méthode proposée 121
Figure 4.6 : Création d'une instance auxiliaire 122
Figure 4.7 : Données pour l'affectation 125
Figure 4.8 : Application de l'algorithme du simplexe de transport 126
Figure 4.9 : Création des instances auxiliaires 127
Figure 4.10 : Schéma hybridation entre GRASP et VNS 129
Figure 4.11 : Construction d'une solution auxiliaire 131

Figure 5.1 : Limitations de modélisation classique 146
Figure 5.2 : Représentation des deux exemples 147
Figure 5.3 : Illustration du PFEC Acyclique 148
Figure 5.4 : Evolution des deux bornes en fonction des itérations 151
Figure 5.5 : Réseau dynamique 157
Figure 5.6 : Représentation d'une solution de PPRV-T en PFEC-A-Dyn 158

Liste des Schémas Algorithmiques

Algorithme 1.1 : Schéma de la méthode approchée 37
Algorithme 1.2 : Principe de l'algorithme de Clarke et Wright 38
Algorithme 1.3 : Schéma des méta-heuristiques de type recherche voisinage 45
Algorithme 1.4 : Schéma de l'algorithme génétique 48
Algorithme 1.5 : Schéma d'un algorithme de colonies de fourmis 50
Algorithme 1.6 : Décomposition de Benders 58

Algorithme 3.1 : Schéma de l'ILS 89
Algorithme 3.2 : Schéma de la VNS 90
Algorithme 3.3 : Schéma de la VND 91
Algorithme 3.4 : Schéma de l'ILS/VND 92
Algorithme 3.5 : Heuristique *Nearest Neighbour* randomisée (NN_Rand) 93
Algorithme 3.6 : Heuristique NN2 modifiée (NN2_modif) 93
Algorithme 3.7 : Schéma de construction d'une tournée géante (Construire_tournée) 94
Algorithme 3.8 : Procédure Split adaptée (Split) 100
Algorithme 3.9 : Procédure de génération d'une solution PPRV (Transformer_solution) 101
Algorithme 3.10 : Schéma de VND adapté (Recherche_locale_1) 105
Algorithme 3.11 : Schéma de la recherche locale pour les solutions PPRV (Recherche_locale_2) ... 106

Algorithme 4.1 : Algorithme du coin supérieur gauche 123
Algorithme 4.2 : Algorithme du simplexe de transport 124
Algorithme 4.3 : Schéma de GRASP/VND 129
Algorithme 4.4 : Procédure de construction (Construire_solution) 130
Algorithme 4.5 : Schéma d'un opérateur du type RAR (Opérateur_RAR) 132
Algorithme 4.6 : Schéma de la recherche locale (Recherche_locale) 133
Algorithme 4.7 : Projection d'une solution sur l'instance initiale (Projection_solution) 134

Algorithme 5.1 : Schéma de résolution de type Benders 152
Algorithme 5.2 : Schéma d'insertion pour le PFEC-A (Insertion-PFEC-A) 153
Algorithme 5.3 : Algorithme de Dijkstra adapté 155

Liste des Tableaux

Tableau 2.1 : Problèmes proches du PPRV 71
Tableau 2.2 : Comparaison des contraintes d'élimination de cycles 79
Tableau 2.3 : Résultat sur les petites instances du PPRV avec un agent ($K = 1$) 80
Tableau 2.4 : Résultat sur les petites instances du PPRV avec deux agents ($K = 2$) 81
Tableau 2.5 : Résultat sur les petites instances du PPRV avec trois agents ($K = 3$) 81

Tableau 3.1 : Résultat sur les petites instances à un agent ($K = 1$) 108
Tableau 3.2 : Résultat sur les instances moyennes à un agent ($K = 1$) 109
Tableau 3.3 : Résultat sur le groupe d'instances N100q10 à un agent ($K = 1$) 110
Tableau 3.4 : Comparaison des opérateurs d'amélioration 110
Tableau 3.5 : Résultat sur les petites instances avec deux agents ($K = 2$) 111
Tableau 3.6 : Résultat sur les petites instances avec trois agents ($K = 3$) 112

Tableau 4.1 : Problèmes proches du PPRV-PM 120
Tableau 4.2 : Taille moyenne des instances auxiliaires 135
Tableau 4.3 : Comparaison des opérateurs d'amélioration (seuil $= 3$) 135
Tableau 4.4 : Influence du seuil d'affectation sur l'opérateur *2-som-opt* 136
Tableau 4.5 : Influence du seuil d'affectation sur l'opérateur *3-som-opt* 136
Tableau 4.6 : Résultat sur les petites instances avec un agent (seuil $= 3$) 137
Tableau 4.7 : Résultats sur les différents seuils d'affectation ($K = 1$) 138
Tableau 4.8 : Comparaison du résultat sur les différents seuils ($K = 1$) 139
Tableau 4.9 : Résultats avec deux agents ($K = 2$) 140
Tableau 4.10 : Résultats avec trois agents ($K = 3$) 141
Tableau 4.11 : Comparaison des résultats pour les différents seuils ($K = 2$) 142
Tableau 4.12 : Comparaison des résultats pour les différents seuils ($K = 3$) 142

Tableau 5.1 : Résultats obtenus par l'heuristique d'insertion 160

Introduction

1. Contexte général

Depuis quelques décennies, avec l'émergence de la notion de développement durable, le respect de critères environnementaux, sociaux et économiques devient un maillon essentiel à considérer dans tous les domaines de la société, y compris dans celui du transport individuel et collectif. L'utilisation croissante des véhicules individuels depuis la seconde moitié du $20^{ème}$ siècle a eu pour conséquence une très forte augmentation des rejets de CO_2 (dioxyde de carbone) dans l'atmosphère. En France, selon un rapport de la SNCF de 2007 [Mel07], les transports participent à plus de 20 % de l'émission totale de CO_2, dont la moitié provient de l'automobile et principalement des véhicules individuels. L'augmentation du CO_2 et des autres gaz à effet serre dans l'atmosphère induit un réchauffement global incontestable [Bru87], dont l'impact sur les activités se fait déjà sentir. Pour limiter les conséquences du réchauffement, le protocole de Kyoto préconisa la mise en place de mesures destinées à réduire les émissions polluantes, avec pour objectif une réduction de 5,2 % des émissions de CO_2 jusqu'à 2012, par rapport à 1990 [PK98]. Parmi ces mesures, la question du plafonnement des émissions polluantes au niveau individuel a été introduite lors du sommet de Copenhague en décembre 2009 par un groupe de chercheurs de l'Université de Princeton. L'originalité de l'approche réside dans la proposition de fixer des objectifs par individu en fonction du niveau de richesse, partant du principe que les plus nantis sont également ceux qui polluent le plus. L'idée est de traiter de manière égale les individus ayant le même taux de pollution et ce, où qu'ils vivent.

Outre les enjeux environnementaux, l'augmentation de l'utilisation des véhicules individuels pose aussi le problème de la consommation des énergies fossiles. En particulier, le pétrole est à l'origine de 38% de la consommation mondiale d'énergie, dont 48% est utilisé directement dans les transports [Hel05]. Le pétrole est une ressource non-renouvelable : selon les analyses de l'*Association for the Study of Peak Oil* (ASPO), le pic pétrolier mondial (en termes de production) sera ou serait atteint vers 2010-2015 [Rod08, Hir05]. Ainsi, la production de pétrole va diminuer à partir de ce pic, jusqu'à épuisement. Cependant, dans le même temps, la consommation mondiale de pétrole va continuer à augmenter jusqu'en 2030 [Weo08]. Au final, bien que les prévisions divergent sensiblement selon ces analyses, le pétrole risque donc d'être épuisé au début du siècle prochain.

Sur le plan des transports, la raréfaction du pétrole va nécessiter la mise en place de stratégies alternatives destinées, soit à remplacer le pétrole, soit à réduire sa consommation globale. Actuellement, l'utilisation de sources d'énergie propres et renouvelables n'en est qu'au stade embryonnaire. Ainsi, le Brésil a mis en place une stratégie de remplacement partiel par les biocarburants (éthanol) [Dou06] qui assure de fait une autosuffisance depuis quelques années. D'autres pays semblent vouloir suivre le même processus. Cependant, la solution des biocarburants présente des défauts importants, notamment dans l'appauvrissement de la biodiversité, la réduction des surfaces agricoles vivrières, le processus-même de production et

le fait que cela ne résolve pas les problèmes de pollution et de congestion des réseaux. Les autres solutions de remplacement du pétrole (véhicules électriques, hybrides, piles à combustible, notamment) ne sont pas encore suffisamment matures. Ainsi, la recherche d'une réduction de la consommation globale est-elle une alternative valide, qui peut être à même de compenser le délai d'obtention d'une solution de remplacement, les deux approches, réduction et solution alternative, pouvant d'ailleurs coexister.

Enfin, au niveau du transport urbain, le flot actuel des véhicules a également de nombreux impacts négatifs : congestion sur les grands axes, durées de transport croissantes, pollution sonore, dégradation des édifices, coût d'entretien de l'infrastructure et limitation des disponibilités de stationnement, entre autres.

Dans cette optique, plusieurs stratégies gouvernementales ont été mises en place pour renforcer la protection de l'environnement : taxe à l'achat, taxe sur les carburants et prochainement taxe carbone. Parallèlement, des réflexions et des études sont en cours pour repenser le transport, notamment urbain, et apporter de nouvelles solutions dans le monde de la mobilité. Cela induit une évolution accompagnant un double mouvement de la société, à la fois sociétal et technologique, se servant de l'essor des nouvelles technologies pour favoriser l'émergence des nouveaux modes de transport.

Au niveau sociétal

L'évolution sociétale de la mobilité est motivée par la notion de développement durable. Elle concerne à la fois l'évolution des véhicules et l'organisation de la mobilité, afin de prendre en compte les enjeux environnementaux et sociaux. Elle se focalise sur trois préoccupations :

- la consommation d'énergies non-renouvelables, conduisant à terme à l'épuisement des ressources fossiles ;
- le réchauffement climatique, conséquence d'une émission excessive de CO_2 ;
- la congestion des trafics urbains et la saturation de l'infrastructure de transport.

Avec l'objectif de diminuer le nombre de véhicules circulant quotidiennement dans un réseau de transport et de réduire la consommation de carburant et l'émission de CO_2 par les véhicules, les innovations ont porté sur de nouveaux services et de nouvelles catégories de véhicules. Au niveau des services, les applications émergentes sont motivées par la volonté d'introduire de nouvelles formes de partage des véhicules. Elles se traduisent par l'utilisation de nouveaux types de véhicules consommant peu ou pas d'énergie fossile : vélos, véhicules électriques, véhicules hybrides, etc. Les défis à la conception de ces applications portent sur les critères suivants :

- le niveau de mutualisation induit au niveau des utilisateurs et du système ;
- la flexibilité et la réactivité dans un contexte de temps-réel ou quasi-réel ;
- la sûreté de service, tant du point de vue des usagers que de la supervision ;
- la viabilité en termes économiques de l'offre de service ;
- l'attractivité au niveau des usagers.

L'importance prise par ces enjeux sociétaux tend enfin à pousser de plus en plus les acteurs publics à s'impliquer fortement dans la mise en œuvre de ces nouveaux services de transport.

Au niveau technologique

L'évolution technologique de la mobilité se fait selon deux tendances. La première tendance correspond à l'évolution de la technologie des véhicules automobiles pour satisfaire deux exigences : d'une part en termes écologiques, pour respecter des normes de plus en plus strictes, et d'autre part en terme de robotique pour disposer de véhicules autonomes ou semi-autonomes. Sur le plan écologique, le recours à des énergies relativement propres et durables (énergies renouvelables, avec peu ou sans pollution de l'air, comme l'électricité, l'hydrogène, l'énergie solaire, etc.) est quasiment incontournable. Le développement de transports autonomes nécessite une profonde réorganisation de l'architecture des systèmes embarqués (pilotage de façon automatique, avec peu ou pas d'intervention humaine pour effectuer les missions de transports).

La seconde tendance correspond à l'utilisation croissance des STIC (Sciences et Technologies de l'Information et de la Communication) dans la gestion de la mobilité : les dispositifs de guidage et de géolocalisation (pour le suivi et la supervision des véhicules), les systèmes de communication mobile (à courte ou longue portée), les systèmes embarqués (en temps-réel ou légèrement différé), les systèmes d'information (centralisés ou répartis) et les technologies web (facilitant l'interaction de services dédiés) sont autant d'éléments permettant le développement d'offres de transport autonomes ou semi-autonomes.

Innovations en services de transport

L'accumulation technologique dans ces différents domaines au cours de la dernière décennie rend techniquement réalisable la création de nouveaux services de transport et de nouveaux modes d'organisation de la mobilité. Par ailleurs, ces services peuvent aussi à leur tour motiver la réflexion sur l'évolution des nouvelles technologies.

Cette évolution technologique et sociétale entraîne donc des innovations fortes en matière de mobilité. A ce niveau, plusieurs systèmes sont actuellement en cours d'expérimentation, certains étant déjà opérationnels et implantés au cours des dernières années dans les réseaux de transport urbain ou rural. On peut citer à ce titre :

- les dispositifs de type ***vélo-partage*** dans certaines grandes villes de France et d'Europe : *Vélo'v* à Lyon, *Bicing à* Barcelone, *Vélo'+ à* Orléans, *Vélomagg' à* Montpellier, *Vélib'* à Paris, etc…
- des systèmes de transport à la demande, suite à l'expérience fondatrice de *TELEBUS* à Berlin à partir de juin 1995. Plus de 100 minibus sont ainsi utilisés pour le transport de 25000 usagers abonnés (personnes handicapées), et 2000 demandes sont traitées par jour. [Bor97]
- le projet *Communauto*, développé en 1994 au Canada, où près de 6000 abonnés se partagent une flotte de quelque 300 véhicules ; en 2009 après 15 ans d'existence, le système est constitué d'environ 360.000 abonnés et 8000 véhicules partagés ;
- le système *Lilas* à Lille, créé en février 2007, permet la réservation de véhicules pour des abonnés utilisant le téléphone ou Internet, et offre la possibilité de trajets intermodaux. En septembre 2009, le système compte 1000 adhérents pour 25 voitures ;
- les systèmes *Modulauto* (Montpellier) et *Mobility* (Suisse). Pour ce dernier, 1700 véhicules de 8 catégories différentes sont mis à disposition de 55.500 clients répartis sur 1000 emplacements dans toute la Suisse. L'expérience montre que 80% des réservations se font par le biais de canaux virtuels, dont 65% par Internet [Mcs03] ;

- les systèmes *CityCarClub* (Grande-Bretagne), *I-GO* et *ZipCar* (Etats-Unis) et *CambioStadt* (Allemagne). Ce dernier a connu une croissance de 43 % du nombre de ses utilisateurs entre 2002 et 2005 ;
- le système *Praxitèle* est le fruit d'un consortium impliquant CGEA, Renault, Dassault Electronique, EDF, l'INRIA et l'INRETS. *Praxitèle* a été expérimenté à St-Quentin-en-Yvelines d'octobre 1997 à juin 1999, avec 50 véhicules et 5 parkings. Il est considéré comme le premier système de location en libre service utilisant les technologies de l'information et de la communication ;
- le système *Liselec* a été créé en 1999 à La Rochelle par les trois partenaires PSA Peugeot Citroën, VIA GTI et Alcatel CGA Transport. Il offre pour ses abonnés la possibilité de louer des voitures électriques disponibles en libre service ;
- Le système *Autolib'* (Paris) a été mis en route fin 2011. Il doit atteindre progressivement 2000 voitures et 7000 stations.

Un point commun entre tous ces systèmes émergents est qu'ils reposent sur le concept de *véhicule partagé*, que ce soit au niveau individuel ou collectif. Il semble que ce concept soit une tendance lourde dans la mobilité, lequel deviendra sans doute une thématique centrale lorsqu'on se penchera sur les perspectives des services de transport collectifs / individuels futurs.

2. Systèmes de véhicule partagé

D'après une synthèse de l'Agence européenne pour l'environnement (AEE), une voiture est inutilisée pendant 92 % de sa durée de vie. Elle ne transporte en moyenne que 1,2 personnes par voyage. Cette situation induit un gaspillage important, tant sur le plan économique que sur le plan environnemental. De plus, elle a des conséquences fortes sur l'utilisation de l'espace urbain. Le véhicule partagé est une réponse possible à ces préoccupations. Il consiste à créer de nouvelles formes de services en partageant l'utilisation de véhicules à la manière du covoiturage ou de l'auto-partage. On souhaite ainsi faire profiter un groupe de passagers d'un même véhicule pour un trajet (covoiturage), ou laisser un véhicule à la disposition de plusieurs utilisateurs successifs (auto-partage). Le concept de *véhicule partagé* peut aussi impliquer le recours aux véhicules consommant peu ou pas d'énergie fossile, comme par exemple le vélo et les véhicules électriques ou hybrides. Il peut également impliquer l'utilisation de véhicules autonomes. L'avantage de telles approches est qu'elles ne nécessitent pas une remise en question importante de la problématique du transport des individus, que ce soit en milieu urbain ou rural.

Nous allons nous intéresser plus particulièrement à trois catégories de systèmes associés au *partage* de véhicules de manière collective ou/et individuelle :

Le covoiturage (partage d'un véhicule)

Son principe repose sur l'utilisation d'un même véhicule par un groupe de passagers, dans le but de leur faire partager une partie du trajet du conducteur. Il permet aux utilisateurs d'économiser des dépenses liées au déplacement, et fournit ainsi une solution « simple » à l'augmentation de la pollution et à la congestion du trafic. Cette solution est considérée comme simple, car elle ne remet pas en cause fondamentalement la structure de transport.

Dans le cas général du covoiturage, il est proposé à un conducteur de partager sa voiture avec d'autres passagers. Une partie des frais (carburant, péage) est également partagée. Notons que l'autostop, aujourd'hui en perte de vitesse, correspondait aussi à cette approche. Les différences par rapport au covoiturage reposent sur le caractère non planifié du service et sur la gratuité pour le passager.

On peut distinguer deux formes de covoiturage, en fonction du type de trajet effectué :

- régulier : le covoiturage est associé à une activité récurrente des passagers, comme aller au travail, à l'école, etc. ;
- occasionnel : il est associé à des événements occasionnels, comme un voyage, une rencontre sportive, etc.

Les systèmes utilisant le covoiturage se sont développés depuis les années 80. Ils ont été initialement proposés par des associations. Puis, grâce au développement des technologies autour d'Internet, de nombreux sites web (essentiellement gratuits) ont été mis en place pour faciliter cette pratique de transport. Le covoiturage s'étend actuellement un peu partout dans le monde. Il existe plusieurs formes d'organisation des systèmes de covoiturage :

- associative : soutenue à la base par les associations (par exemple *Communauto*) ;
- individuelle : développée par un réseau de personnes ou à l'aide de portails Internet.

Le covoiturage a été fortement encouragé par les collectivités et autorités dans certains pays, surtout en Amérique du Nord, où plus de 4000 kilomètres de voies, répartis sur plus de 30 villes, sont aujourd'hui réservés aux véhicules à occupation multiple. De nombreuses aires de stationnement sont réservées aux véhicules assurant du covoiturage dans ces villes. En France, certaines collectivités réfléchissent à des formes d'incitation financière ou fiscale, sous-tendues par des mécanismes de traçabilité.

La location à libre service (partage des véhicules)

Les systèmes de transport tombant dans cette catégorie ont un caractère individuel. Dans un tel système de véhicules partagés, l'utilisation d'un véhicule est réservée à plusieurs utilisateurs, successifs ou simultanés, selon une procédure de prise en charge simplifiée par rapport à la location traditionnelle. Ces systèmes introduisent la notion de *véhicule individuel partagé*, dont l'idée principale est de permettre à des utilisateurs l'accès à des véhicules qu'ils ne possèdent pas. Les usagers ont donc accès à une flotte de véhicules dont le coût d'utilisation est associé à la distance parcourue ou à la durée de location. La conception de ces systèmes est dite « nouvelle » ; parce qu'elle diffère du système classique de location de véhicules sur plusieurs points :

- le mode d'identification : un client est identifié de façon préalable par un abonnement hebdomadaire, mensuel ou annuel. Ainsi, seuls les abonnés ont accès au service ;
- le mode de location : la réservation d'un véhicule est simplifiée et performante. Les véhicules sont mis en libre service et accessibles 24h/24 et 7j/7 ;
- le mode d'utilisation : le système offre plusieurs points de prise et de rendu des véhicules, afin de faciliter et d'optimiser l'accès au service par les clients.

Les systèmes peuvent également être classés selon le type de véhicule mis à disposition :

- Auto-partage : on met en location de courte durée des voitures ;
- Vélo-partage : on met à disposition des vélos. Certains dispositifs expérimentaux proposent même le service gratuitement (*Cyclocity* en Espagne).

Le premier système correspondant à cette définition est apparu en Suisse en 1987, puis, cette nouvelle catégorie de systèmes s'est répandue dans plusieurs pays. Cependant, elle a été essentiellement développée dans les pays de l'OCDE, en Amérique du Nord, en Europe et au Japon.

En France, la location à libre partage a été réellement développée à partir de la fin des années 1990. Dans les dernières années, plusieurs extensions ont même été mises en place dans une vingtaine de grandes villes françaises, en particulier Paris (Caisse-Commune sur l'auto-partage et *Vélib* sur le vélo-partage, voir la Figure 1) et Lyon (*Autolib'* sur l'auto-partage et *Vélo'v* sur le vélo-partage).

(a) un site de service du *Vélib* à paris (b) une camionnette de redéploiement

Figure 1 : Système de location de vélo en libre service

Un cas particulier : *les véhicules autonomes électriques*

À partir des deux formes précédentes (covoiturage et location à libre service), une nouvelle catégorie de systèmes est en cours de développement. Elle repose sur l'intégration des derniers développements au niveau des technologies de la mobilité : ces systèmes sont conçus autour de véhicules électriques et autonomes. Ils sont portés par des services de transport individuels ou semi-collectifs gérés par des systèmes d'information appropriés. Plusieurs projets, reposant sur le concept de *VIP* (*Véhicules Individuels Publics*), sont en train d'émerger : les projets *Cristal* en France, *PRT (Personal Rapid Transit)* en Grande-Bretagne et *CTS (Cybernetic Transport System)* en Italie, etc. Outre la mise en œuvre de nouvelles formes de transport, ces projets favorisent également les innovations en matière de moyens de transports en commun aptes à compléter les systèmes de transport existants.

Dans le principe, les systèmes VIP sont accessibles par le biais de stations ouvertes en tout temps. La location s'effectue simplement à l'aide d'une carte d'identification électronique. Les stations sont implantées essentiellement sur les points stratégiques du tissu urbain : aéroports, gares, quartiers commerçants, zones d'activités ou de loisir, etc... Une particularité du système VIP est d'impliquer des véhicules autonomes ou semi-autonomes. Ceci permet de décliner des services en deux modes :

- Un mode libre-service (individuel) : ce mode répond à des déplacements individuels de façon ponctuelle ou instantanée. Dans le cas ponctuel, il permet à un utilisateur de

faire venir un véhicule à un endroit précis à une date convenue (dans un périmètre de service défini) en utilisant un portail de requête distante, que ce soit via Internet ou par téléphone (on parle alors de **transport à la demande**). Dans le cas instantané, l'utilisateur se rend directement sur une station et loue directement un véhicule, sans passer par une réservation (on est alors dans la **location à libre service**). Un véhicule répond en principe à un usage individuel. On peut également considérer une utilisation simultanée par plusieurs personnes (on entre alors dans le cadre de la **mutualisation**).

- Un mode navette (collectif) : il répond à des demandes de déplacement planifiées (événement culturel ou sportif, transport de personnes handicapées, services spécifiques). Plusieurs véhicules peuvent être alors assemblés pour former un convoi. La capacité du convoi est flexible par attelage des véhicules, de façon à répondre à la demande réelle. Dans cet usage, une navette participe à une offre de type transport en commun.

Les véhicules sont « intelligents », grâce à un micro-ordinateur embarqué qui analyse toutes les informations recueillies par des capteurs (optique, sonore ou laser) et qui prend des décisions pour le micro-pilotage (par exemple, évitement d'un obstacle sur la route). La décision de routage est, quant à elle, prise par le système de gestion global. Les véhicules peuvent communiquer via un réseau mobile et se suivre automatiquement par guidage électronique. L'un des véhicules autonomes les plus connus en France est le C*yCab* développé par l'INRIA/Robosoft (Figure 2).

Figure 2 : Véhicule *CyCab (CyberCars)*

Le *CyCab* est un véhicule concept conçu en 1998 par l'INRIA. Il est destiné au futur système de transport urbain. C'est un petit véhicule électrique, de la longueur d'un vélo. Il ne possède pas de volant et est entièrement contrôlé par des systèmes d'information embarqués. Après une dizaine années de développement, le *CyCab* va enfin apparaître dans les villes en 2011. [Web1]. Parallèlement aux véhicules, un système de gestion globale des véhicules autonomes doit posséder trois blocs de fonctionnalités : la partie réservation permet un accès simple et rapide au système, la partie gestion est chargée de l'affectation des demandes aux ressources et du calcul des routes, et la partie transport est composée des véhicules autonomes pour la réalisation des trajets et la micro-gestion du transport. La Figure 3 illustre un exemple d'un tel système.

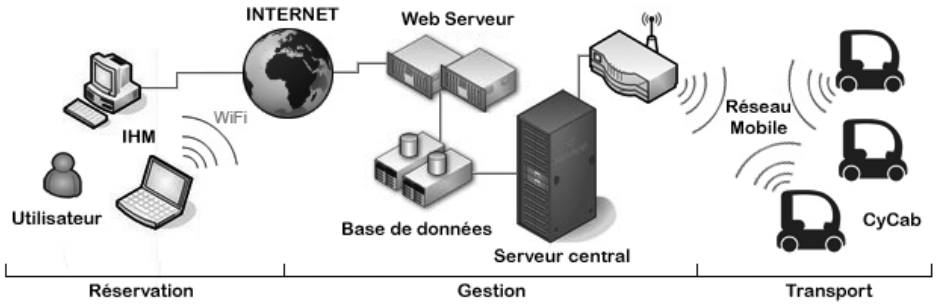


Figure 3 : Schéma d'un système CyCab

Défis lié à la gestion des systèmes de véhicule partagé

Ces systèmes sont en cours de développement. Ils doivent faire face à des problèmes dont les solutions n'ont pas encore été éprouvées par la réalisation, l'installation et la mise en application. L'enjeu scientifique est de parvenir à concevoir et introduire dans les systèmes des outils de contrôle et de gestion :

- outils de contrôle : des dispositifs électroniques ou des programmes d'aide à la gestion du trafic et de la flotte de véhicules qui permettent de suivre et de contrôler en temps-réel une flotte de véhicules (supervision, communication embarquée, localisation, autonomie partielle)

- outils gestionnaires : des outils/logiciels décisionnels qui permettent d'assurer la qualité de service et le contrôle des coûts. Il s'agit ici d'assurer l'affectation des demandes aux véhicules, le routage des véhicules et le repositionnement efficace de la ressource « véhicules » (anticipation des demandes, planification de parcours, allocation des charges de véhicules/stations), de manière à assurer à la fois la mutualisation et les acheminements rapides.

Les problèmes concernant la construction des outils gestionnaires font partie de la *Recherche Opérationnelle* (aussi appelée *aide à la décision*) et de l'*Optimisation Combinatoire* (routage, allocation de charges, etc.). Parmi ces problèmes, on s'intéresse plus spécifiquement au problème de « *Redéploiement de Convois* », qui consiste à déterminer quand et comment effectuer la redistribution des ressources, tout en respectant des critères/contextes applicatifs. Cette question concerne deux niveaux de décision : le premier niveau concerne la décision sur les conditions de la mise en œuvre du redéploiement (quand), le deuxième niveau traitant de la stratégie de redéploiement (comment). Quant aux solutions à ce problème, elles doivent être fournies de manière rapide, de façon à pouvoir s'intégrer dans un contexte de temps-réel. La recherche des solutions à ce problème est l'objet principal du présent travail de thèse.

3. Les problèmes au cœur de la thèse

3.1 Les problèmes concernant la construction des outils gestionnaires

Le problème qu'on nommera « *Problème du Redéploiement de Convois* » (PRC) peut être défini comme suit : soit un réseau de transport urbain, qui peut être constitué d'un cœur d'agglomération ou d'une zone périurbaine. Ce réseau constitue une forme de « boucle d'accès local » à un réseau de transport public à haute capacité utilisant par exemple le train,

le métro ou le tramway. On considère un opérateur (collectivité, syndicat mixte, société privée) chargé de la gestion d'une flotte de véhicules peu ou pas polluant (véhicules électriques, vélos, etc.). Un ensemble de clients abonnés peuvent emprunter ces véhicules sur des points de service, puis les utiliser et, enfin, les rendre sur des points de services (pas forcément les mêmes que les points d'emprunt). Ces véhicules sont munis d'une balise qui permet leur identification et leur localisation. Ils peuvent être déplacés par lots dans certaines conditions. C'est le cas, évidemment, des véhicules deux roues (vélos). C'est aussi celui de véhicules autonomes, lorsqu'ils sont à même d'être intégrés sans chauffeur dans des convois sans fils formés de véhicules similaires. Le PRC appartient à la classe des problèmes *NP-complet*. Il possède certaines caractéristiques des « *problèmes de tournées de véhicules* », mais il diffère de ce problème par plusieurs points particuliers qui le rendent plus difficile à traiter.

Dans le PRC, le pilotage de la flotte met en jeu le processus suivant : à un instant t donné, l'état de la répartition des véhicules sur les points de service justifie leur redéploiement, afin de mieux servir la demande. Cela correspond, le plus souvent, à un déséquilibre apparu ou allant apparaître sur la disponibilité des véhicules : soit il n'y a plus ou pas assez de véhicules pour fournir sur certains points (points de déficit), soit il y en a trop (points d'excès). Le redéploiement est réalisé par des agents qui « collectent » les véhicules sur les points en excès, constituant des lots, puis les « livrent » aux points en déficit. Ces lots sont susceptibles d'évoluer au cours du redéploiement, en particulier par l'échange des véhicules entre convois en différents points de relais. Le processus de redéploiement se termine à un instant $t+\Delta$, Δ étant la durée du processus, avant éventuellement un nouveau redéploiement.

Les difficultés qui apparaissent lors de la mise en œuvre de ce processus de pilotage peuvent être distinguées de la façon suivante, donnant lieu à autant de sous-problèmes :

- Le « *Problème du Plan de Redéploiement* » (PPR) : il correspond à la planification des opérations de redéploiement des véhicules et à l'évaluation des performances, posé du point de vue *statique*. Dans ce contexte, on ne tient pas compte de l'évolution des informations pendant la durée du redéploiement. À l'intérieur de la fenêtre temporelle $[t, t+\Delta]$, les véhicules sont redistribués par les agents « tracteurs », en forme de lots ou de convois. Un agent représente ici un chauffeur ou un véhicule tracteur, selon le type d'objet à réorganiser. Au fil des routes suivies par les agents, les lots de véhicules peuvent être assemblés ou désassemblés, de façon à réduire le coût de redéploiement. Ce coût inclue le nombre d'agents « tracteurs » mis en jeu, la longueur des trajets effectués et la fréquence des opérations de redéploiement. Par ailleurs, les objectifs du redéploiement sont associés à une analyse probabiliste de la demande d'accès au système, ce qui renvoie à une optimisation par rapport à un critère de robustesse et permet d'intégrer une notion de qualité de service.

- Le « *Problème de la Mise à jour du Plan de Redéploiement* » (PMPR) : ce sous-problème correspond, lui aussi, à la planification des opérations de redéploiement, mais il traite le point de vue *dynamique*. À un instant t' dans la fenêtre de temps $[t, t+\Delta]$, le système de supervision peut prendre connaissance de modifications de la disponibilité des véhicules sur un point de service. Il doit alors prendre en compte ces changements pour ajuster le plan de redéploiement en cours, dans un contexte en temps réel ou en léger différé. Ceci induit une modification sur le chemin des trajets d'agent et sur l'état des opérations sur des points de service. Dans certains cas, le

PMPR peut être considéré comme une séquence de problèmes « statiques » (PPR) qui possèdent des durées de redéploiement Δ relativement courtes. Ainsi, dans le cadre du transport de fret, il n'existe aucune continuation pour les objets transportés : un objet peut être livré sur un site quelconque qui en a besoin, cela signifie qu'il permet le changement de destination des objets après une nouvelle mise à jour du plan. Évidemment, ce n'est pas le cas pour le transport des personnes. De manière générale, le PMPR est un problème de décision impliquant des contraintes sur le délai de réponse et sur la prise en compte de signaux ; ainsi, peut-il être classé dans la « *Recherche Opérationnelle embarquée* ».

- Le « *Problème du Déclenchement du Redéploiement* » (PDR) : il porte sur l'identification des conditions de lancement du processus de planification. Le PDR peut être définit comme suit : à un instant donné, à partir de l'information sur la répartition géographique et sur la demande à venir en véhicules (à l'aide d'hypothèses probabilistes ou d'un modèle de simulation), on cherche à savoir s'il est nécessaire ou désirable de procéder au lancement du processus. Une telle décision repose, pour partie, sur un système de règles de décision pour s'intégrer dans une réponse temps-réel. L'identification doit se faire à partir des informations disponibles sur la répartition des disponibilités et des demandes d'accès au système. Elle doit donc s'appuyer sur un dispositif de collecte des données de géo-localisation qui permette la traçabilité du parcours des véhicules. Elle doit aussi pouvoir être validée et évaluée en termes de performances à partir d'une simulation à événement discret de l'ensemble du système.

- Le « *Problème de la Robustesse du Plan* » (PRP) : il est directement associé à la modélisation des critères de performances, dans le sens que l'un d'entre eux doit porter sur la sûreté du système, et tendre dès lors à découper les parcours et les acheminements de la façon la plus simple possible. La sûreté se définit comme la capacité à assurer le bon fonctionnement du système de transport en minimisant les opérations complexes, telles que la synchronisation et le transbordement (chargement, déchargement, croisement complexe). Moins le redéploiement nécessite de telles opérations, moins il est susceptible d'être sujet à retard ou à aléa. La modélisation d'un tel critère de sûreté est complexe, car elle débouche à priori sur des phénomènes non linéaires résultant de la résolution de problèmes d'Optimisation Combinatoire.

- Le dernier sous-problème posé est lié aux conditions périphériques et contient deux parties, la première consistant principalement à assurer le pilotage informatique (de façon autonome ou semi-autonome). A ce niveau, les difficultés sont liées à l'architecture du service et de la plate-forme informatique de suivi et pilotage du système, laquelle met notamment en jeu des techniques de communication mobile, d'acquisition et de traitement de données via des réseaux de capteurs. Quant à la deuxième partie, elle concerne les systèmes d'information et de supervision qui gèrent la flotte des véhicules et elle aborde en particulier les questions de dimensionnement des services de véhicules, d'infrastructure de suivi et de maintenance des véhicules, de pilotage informatique, de sécurisation des véhicules et de planification des activités dans le service.

Les deux sous-problèmes (PPR et PMPR) peuvent être regroupés dans le problème dit « *Problème de la Robustesse du Plan* » (PRP), qui concerne un point de modélisation particulier, dérivant de la possible utilisation de véhicules automatisés, et sera donc traité comme une extension des deux premiers sous-problèmes. Le sous-problème (PDR) relève de la décision en environnement aléatoire. Il sera traité ici de façon empirique, sachant qu'une partie de la difficulté concerne l'acquisition des données. La Figure 4 indique le schéma hiérarchique des relations entre ces problèmes :

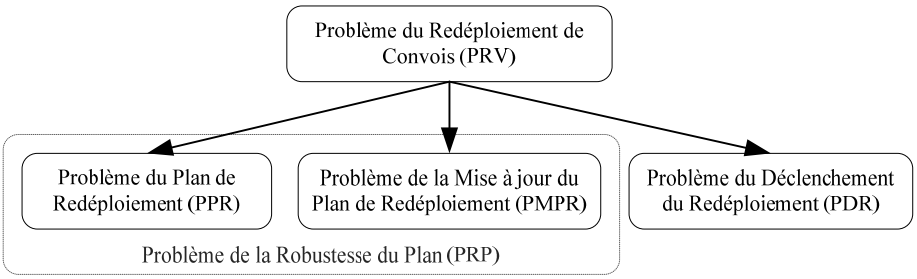


Figure 4 : Schéma hiérarchique des problèmes liés à la construction des outils

3.2 Les problèmes d'optimisation combinatoire au cœur de la thèse

Nous nous intéressons plus spécifiquement au traitement des deux premiers sous-problèmes (PPR et PMPR) présentés dans la section précédente. Le traitement de ces deux sous-problèmes relève, comme précisé plus haut, de la Recherche Opérationnelle, et plus particulièrement de l'Optimisation Combinatoire. Nous nous intéressons dans un premier temps au traitement du PPR par plusieurs approches possibles (exactes comme approchées), comme par exemple celles utilisées pour les problèmes de tournées de véhicules de type CVRP (*Capacitated Vehicle routing Problem*) ou GPDP (*General Pickup and Delivery Problem*), en considérant les diverses caractéristiques comme le passage multiple ou encore l'échanges d'objets sur un point de service durant le redéploiement. Ces caractéristiques peuvent être s'appliqués à des différentes situations.

Le PPR concerne ici donc un ensemble de problèmes de tournées et d'allocation de charge assez complexes, en combinant les différentes caractéristiques. Nous utilisons ici un quadruplet (nombre de transporteurs, charge initiale, type d'accès, transfert d'objets) pour caractériser les variantes du PPR, dont les éléments sont définis comme suit :

- nombre de transporteurs : nombre de transporteurs (pilotes par exemple) disponibles pour effectuer le redéploiement ;
- charge initiale : possibilité d'avoir une charge initiale d'un transporteur lorsqu'il part du dépôt ;
- type d'accès : nombre d'accès sur un site de services ;
- transfert d'objets : autorisation de l'échange d'objets sur un site.

Nous considérons d'abord le Problème de la Planification du Redéploiement de Véhicules partagés (PPRV) qui est décrit par (multi-transporteurs, avec charge initiale, mono-accès, sans transfert d'objets) comme la version de base du PPR. Le PPRV est cohérent à un problème 1-PDP (*One-commodity Pickup and Delivery Problem* [Hos10a]) dans la littérature. Nous avons aussi les variantes comme :

- PPRV-Sans Charge Initiale (PPRV-SCI) : (multi-transporteurs, sans charge initiale, mono-accès, sans transfert d'objets) ;

- PPRV avec Passage Multiple (PPRV-PM) : (multi-transporteurs, sans charge initiale, multi-accès, sans transfert d'objets) ;
- PPRV avec transfert (PPRV-T) : (multi-transporteurs, sans charge initiale, multi-accès, avec transfert d'objets) ;

Au point de vue dynamique, nous avons aussi le PPRV Dynamique (PPRV-D) qui relie le PPR et le PMPR. La Figure 5 illustre la hiérarchie des problèmes considérés. Le détail sur ces variantes sera présenté dans le chapitre 2.

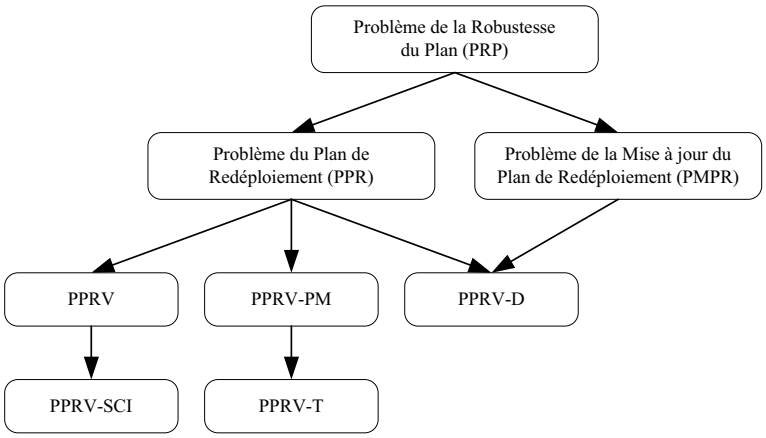


Figure 5 : Schéma hiérarchique des problèmes d'optimisation combinatoire

4. Contenu de la thèse

Dans le cadre de la thèse, on se préoccupe essentiellement du traitement des problèmes liés à la gestion d'un système de transport en libre service. On s'intéresse à une flotte de véhicules partagés pouvant être déplacés par lots. Ceci concerne principalement les véhicules deux roues (vélos) et les véhicules autonomes de type *CyCab*. Une grande partie du travail réalisé dans le cadre de la thèse a pour objectif la recherche de méthodes algorithmiques pour le problème PPR et ses variantes.

La thèse est organisée en trois parties principales :

La **première partie** est l'étude générale liée aux problèmes « proches » dans la littérature. Le **chapitre 1** correspond à l'état de l'art. Il regroupe les connaissances du domaine de l'optimisation du transport par flotte de véhicules selon deux approches. Dans un premier temps, on établit une liste des principaux problèmes connexes au sujet de recherche. On s'intéresse essentiellement aux problèmes de type « *tournées de véhicules* » (CVRP, GPDP, etc..) et aux problèmes de type multi-flots couplés dans les réseaux. Dans un second temps, on analyse les principales méthodes utilisées pour traiter ces problèmes : les méthodes exactes reposant sur une modélisation linéaire (mixte), méthodes approchées de type heuristiques et méta-heuristiques et les méthodes de recherche de flot dans un réseau.

La **deuxième partie** de la thèse correspond à la présentation des problèmes abordés et à la modélisation de la version de base des problèmes : le PPRV. Elle comporte ensuite la mise en œuvre d'une méthode approchée de type route-first / cluster-second pour le PPRV.

Le **chapitre 2** introduit la formalisation et la modélisation du *«Problème de la Planification du Redéploiement de Véhicule*» (PPRV) liée à la gestion de la flotte de véhicules partagés. Le modèle est basé sur le modèle de problème de collecte et livraison (PDP). Plus précisément, il formalise le PPRV comme un problème de collecte et livraison avec un type d'objets à transporter (1-PDP). Ce modèle est traduit sous la forme d'un programme linéaire mixte, puis est résolu à l'aide d'un solveur linéaire (Gurobi). Nous présentons des résultats sur de petites instances (de 20 à 40 nœuds), qui sont utilisés comme référence dans le chapitre suivant.

Le **chapitre 3** présente un schéma méta-heuristique pour traiter le PPRV. Cette méthode est basée sur l'hybridation de deux méta-heuristiques : ILS et VNS. Par ailleurs, deux espaces de solutions sont définis et utilisés : l'espace E1 des tournées géantes et l'espace E2 des solutions de trajectoires de transporteurs. L'opérateur de projection de E1 vers E2 est un algorithme de type SPLIT, tandis que la projection de E2 vers E1 utilise un algorithme de type concaténation. Le schéma méta-heuristique proposé est non-déterministe et repose sur une approche de type itératif. Au début de chaque itération, une tournée voisine est obtenue en appliquant une procédure de perturbation sur la tournée courante. La procédure de perturbation est dotée d'une sorte de liste tabou afin d'éviter l'exploration de zones de l'espace déjà explorées. Une tournée obtenue correspond à une séquence des clients à visiter. Elle est améliorée par les opérateurs reposant sur la relaxation de la contrainte de capacité (RCC) avant d'être séparée en plusieurs groupes (clustering) à l'aide de SPLIT. Les groupes de clients correspondent à une partition de l'ordre des clients initiaux, et chacun représente un trajet effectué par un transporteur. Ensuite, un processus de Recherche Locale est appliqué sur les groupes pour tenter d'améliorer la solution. Tous ces trajets sont concaténés pour définir un nouvel ordre de clients. Les itérations se terminent au bout d'un certain nombre d'itérations. Le résultat avec mono-véhicule est comparé à celui de la littérature, et pour le résultat avec multi-véhicules, nous le comparons avec celui obtenu dans le chapitre 2.

La **troisième partie** de la thèse correspond à la mise en œuvre des méthodes de traitement aux deux variantes : PPRV-PM et PPRV-T.

Le **chapitre 4** aborde l'extension du PPRV avec Passage Multiple (PPR-PM). Cette variante est motivée par le cas où la capacité de transporteur est susceptible d'être plus petite que les demandes. Le passage multiple sur un site de service permet d'envisager qu'une demande de collecte/livraison puisse être divisée en plusieurs parties et chaque partie est susceptible d'être traitée par un transporteur différent. Nous proposons dans ce chapitre une méthode approchée de type « *divide-first, route-second* ». On divise d'abord les demandes en plusieurs parties dans un graphe auxiliaire, à l'aide d'un modèle d'affectation. On applique ensuite un schéma heuristique sur le graphe auxiliaire, et la solution obtenue est projetée sur le graphe initial. Lors de la création du graphe auxiliaire, le modèle d'affectation cherche une affectation d'objets de l'ensemble des sites « collecte » vers des sites de « livraison » en minimisant le coût associé. Un ensemble de couples offre-demande est obtenu en découpant la solution d'affectation avec un seuil d'affectation prédéfini. Nous utilisons les opérateurs d'amélioration de type «*Remove-And-Reinsert*» (RAR) basés sur les couples. Au niveau des résultats, nous comparons le résultat du PPRV-PM avec celui du PPRV, afin de mesurer la réduction apportée par la contrainte de passage multiple.

Le **chapitre 5** présente le travail sur le PPRV avec Transfert d'objets (PPRV-T). On commence par présenter un problème auxiliaire : le Problème des Flots Entiers Couplés

(PFEC), plus précisément, la version Acyclique du PFEC (PFEC-A). Dans le PFEC-A, on utilise deux flots : un pour représenter le déplacement des transporteurs (flot noté *F*) et un autre correspondant au routage des ressources « véhicule » (flot noté *f*). Ces deux flots sont couplés par une contrainte de capacité. Nous proposons une modélisation du PPRV-T comme PFEC-A posé sur un réseau dynamique. Ce modèle intègre un mécanisme de synchronisation entre des transporteurs. Par contre, il reste statique, dans le sens où l'information n'évolue pas durant le redéploiement. Le réseau est dynamique, au sens où ses nœuds sont des copies des sommets indexés sur le temps. Dans une méthode basée sur une telle modélisation, on cherche donc à acheminer les deux flots sur un réseau dynamique en minimisant le coût total de transport. Nous proposons une heuristique dont l'idée est inspirée d'un schéma basé sur la décomposition de Benders. À chaque étape, on applique un algorithme de type primal-dual qui permet de redistribuer le flot *f* connaissant le flot *F*, tout en minimisant le coût de transport. En l'absence de base de comparaison avec des résultats de la littérature, nous comparons avec les résultats du PPRV-PM.

Chapitre 1

État de l'art : problèmes et méthodes autour des tournées de véhicules

Sommaire

1.1 Problèmes associés à la mobilité et au transport 15
1.1.1 Problèmes de tournées de type VRP 15
1.1.2 Problèmes de tournées de type PDP 17
1.1.3 Classification des problèmes de Pickup and Delivery 19
1.1.4 Principaux problèmes impliquant le « *Pickup and Delivery* » 23
1.1.5 Variantes dynamiques des problèmes de transport 26
1.2 Méthodes d'optimisation 29
1.2.1 Méthodes exactes pour les problèmes de tournées de véhicules 29
1.2.2 Méthodes approchées pour les problèmes de tournées de véhicules 37
1.3 Application des problèmes de flots sur le transport 52
1.3.1 Problèmes de « synthèse de réseaux » sur le transport 52
1.3.2 Principaux outils pour les problèmes de flots 55
1.4 Conclusion 62

1.1 Problèmes associés à la mobilité et au transport

Cette section concerne l'étude générale de problèmes concernant les tournées de véhicules. Nous commençons par présenter le *Capacitated Vehicle Routing Problem* (CVRP), qui est considéré comme le problème de base pour tous les problèmes de tournées de véhicules. Puis, nous nous concentrons sur le *General Pickup and Delivery Problem* (GPDP) et ses principales extensions.

1.1.1 Problèmes de tournées de type VRP

Les problèmes de tournées de véhicules sont connus sous le nom de « *Vehicle Routing Problem* » (VRP) [Chr79, Tot02]. Le but principal de ces problèmes est de déterminer un ensemble de *tournées* pour une flotte de véhicules afin de visiter un ensemble de clients tout en minimisant le coût total de transport. Les véhicules sont initialement localisés sur un dépôt unique et ils sont utilisés pour un même type d'opération (livraison ou collecte). On désigne par *tournée* le chemin parcouru par un véhicule. Le VRP est en fait une des principales extensions du problème du voyageur de commerce (*Travelling Salesman Problem* - TSP) [Lap86]. De ce fait, les problèmes de type VRP entrent dans la classe des problèmes NP-difficiles [Len81].

Les données d'entrée d'un problème de VRP sont définies dans un cadre statique et déterministe. Elles n'évoluent pas durant la mise en place des tournées. Dans un problème classique de ce type, on ne réalise que des opérations de livraison. Un véhicule part du dépôt

avec un chargement d'objets, puis il les livre en visitant les clients de sa tournée avant de retourner au dépôt. La charge du véhicule doit respecter sa capacité. Cette dernière caractéristique est implicite dans tous les problèmes de type VRP. Par conséquent le CVRP peut être considéré comme le problème de base de type VRP. Dans la littérature, notons que la signification du terme « VRP » varie selon les auteurs : certains auteurs le considèrent comme la classe des problèmes de tournées de véhicules (par exemple Toth *et al.* [Tot02]) alors que d'autres le considèrent comme un problème spécifique (par exemple Cordeau *et al.* [Ber07]). Nous utiliserons ici « VRP » pour désigner la classe des problèmes de tournées de véhicules.

1.1.1.1 Définition du CVRP

Le CVRP est définit comme suit : on considère un graphe orienté complet $G = (V, E)$ où $V = \{0, .., n\}$ est l'ensemble des sommets et $E = \{(i, j) \mid \forall i, j \in V, i \neq j\}$ est l'ensemble des arcs. Les sommets regroupent le dépôt (sommet 0) et les clients (sommets 1 à n). Chaque arc (i, j) représente le plus court chemin pour aller de i à j. On lui associe un coût non-négatif C_{ij} qui représente la distance (ou la durée) de parcours pour aller de i à j. Un ensemble de K véhicules (numérotés de 1 *à* K) homogènes sont initialement positionnés sur le dépôt. Chaque véhicule a une capacité Q. Chaque client possède une demande de livraison D_i. On considère que $\sum_{i=1,..,n} D_i \leq KQ$, pour garantir que la capacité totale est suffisante. Cette condition n'est cependant pas suffisante pour assurer la réalisabilité (c'est un problème de type *bin packing* en une dimension). On cherche au plus K tournées de véhicules sur le graphe qui satisfont les propriétés suivantes :

- ($c1$) une tournée commence et se termine au dépôt ;
- ($c2$) un client est visité une et une seule fois ;
- ($c3$) la charge d'un véhicule ne peut pas dépasser sa capacité ;
- ($c4$) toutes les demandes doivent être satisfaites ;
- ($c5$) la somme des coûts de transport est minimale.

1.1.1.2 Modèle linéaire du CVRP

Il existe plusieurs formulations linéaires pour le CVRP. La première formulation reprend la formulation de Dantzig *et al.* [Dan54] du TSP pour l'adapter au CVRP. Elle utilise des variables de décision $x_{ij} \in \{0, 1\}$ pour indiquer si l'arc (i, j) fait partie de la solution. Une formulation à trois indices a été développée par Fisher et Jaikumar [Fis78, Fis81]. Elle introduit les variables x_{ij}^k où $(i, j) \in E$ et $k = 1, .., K$, pour indiquer si l'arc (i, j) est utilisé par la tournée k ou non. De nombreuses autres formulations existent (voir Laporte [Lap92], Toth et Vigo [Tot02]).

Voici un modèle du CVRP avec les variables à trois indices x_{ij}^k. On définit aussi les variables redondantes $y_i^k \in \{0, 1\}$, $i \in V$ et $k = 1, .., K$, pour déterminer si le client i est visité par la tournée k. La formulation (CVRP1) a été formalisée par Toth et Vigo [Tot02] :

$$
(\text{CVRP 1})\begin{cases}
\text{Minimiser } \displaystyle\sum_{k=1}^{K}\sum_{(i,j)\in E} C_{ij}x_{ij}^{k} & & (1.1)\\
\text{s.c.} & & \\
\displaystyle\sum_{k=1}^{K} y_i^k = \begin{cases}1\\ K\end{cases} & \begin{matrix}\forall i \in V\setminus\{0\}\\ \forall i = 0\end{matrix} & (1.2)\\
\displaystyle\sum_{j\in V} x_{ij}^k = \sum_{j\in V} x_{ji}^k = y_i^k & \forall i \in V, k = 1,..,K & (1.3)\\
\displaystyle\sum_{i\in V} D_i y_i^k \le Q & \forall k = 1,..,K & (1.4)\\
\displaystyle\sum_{i\in S}\sum_{j\notin S} x_{ij}^k \ge y_h^k & \forall S \subseteq V\backslash\{0\}, h \in S, k = 1,..,K & (1.5)\\
x_{ij}^k \in \{0,1\} & \forall (i,j) \in E, k = 1,..,K & (1.6)\\
y_i^k \in \{0,1\} & \forall i \in V, k = 1,..,K & (1.7)
\end{cases}
$$

La fonction objectif (1.1) vise à minimiser le coût total de transport. Les contraintes (1.2) imposent les propriétés (*c2*) et (*c1*). Les contraintes (1.3) représentent la conservation de flot de véhicules sur un client : le nombre de véhicules entrant sur le sommet i est égal au nombre de véhicules sortant de i. Les contraintes (1.4) assurent le respect de la capacité des véhicules. Les contraintes (1.5) sont les contraintes d'élimination de cycles [Tot02]. Les variables sont définies en (1.6) et (1.7). La durée de résolution d'une telle formulation explose lorsque le nombre de clients ou de véhicules augmente. Il existe des approches plus performantes.

Les études sur le CVRP ont débutés à la fin des années cinquante [Dan59, Bal64, Chr69]. Après une cinquantaine d'années de travaux, le CVRP contient un grand nombre de variantes et d'extensions, comme par exemple l'*Open VRP* qui relaxe la propriété (*c1*). Il existe des contraintes supplémentaires portant par exemple sur la durée maximale de chaque tournée, des fenêtres de temps pour chaque client, le type de véhicules, le nombre de dépôts, la spécification des demandes. Pour plus de références sur ces problèmes, on peut citer les états de l'art de Fischer [Fis95], Laporte *et al.* [Lap95], Larsen [Lar01], Toth *et al.* [Tot02], Eksioglu *et al.* [Eks09].

1.1.2 Problèmes de tournées de type PDP

Le type PDP (pour *Pickup and Delivery Problem*) regroupe l'ensemble des problèmes de tournées particuliers dans lesquels un véhicule peut effectuer les deux opérations (livraison et collecte) dans une même tournée. Les problèmes de *Pickup et Delivery* sont connus sous la dénomination français « problèmes de ramassage et livraison ». Dans ces problèmes, on cherche à transporter des objets depuis une origine vers une destination à l'aide d'une flotte de véhicules. L'origine et la destination concernant une demande sont souvent différentes du dépôt. Concrètement, ce type de problèmes s'applique dans les domaines de la logistique, du service ambulatoire et de la robotique. Pour une étude détaillée des problèmes de *Pickup and Delivery*, on pourra se référer aux travaux de [Des95, Sav95, Rul97, Ber07, Des08, Cri10].

1.1.2.1 Définition du GPDP

Le GPDP (pour *General Pickup and Delivery Problem*) [Sav95] est la forme basique pour les problèmes de *Pickup and Delivery*. Dans le GPDP, une requête de demande spécifie une seule origine, une seule destination et une quantité d'objets à transporter. Il est défini sur un graphe orienté complet $G = (V, E)$ qui contient l'ensemble de sommets $V = \{0, .., n\}$. Le sommet 0 représente le dépôt, et les sommets numérotés de 1 à n représentent l'ensemble des clients. Chaque sommet $i \in V\backslash\{0\}$ associe à la quantité de demande D_i où $D_i > 0$ si c'est un sommet de collecte, $D_i < 0$ sinon. On considère que la somme des demandes est nulle. L'ensemble des sommets est donc séparé en deux groupes : le groupe de collecte (noté P) et le groupe de livraison (noté L). Dans le GPDP, la taille de ces deux groupes est identique ($|P| = |L|$) puisqu'un sommet origine est couplé à un sommet destination par une requête. Pour une requête m, son sommet origine se situé dans le groupe P (noté P_m) et son sommet destination est dans le groupe L (noté L_m). On a donc un nombre total de $|P|$ (ou $|L|$) requêtes dans le GPDP. L'ensemble d'arcs est noté : $E = \{(i, j\} \mid \forall i, j \in V, i \neq j\}$. Chaque arc (i, j) est associé à un coût de parcours non-négatif (pour un véhicule k) C_{ij}^k. Un ensemble de K véhicules (numéroté de *1 à K*) homogènes sont disponibles au dépôt, chacun a une capacité Q. Un véhicule peut effectuer au plus une tournée. Pour une tournée k, on note L_k l'ensemble des sommets à visiter, et S_k est la partie l'ensemble des sommets déjà visités à un instant donné. Le GPDP respecte ainsi : $\sum_{i \in L_k} D_i = 0$ et $\sum_{i \in S_k} D_i \leq Q$. Le GPDP cherche à déterminer des tournées en respectant les contraintes suivantes :

- les contraintes (de *c1* à *c5*) de la section **1.1.1.1** ;
- (*c6*) chaque couple origine-destination doit être visité par un même véhicule ;
- (*c7*) une origine doit être visitée avant la destination correspondante dans une tournée ;

1.1.2.2 Modèle linéaire du GPDP

Dans la littérature, la plupart des formulations linéaires du GPDP utilisent des variables à trois indices. Voici un modèle du GPDP basé sur une formulation du VRPTW proposée par Cordeau *et al.* [Cor02] :

$$
(GPDP\ 1)\begin{cases}
\text{Minimiser} \sum_{k=1}^{K} \sum_{(i,j)\in E} C_{ij}^k x_{ij}^k & & (2.1)\\
\text{s.c.} & & \\
\sum_{k=1}^{K} \sum_{j:(i,j)\in E} x_{ij}^k = 1 & \forall i \in P \cup L & (2.2)\\
\sum_{j:(i,j)\in E} x_{ij}^k - \sum_{j:(l,j)\in E} x_{lj}^k = 0 & \forall m = 1,..,|P|, i \in P_m, \; l \in L_m, k = 1,..,K & (2.3)\\
\sum_{j:(0,j)\in E} x_{0j}^k = 1 & \forall k = 1,..,K & (2.4)\\
\sum_{i:(i,j)\in E} x_{ij}^k - \sum_{i:(j,i)\in E} x_{ji}^k = 0 & \forall j \in P \cup L, k = 1,..,K & (2.5)\\
\sum_{i:(i,|P|+|L|+1)\in E} x_{i,|P|+|L|+1}^k = 1 & \forall k = 1,..,K & (2.6)\\
x_{ij}^k = 1 \implies \begin{cases} T_j^k \geq T_i^k + C_{ij}^k \\ Y_j^k = Y_i^k + D_j \end{cases} & \forall (i,j) \in E, k = 1,..,K & (2.7)\\
T_j^k - T_i^k \leq 0 & \forall m = 1,..,|P|, i \in P_m, \; j \in L_m, k = 1,..,K & (2.8)\\
\min\{0, D_i\} \leq Y_i^k \leq \min\{Q_k, Q_k + D_i\} & \forall i \in V, k = 1,..,K & (2.9)\\
x_{ij}^k \in \{0,1\} & \forall (i,j) \in E, k = 1,..,K & (2.10)
\end{cases}
$$

La fonction objectif (2.1) cherche à minimiser le coût total de transport. Les contraintes (2.2) et (2.3) assurent que chaque sommet/client est servi exactement une seule fois, et que le sommet d'origine et le sommet de destination sont visités par un même véhicule. Les contraintes (2.4), (2.5) et (2.6) garantissent qu'une tournée commence et se termine au dépôt. Les contraintes (2.7) contiennent deux parties : la partie d'élimination de cycles en utilisant des variables de temps et celle de conservation de la charge. Les contraintes (2.8) décrivent l'ordre de visite quand on traite une requête, son origine est visitée avant sa destination. Les contraintes de capacité sont définies en (2.9).

Par ailleurs, on pourra se référer aux formulations réalisées par Desrosiers *et al.* [Des95], Ruland *et al.* [Rul97], Gendreau *et al.* [Gen98], Cordeau *et al.* [Cor02]. Plus récemment, on peut citer les formulations du GPDP et des PDPs dans le chapitre de Barnhart *et al.* [Bar07] et dans l'état de l'art de Parragh *et al.* [Par08].

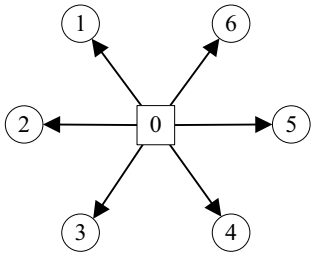
1.1.3 Classification des problèmes de Pickup and Delivery

1.1.3.1 Caractérisation des requêtes

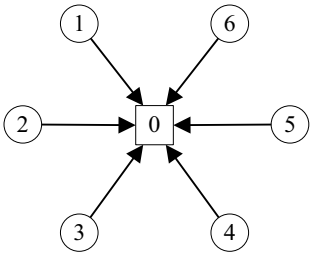
Dans les tournées de véhicules, une requête se compose structurellement de trois parties : client(s) origine, client(s) destination et quantité d'objets à transporter. Une *requête* décrit des relations entre l'offre et la demande des objets. Nous proposons de classer les requêtes en deux groupes selon la spécification des sommets origine/destination : classe désagrégée et classe agrégée.

Une requête de classe désagrégée est sous le format d'un triplet (*origine*, *destination*, *quantité*), qui signifie que le client origine souhaite envoyer une quantité d'objets au client

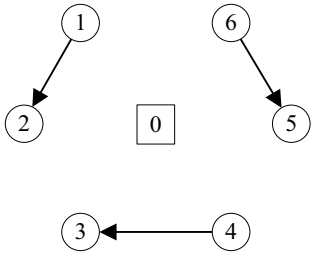
destination. Les sommets origine et destination sont spécifiques à chaque requête même s'ils peuvent correspondre au même client physique. Dans une requête (i, j, q_{ij}), on note q_{ij} la demande de l'origine et $-q_{ij}$ la demande de la destination. La Figure 1.1 illustre quatre types classiques de requête de classe désagrégée. La Figure 1.1 (a) montre un type 1-M (pour *One-to-Many*) qui est utilisé essentiellement dans le CVRP. Le dépôt joue toujours le rôle d'origine des objets dans les requêtes de ce type. La Figure 1.1 (b) indique un type M-1 (pour *Many-to-One*). Les requêtes de M-1 se trouvent dans le cas de collectes où les objets sont initialement situés sur les clients pour être collectés puis acheminés au dépôt. La Figure 1.1 (c) est le type 1-1 (pour *One-to-One*), dans laquelle le dépôt ne participe pas au transport des objets. Le GPDP utilise ce type de requête, ainsi que d'autres problèmes de *Pickup and Delivery* comme le VRPPD (*VRP with Pickup and Delivery*), le DARP (*Dial-a-Ride Problem*) et le SCP (*Stacker Crane Problem*) (voir la section 1.1.2.5.2). La Figure 1.1 (d) correspond au type 1-M-1 (pour *One-to-Many-to-One*). Ce type de requête regroupe les caractéristiques de 1-M et de M-1. Le dépôt est le centre du transport, les objets initialement sur le dépôt doivent être livrés aux clients et ceux qui sont situés initialement sur les clients doivent être collectés et acheminés au dépôt. Le type 1-M-1 est utilisé principalement par le VRPB classique (qui sera présenté dans la section 1.1.2.5.1). La différence entre un problème de VRP et un problème de *Pickup and Delivery* provient implicitement du type de requête considéré.



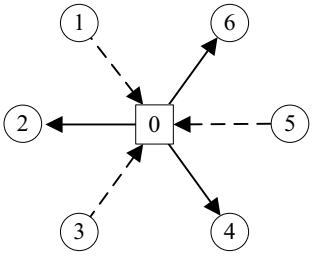
(a) type 1-M (CVRP classique)



(b) type M-1 (CVRP collecte)



(c) type 1-1 (GPDP / VRPPD)



(d) type 1-M-1 (VRPB classique)

Figure 1.1 : Types de requêtes désagrégées

Une requête de type agrégée traduit un besoin de desserte sous forme d'étoile. L'origine joue alors de rôle de fournisseur centralisé, qui se présente comme : (*origine*, (*destination* 1, *quantité* 1), (*destination* 2, *quantité* 2), …, (*destination* N, (*quantité* N))). Elle décrit une relation offre-demande associée à un même type d'objets. Il n'y a pas d'identification parmi les objets d'un type donné. Un objet ramassé sur un client de collecte (origine) peut être utilisé pour répondre à la demande de tout autre client qui en a besoin. Une requête de ce type est « agrégée » dans le sens où une origine peut avoir un ensemble de couples (*destination*, *quantité*) possible. La Figure 1.2 illustre le type M-M :

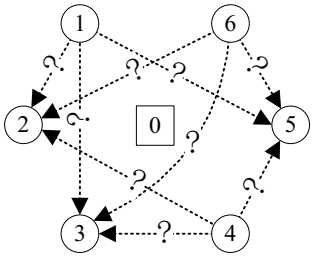


Figure 1.2 : Type de requêtes agrégées (type M-M)

1.1.3.2 Critères de classification dans les problèmes de Pickup and Delivery

La famille des problèmes de *Pickup and Delivery* contient un grand nombre d'extensions. Pour classer ces extensions, Berbeglia *et al.* [Ber07] ont proposé un schéma de classification selon trois critères : (i) la spécification de la structure des requêtes ; (ii) les opérations associées à un client ; (iii) le nombre de véhicules. Dans l'état de l'art de Parragh *et al.* [Par08], on trouve essentiellement sept critères, en plus des trois critères précédents : (iv) l'opération du client ; (v) le nombre de types d'objets ; (vi) l'ordre des visites ; (vii) le critère associé au temps. Il peut exister d'autres critères comme (viii) le « transfert » mentionné dans [Ger07, Cort10]. Tous ces critères peuvent être regroupés en 3 classes :

- les critères sur la spécification de la structure des requêtes : (i).
- les critères associés au client : (ii), (iv), (vi), (vii) et (viii).
- les contraintes quantitatives : (iii) et (v).

Un problème de type *Pickup and Delivery* est caractérisé par le choix de chacun de ces critères. En se référant aux classifications de Savelsbergh *et al.* [Sav95], Toth et Vigo [Tot02], Berbeglia *et al.* [Ber07], Parragh *et al.* [Par08], nous nous intéressons essentiellement aux critères suivants :

- (s1) le type de transport : si le transport est centralisé, les requêtes sont émises ou sont à destination du dépôt (voir le type de requête 1-M-1 de la Figure 1.1 (d)). Dans le couple origine-destination de la requête, un des sommets correspond au dépôt. Les objets sont transportés du dépôt au sommet de réception (opération de livraison) ou du sommet d'émission au dépôt (opération de collecte). Par exemple, dans le problème de livraison des boissons en bouteille, les véhicules livrent les boissons aux clients (VRPB, voir 1.1.4.1), et en même temps, font la collecte des bouteilles vides pour les ramener au dépôt. Si le transport n'est pas centralisé, les requêtes n'ont pas de lien spécifique avec le dépôt. Dans ce cas, les objets sont collectés sur un sommet client avant d'être livrés sur un autre sommet client. Les Figure 1.1 (c) et (d) montrent la différence entre les deux formes de requêtes.

- (s2) l'identification d'objet : si les objets sont identifiés et distincts, alors ils ne sont pas miscibles, comme dans les problèmes de transport de courriers ou de personnes. Dans ces problèmes, la destination d'une requête de transport est bien précisée. Les requêtes utilisées sont ainsi sous la forme de triplet. Les extensions des problèmes de *Pickup and Delivery* comme le VRPPD et le DARP sont typiquement de cette classe. Lorsque les objets ne sont pas identifiés, un objet collecté sur un client peut être utilisé pour répondre à la demande d'un autre client qui en a besoin. On peut avoir plusieurs types d'objets à transporter, chaque objet étant identique dans un type donné. Des

requêtes utilisées sous la forme d'étoile peuvent s'appliquer aux problèmes de cette classe.

- (s3) le nombre de véhicules utilisés. On peut avoir trois groupes d'extensions selon ce critère : (i) exactement un véhicule ; (ii) exactement K véhicules ; (iii) au plus K véhicules. Le premier groupe possède un caractère fort de type TSP, comme le TSPPD et le TSPB. Les deux autres groupes sont des problèmes de type VRP, comme le VRPPD.

- (s4) la capacité des véhicules : (i) capacité unitaire ; (ii) capacité limitée. Ce critère permet de différencier le VRPPD du SCP par exemple. Dans le SCP, la capacité des véhicules est unitaire.

- (s5) les contraintes temporelles : ces contraintes sont souvent liées à la mesure de la qualité de service. Elle intègre les fenêtres de temps ou la durée maximale du voyage pour un objet.

- (s6) l'opération du client : (i) on peut effectuer à la fois les deux opérations (collecte et livraison) sur un client ou (ii) on ne peut en faire qu'une.

- (s7) l'ordre des visites : il s'applique uniquement dans le cas où un client peut avoir au plus une opération (collecte ou livraison). Les clients sont regroupés en deux ensembles selon le type d'opération associée. (i) l'ordre des visites est imposé : dans une tournée de véhicules, les clients dans le premier ensemble doivent être visités avant les clients dans le second ensemble, comme le VRPB classique. (ii) l'ordre des visites est arbitraire.

- (s8) « transfert » : un sommet peut être utilisé pour stocker temporairement des objets. Ainsi l'acheminement d'un objet peut être réalisé par plusieurs véhicules.

- (s9) requête divisible ou indivisible : dans la quasi-totalité des travaux, les requêtes sont indivisibles, c'est-à-dire qu'elles doivent être traitées par un seul véhicule. Dans le cas divisible, on peut utiliser plusieurs véhicules pour traiter une requête, c'est le cas quand la demande d'un client est supérieure à la capacité du véhicule.

1.1.3.3 Classification des problèmes de Pickup and Delivery

Nous proposons ici un schéma de classification en utilisant les cinq premiers critères. La Figure 1.3 illustre une structure hiérarchique des principales catégories des problèmes de *Pickup and Delivery*. Cette classification montre ainsi les 3 groupes d'extensions du GPDP selon la structure des requêtes : 1-M-1, 1-1 et M-M. Les problèmes illustrés ici vont être présentés dans la section suivante. Outre cette classification, on pourra aussi se référer aux schémas de classification compatibles se trouvant dans [Sav95, Ber07, Par08].

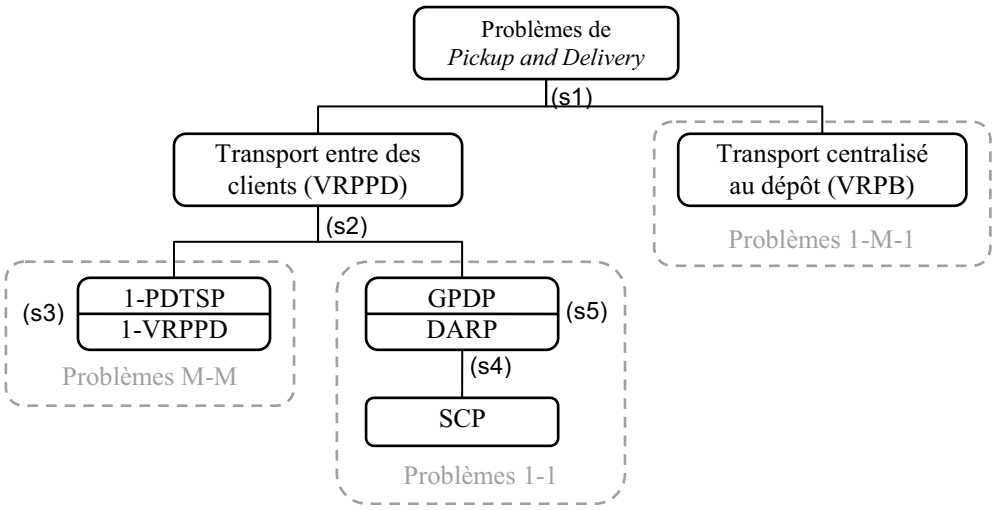


Figure 1.3 : Classification pour les problèmes de *Pickup and Delivery*

1.1.4 Principaux problèmes impliquant le « *Pickup and Delivery* »

Comme on a vu sur la figure 1.3, les problèmes impliquant le « *Pickup and Delivery* » peuvent être classés essentiellement en deux grandes catégories selon le critère (s1) : avec le transport centralisé au dépôt et avec le transport entre des clients. Le problème de tournées de véhicules avec « *Backhaul* » (VRPB) et le problème de tournées de véhicules avec livraison et collecte (VRPPD) correspondent respectivement au problème classique de chaque catégorie.

1.1.4.1 Problème de tournées de véhicules avec « *Backhaul* » (VRPB)

Le VRPB contient des requêtes du type 1-M-1. Dans le VRPB, l'ensemble de clients V sont séparés en deux sous-ensembles : ensemble L de livraison (*linehaul*) et ensemble B de collecte (*backhaul*). Les objets disponibles initialement au dépôt vont être livrés aux sommets dans L, et ceux qui sont disponibles initialement aux sommets de B doivent être collectés et acheminés au dépôt. Le dépôt participe toujours au transport des objets, que soit en émission ou en réception. Le transport des objets est donc centralisé au dépôt. Dans la version classique du VRPB, un véhicule part du dépôt avec un chargement d'objets, il fait d'abord la livraison de ces objets sur les sommets de L, puis il effectue les collectes sur les sommets de B avant de retourner au dépôt avec les objets collectés. Dans le cas général, les sommets de L possèdent une priorité plus haute que ceux de B. Ils doivent donc être traités avant les sommets de B.

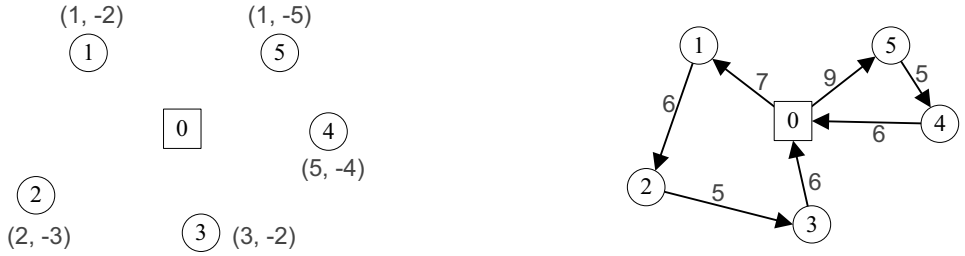
Le VRPB est décliné principalement en quatre variantes. Dans les deux premières variantes, un client n'est associé qu'à une opération. Dans les deux dernières variantes, chaque client demande une opération de collecte et une opération de livraison.

La première variante est caractérisée par le fait que les clients de l'ensemble L doivent être servis avant les clients de B. Elle est considérée comme la version classique du VRPB. Elle a été présentée initialement par Goetschalckx et Jacobs-Blecha [Goe89]. Il existe aussi des versions où les clients de B doivent servis avant ceux de L [Cas88], mais cette approche est marginale. Dans le cas à un seul véhicule, le VRPB correspond au TSPB (*TSP with Backhaul*).

La deuxième variante du VRPB ne considère pas de priorité entre les deux ensembles de clients. Une tournée est donc une séquence mixte de clients de L et de B. Anily et Mosheiov [Ani94] dénotent le problème avec un seul véhicule comme TSPDB (*TSP with Delivery and Backhauls*). Baldacci *et al.* [Bal03] l'appellent TSPDC (*TSP with mixed Deliveries and Collections*). La variante avec plusieurs véhicules est nommée MVRPB (*Mixed VRP with Backhauls*) [Sal99, Rop06], VRPBM (*VRP with Backhaul and Mixted load*) [Det02], ou encore VRPMB (*VRP with Mixed linehauls and Backhauls*) [Par08].

La troisième variante décrit des situations où des clients sont associés à la fois à l'opération de collecte et à l'opération de livraison (client $i : i \in B, i \in L, et\ B \cap L = B = L$). Un client peut être visité une seul fois avec les deux opérations réalisées en même temps. L'ensemble de ce type de clients est noté V_u. Un client peut aussi être visité deux fois : une fois pour la collecte et une autre fois pour la livraison. On note l'ensemble de ces clients V_d. Dans le cas général, un véhicule visite d'abord une partie de clients de V_d en ne faisant que l'opération de livraison, afin de vider une partie de sa charge. Ensuite, il visite des clients de V_u en faisant les deux opérations en même temps. À la fin, il passe sur la partie de clients non-visités de V_d pour l'opération de collecte. Gribkovskaia *et al.* [Gri07] nomment la version de ce problème à un véhicule SVRPPD (*Single VRP with Pickups and Deliveries*). Une extension proposée par Archetti *et al.* [Arc06], dite *split delivery*, permet de séparer le service de collecte et le service de livraison pour les clients de V_d. Ainsi le véhicule effectuant la livraison sur un site de V_d n'est pas nécessairement le même que celui qui réalise la collecte.

La dernière variante du VRPB considère que chaque client est associé aux deux opérations, mais qu'il ne peut être visité qu'une seule fois, la Figure 1.4 montre un tel problème. Le VRPSDP (*VRP with Simultaneous Delivery and Pickup*) proposé par Min [Min89] est le premier problème qui correspond à cette classe. Les problèmes comme le TSPPD (*TSP with Pickup and Delivery*) [Gen99], le VRPSPD (*VRP with Simultaneous Pickups and Deliveries*) [Ang02], le *Simultaneous* VRPPD [Nag05] et le VRPSDC (*VRP with Simultaneous Distribution and Collection*) [Del06] entrent tous dans cette variante.



(a) une instance avec les couples de demande (quantité à collecter, - quantité à livrer) (b) une solution (label sur arc = charge du véhicule)

Figure 1.4 : Problème du type 1-M-1

1.1.4.2 Problème de tournées de véhicules avec livraison et collecte (VRPPD)

Le VRPPD est un problème central dans le type *Pickup and Delivery*. Il est destiné essentiellement au transport de fret. Les requêtes de transport concernent donc des objets/colis à acheminer d'un émetteur vers un récepteur. Le VRPPD consiste à router une flotte de véhicules afin de satisfaire ces requêtes. Pour chaque requête, les sommets origine et

destination doivent être servis par le même véhicule. Ce véhicule doit donc visiter d'abord le sommet origine avant de passer plus tard sur le sommet destination. L'objectif du VRPPD est généralement de chercher à minimiser le coût du transport. Un exemple typique de l'application du VRPPD se trouve dans le service de courrier urbain. Le VRPPD contient en fait la plupart des extensions populaires du GPDP. Quand le transport est destiné aux personnes, les contraintes associées à la qualité du service (critère s5, voir la section 1.1.2.4) sont introduites puisqu'il généralise le VRPPD et le problème devient le DARP (*Dial-a-Ride Problem*).

Dans le VRPPD classique, les requêtes sont sous la forme de triplets : $(i, m+i, q_i)$ où m est le nombre de requêtes, i est le numéro du sommet origine, $m+i$ est le numéro du sommet de destination, et q_i est la quantité d'objets à transporter. On note $P = \{1, .., m\}$ l'ensemble des sommets origine et $L = \{m+1, .., 2*m\}$ l'ensemble des sommets destination. On a donc l'ensemble des sommets $V = \{0\} \cup P \cup L$. Chaque sommet i est associé à la demande D_i avec $D_i = q_i$ si $i \in P$ et $D_i = -q_{i-m}$ si $i \quad L$ (voir la Figure 1.5). Le VRPPD respecte donc les contraintes suivantes : (i) le dépôt ne peut pas être l'origine ou la destination d'une requête ; (ii) un sommet $i \in P \cup L$ ne possède qu'une opération (soit collecte, soit livraison) ; (iii) pour la requête (i, m+i, q), on a $\forall i, m+i \in P \cup L$, $D_i = q$, $D_{m+i} = -q$.

(a) une instance du problème (b) une solution (label sur arc = charge du véhicule)

Figure 1.5 : Problème du type 1-1

Comme on l'a vu sur le schéma de classification dans la section précédente, le VRPPD contient principalement deux classes d'extensions en appliquant le critère (s2). Dans la première classe, les objets à transporter sont homogènes. Les requêtes de cette classe sont du type M-M. Elles se réfèrent à des situations où les sommets de collecte et de livraison ne sont pas couplés un à un. C'est-à-dire un sommet de collecte ou de livraison peut être lié respectivement avec plusieurs sommets de livraison ou avec plusieurs sommets de collecte. Les objets collectés sur un client peuvent être utilisés pour répondre à plusieurs demandes des clients de livraison. Le transport dans cette classe est bien sûr destiné au fret. Dans la littérature, les extensions de cette classe avec un seul véhicule sont appelées CTSPPD (*Capacitated TSP with Pickups and Deliveries*) [Ani99], 1-PDTSP (*One-commodity Pickup and Delivery TSP*) [Her03] ou TSPPD (*TSP with pickup and delivery*) [Her04]. Ils correspondent en fait tous au même problème. Pour le cas multi-véhicules, Dror *et al.* [Dro98] ont étudié une application, dite PDVRP (*Pickup and Delivery VRP*). Plus récemment, Martinovic *et al.* [Mar08] ont présenté le 1-VRPPD (*Single-commodity VRP with Pickup and Delivery*). Tous les problèmes de cette classe sont en fait basés sur le PDVRP (ou le PDTSP dans le cas mono-véhicule). Le PDVRP respecte les contraintes suivantes : (i) il ne contient qu'un type d'objet à transporter ; (ii) une requête est sous le format étoile, elle possède une seule origine et plusieurs destinations. La Figure 1.6 donne un exemple de problème de cette classe.

(a) une instance du problème (b) une solution (label sur arc = charge de véhicule)

Figure 1.6 : Problème du type M-M

La deuxième classe du VRPPD contient essentiellement le GPDP et le DARP. Les requêtes sont du type 1-1. Le GPDP traite le transport des frets et le DARP traite le transport des personnes. Cette différence est généralement exprimée en terme de contraintes supplémentaires sur la qualité du service, comme la présence de fenêtres de temps et une durée maximale de voyage. Dans la littérature, le GPDP avec un seul véhicule est présenté comme le PDTSP (*Pickup and Delivery TSP*) [Kal85]. Le SCP (*Stacker Crane Problem*) est le cas particulier du GPDP où la capacité est unitaire (toutes les demandes sont égales à la capacité des véhicules). L'application principale du DARP (*Dial-a-Ride Problem*) est les services de transport à la demande pour des personnes âgées ou des personnes handicapées, dans une ville ou en périurbain. Ces services existent par exemple à Copenhague [Mad95], à Bologne [Tot96], à Berlin [Bor97], à Bruxelles [Rek06] et à Boston [Mel07]. Dans certains articles, le DARP est noté HPTP (*Handicapped Persons Transportation Problem*) [Tot96].

1.1.5 Variantes dynamiques des problèmes de transport

1.1.5.1 Problème de tournées de véhicules dynamiques

Dans la pratique, les problèmes de tournées de véhicules sont souvent « dynamiques ». Ils contiennent une partie d'informations non-déterministes qui évoluent en fonction du temps. Cela rend le problème plus complexe. Par rapport aux problèmes de tournées de véhicules classiques (qui sont souvent statiques et déterministes), certaines données d'entrée d'un problème dynamique évoluent en temps réel ou quasi-réel durant la mise en place des tournées. C'est le cas, par exemple, des nouvelles requêtes qui peuvent arriver pendant le déroulement du processus de transport. Les problèmes de tournées de véhicules dynamiques peuvent être distingués en deux types : VRP dynamique et VRP stochastique. Le type VRP stochastique contient aussi des éléments non-déterministes. Ces éléments sont connus a priori et le sens « non-déterministe » indique une probabilité de présence. Un problème de VRP dynamique est considéré comme une extension dynamique du CVRP [Psa88]. Il peut être définit comme [Lar01] :

- les contraintes (de *c1* à *c5*) de la section 1.1.1.1 ;
- (*c8*) l'information pour la planification des tournées n'est pas connue entièrement par le planificateur quand le processus de planification commence.
- (*c9*) l'information peut changer après la construction des tournées initiales.

Le VRP dynamique est souvent multi-objectif. Outre la minimisation du coût de transport (*c5*), il cherche aussi à optimiser des critères comme le taux de remplissage des véhicules ou la

satisfaction des clients. Dans un problème de VRP dynamique, l'information évolue progressivement au fil du temps. Ainsi, une solution à un problème dynamique ne peut pas être un résultat statique, mais plutôt une stratégie qui utilise les données à un instant t et qui précise quelles actions vont être effectuées, tout en respectant les critères d'optimisation.

Le caractère dynamique impose deux nouvelles classes de décisions que l'on peut prendre durant la mise en place des tournées : (i) attendre sur le sommet ou aller au prochain sommet ; (ii) accepter ou rejeter la demande. Quand un véhicule termine les opérations sur un sommet, on doit décider si le véhicule doit attendre sur ce sommet ou aller sur le sommet suivant. De même, quand le système reçoit une nouvelle requête, il faut décider si on l'accepte ou si on la rejette selon l'état actuel du système et des contraintes associées.

La Figure 1.7 donne un exemple de solution à un problème dynamique. Sur cette figure, la solution initiale est obtenue de manière statique en utilisant les données fournies à l'instant t, cette solution contient deux tournées de véhicules : tournée 1 = $(0, 1, 2, 3, 0)$ et tournée 2 = $(0, 4, 5, 0)$. À l'instant $t + \Delta$, deux nouvelles requêtes notées respectivement $(6, 0, q_6)$ et $(7, 0, q_7)$ apparaissent dans le système. À ce moment-là, les deux véhicules sont situés respectivement sur les sommets 2 et 4. Le système donne la décision « attendre » pour les deux véhicules. Après le processus d'évaluation, il libère le véhicule situé sur le sommet 4 en lui donnant la décision « aller ». Le système affecte la nouvelle requête à la tournée 1. Il effectue la mise à jour du trajet restant pour le véhicule sur le sommet 2 et lui donne enfin la commande « aller ».

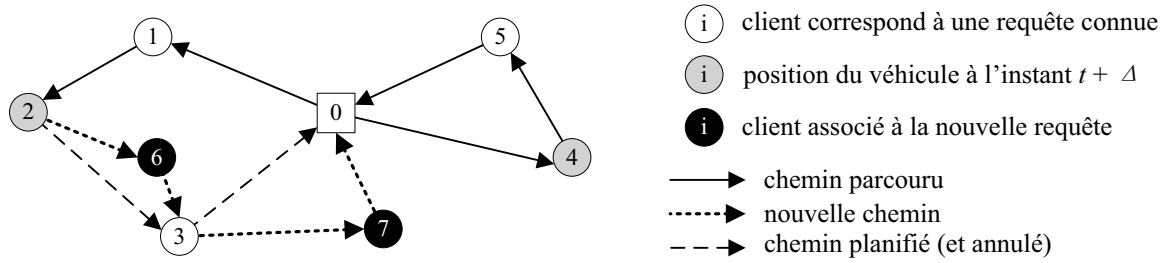


Figure 1.7 : Solution d'un problème de tournées de véhicules dynamique à l'instant $t + \Delta$

Pour examiner la performance d'un problème de VRP dynamique, outre le nombre de requêtes statiques, on doit aussi tenir compte du nombre de requêtes dynamiques. Lund et *al.* [Lun96] ont défini la notion de « degré de dynamisme » qui décrit le ratio entre le nombre de requêtes dynamiques et le nombre total de requêtes pour une instance du VRP dynamique. Puis Larsen [Lar01] a présenté une autre méthode de mesure dite « degré de dynamisme effective » qui est définie comme $D_r = 1/|R| \sum_{i \in R}(T - (l_i - t_i))/T$ où R est l'ensemble des requêtes, l'horizon de planification commence à l'instant 0 et se termine à l'instant T, l_i et t_i sont respectivement l'instant d'apparition et l'instant de service pour la requête i. La valeur t_i est égale à 0 pour toutes les requêtes i qui sont connues au départ. Ainsi $D_r \in [0, 1]$ indique le dynamisme d'une instance.

Dans la littérature, Psaraftis [Psa88] a donné une définition précise sur les problèmes de VRP dynamique. On peut aussi citer des études faites par Gendreau *et al.* [Gen98], Larsen [Lar01], Haghani et al. [Hag05].

1.1.5.2 Problèmes de *Pickup and Delivry* dynamiques

Concernant les problèmes de *Pickup and Delivry*, la plupart des études se sont concentrées sur le cas statique et peu de travail a été effectué sur le cas dynamique. Ces problèmes dynamiques sont principalement de type 1-1. Un exemple réel concernant l'application des problèmes de *Pickup and Delivry* dynamiques est le transport des personnes âgées ou des personnes handicapées en zone urbaine. La partie dynamique de ce problème vient du fait que les demandes de transport sont parfois reçues juste quelques instants avant qu'elles aient besoin d'être servies [Psa80]. Actuellement, il existe très peu d'études sur la version dynamique de type M-M ou de type 1-M-1 dans la littérature. Pour avoir d'avantages de référence, on peut consulter [Psa95, Lar01, Ghi03, Ber10].

Dans un problème de *Pickup and Delivery* dynamique de type 1-1, la requête i s'écrit sous la forme (o_i, d_i, q_i) où o_i est le sommet d'origine, d_i est le sommet de destination et q_i est la quantité d'objets à transporter. On considère P_r l'ensemble des requêtes potentielles. Le problème peut alors être par un graphe orienté complet $G = (V, E)$: l'ensemble de sommets $V = \{0\} \cup \{o_i | i \in P_r\} \cup \{d_i | i \in P_r\}$ contient le dépôt et les origines/destinations de tous les requêtes potentielles et $E = \{(i,j) | \forall i, j \in V, i \neq j\}$ est l'ensemble des arcs. Chaque arc (i,j) est associé à un coût non-négatif C_{ij} et un temps de parcours T_{ij}. Un ensemble de K véhicules (numéroté de 1 *à* K) homogènes est disponible initialement sur le dépôt. Chaque véhicule a une capacité Q. Ce problème respecte ainsi les propriétés (c1, c3, c5, c6, c7) du GPDP (voir 1.1.2.1) avec (c5) comme critère à optimiser.

Il existe essentiellement trois catégories pour les problèmes de *Pickup and Delivery* dynamiques de type 1-1. Comme mentionné dans la section 1.1.2.5.2, ces trois catégories se distinguent par les mêmes critères. Si le transport est destiné au fret et qu'un véhicule peut charger plusieurs unités d'objets, on parle de GPDP dynamique (ou VRPPD dynamique). Si un véhicule ne peut transporter qu'une unité d'objet à chaque fois, on parle de SCP dynamique. Enfin, dans le cas où le transport est destiné aux personnes, on parle de DARP dynamique.

Dans le GPDP dynamique, la plupart des objets à transporter sont des frets de petite taille comme du courrier ou des colis. Il est souvent défini sans contraintes sur la capacité. Il est généralement associé à des fenêtres de temps non serrées. Un exemple de GPDP dynamique est le service de courrier/colis express en zone urbaine. Une entreprise fournissant ce genre de service reçoit habituellement des centaines de requêtes dynamiques dans la journée. Dans la littérature, Swihart *et al.* [Swi99] ont étudié le GPDP dynamique et stochastique. Ils considèrent un seul véhicule dans le problème pour répondre à des requêtes qui sont générées aléatoirement par un processus de Poisson. Mitrovic-Minic [Mit01] a présenté un GPDP dynamique avec fenêtres de temps. Waisanen *et al.* [Wai08] ont étudié deux versions de GPDP dynamique qui ne tiennent pas compte des contraintes sur la capacité. Dans la première version, une nouvelle requête arrive avec seulement son origine et la destination sera connue lorsque le véhicule effectuera l'opération de collecte. Dans la deuxième version, l'origine et la destination sont connues pour chaque nouvelle requête. Pour d'autres références, on peut consulter [Wai08, Ber10].

Dans le SCP dynamique, un objet collecté sur un sommet origine est livré directement au sommet destination. Un véhicule ne peut transporter qu'un objet à la fois à cause de sa

capacité unitaire. Il contient les caractéristiques suivantes : (i) un seul véhicule, $K = 1$; (ii) capacité unitaire, Q =1 ; (iii) pour la requête (i, j, q), on a $i, j \in V\backslash\{0\}$, $q = 1$ et $D_i = 1$, $D_j = -1$. Le SCP est appliqué essentiellement à la gestion des transports par camion avec charge complète entre l'origine et la destination. Pour cette raison, le SCP est appelé aussi FTPDP (pour *Full Truckload Pickup and Delivery Problem*). Dans le cas dynamique, on tient compte des demandes de clients arrivées durant la mise en place des tournées. En général, le SCP dynamique concerne un horizon de planification de quelques jours, mais la moitié des demandes reçues sont pour le même jour. Dans la littérature, Powell [Pow87] a proposé un modèle de SCP dynamique. Yang et al [Yan98] ont présentés le FTPDP dynamique multi véhicules. Ils ont introduit aussi un problème similaire, dit *Real-time multivehicle truckload pickup and delivery problems*, dans lequel une requête peut être rejetée [Yan04]. Pour des référence plus récentes, on peut se référer [Pow07, Ber10].

Dans le DARP dynamique, une requête correspond au transport d'une personne individuelle ou d'un groupe de personnes d'une origine vers une destination. Comme mentionné dans la section 1.1.4.2, le DARP contient des contraintes qui assurent la qualité de service. Il destiné essentiellement au transport de personnes âgées ou handicapées dans une ville. Par rapport au DARP statique, le DARP dynamique a reçu moins d'attention dans la littérature. L'une des premières études sur le DARP dynamique a été réalisée par Psaraftis [Psa80] en utilisant un seul véhicule. Teodorovic et al [Teo00] ont présenté une version générique du DARP dynamique en utilisant la logique floue. Coslovich *et al.* [Col06] ont présenté un DARP dynamique qui permet à des personnes de demander de façon inattendue au chauffeur de monter ou de descendre du véhicule. Dans ce cas, on a besoin de prendre la décision (accepter ou rejeter) immédiatement pour une requête inattendue. On peut citer des études faites par Cordeau *et al.* [Cor07].

1.2 Méthodes d'optimisation

Selon qu'elles garantissent l'optimalité du résultat ou pas, on peut distinguer les méthodes de résolution des problèmes *NP-difficile* en deux classes : (i) les méthodes exactes ; (ii) les méthodes approchées. Une méthode exacte correspond souvent à l'expression d'algorithme d'énumération qui permet de trouver des solutions optimales pour un problème dans la classe *NP-difficile*. Elle est souvent utilisée pour résoudre des instances de petite taille (de l'ordre d'une cinquantaine de clients en pratique pour le CVRP) [Tot02b]. Quand le nombre de clients dépasse un certain seuil, le temps de calcul explose. Il augmente de manière exponentielle en fonction du nombre de clients. Il existe peu d'instances de taille moyenne (entre 100 et 200 clients), qui peuvent être résolues par des méthodes exactes dans un délai raisonnable. Pour cette raison, on utilise généralement des méthodes approchées, en particulier pour des instances de grande taille (plus de 200 clients). Une méthode approchée ne garantit pas l'optimalité de la solution obtenue, mais elle consomme un temps de calcul bien moindre. Les méthodes approchées ont beaucoup été étudiées depuis les trois dernières décennies.

1.2.1 Méthodes exactes pour les problèmes de tournées de véhicules

En général, les méthodes exactes pour le CVRP et ses extensions utilisent des algorithmes reposant sur des formulations mathématiques de ces problèmes. Ils sont efficaces sur les petites instances. Selon la classification de [Lap92], ces méthodes peuvent être distinguées en

trois groupes : (i) les méthodes de type branchement (*branch-and-bound* et *branch-and-cut*) ; (ii) la programmation dynamique ; (iii) la programmation linéaire en nombres entiers.

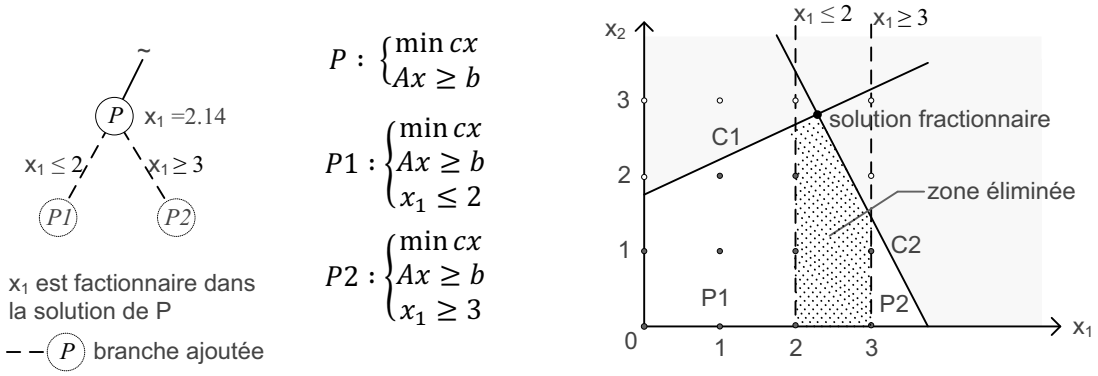
1.2.1.1 Méthodes de type branchement

Les méthodes de branchement sont aussi appelées méthodes de recherche arborescente. Elles utilisent un arbre de recherche dont chaque nœud est lié à un sous-problème. Les méthodes de branchement utilisent une stratégie de type « diviser pour régner ». À chaque itération, elles partitionnent un sous-problème sélectionné en plusieurs branches (partie « diviser »). Chaque branche correspond à un nouveau sous-problème, qui peut être résolu individuellement par des méthodes de programmation linéaire comme le simplexe (partie « régner »). Il existe essentiellement deux types de méthodes de branchement : (i) *branch-and-bound* (séparation et évaluation) ; (ii) *branch-and-cut* (coupes et branchement).

(i) Méthode *branch-and-bound*

Dans l'arbre de recherche du *branch-and-bound*, chaque arc de branchement est associé à l'ajout d'une contrainte sur la borne d'une variable. Un sous-problème lié à un nœud est en fait une relaxation linéaire du problème à laquelle on a ajouté des contraintes de borne sur certaines variables. Selon la solution du sous-problème associé à un nœud, on distingue les types de statut pour les nœuds : (i) le statut « réalisable » permet de poursuivre l'opération de séparation ; (ii) le statut « solution » correspond à une solution réalisable pour le problème initial ; (iii) le statut « rejet » correspond à un sous-problème non réalisable ou dont la solution n'est pas intéressante. La résolution du problème consiste à ajouter récursivement des branches sur des nœuds feuilles de statut « réalisables ». La solution optimale est la meilleure des solutions des nœuds feuilles réalisables.

À chaque itération, on choisit un nœud feuille de type « réalisable » pour l'opération de séparation. Cette opération consiste à ajouter deux branches sur le nœud choisi. Chaque branche correspond à une contrainte sur la variable sélectionnée (qui prend une valeur en fractionnaire). La Figure 1.8 illustre cette opération : on suppose avoir parcouru l'arbre jusqu'au P sur lequel on considère le sous problème P. La résolution de P donne x_1 fractionnaire ($x_1 = 2.14$). On associe alors la contrainte $x_1 \leq \lfloor 2.14 \rfloor = 2$ sur la première branche et la contrainte $x_1 \geq \lceil 2.14 \rceil = 3$ sur la seconde branche. Chaque branche conduit à un nouveau sous-problème (respectivement P1 et P2) qui correspond à l'insertion de la contrainte liée à la branche dans le sous-problème parent. La Figure 1.8 (b) montre un exemple de l'opération de séparation sur un problème à deux dimensions avec deux contraintes C1 et C2. On voit que l'espace des solutions est considérablement réduit dans chacun des sous-problèmes P1 et P2.



(a) sous-problèmes liés aux nœuds (b) contraintes liées au branchement

Figure 1.8 : Opération de séparation

Après l'opération de séparation, on passe dans l'opération d'évaluation de chaque nouveau nœud : si la solution d'un sous-problème est entière et qu'elle possède une valeur meilleure que la meilleure solution réalisable actuelle, alors on la garde comme la meilleure solution et le nœud est de statut « solution ». Si la solution d'un sous-problème est fractionnaire et la valeur optimale n'a pas dépassé la borne supérieure actuelle, alors ce nœud est de statut « réalisable ». Sinon, le nœud est de statut « rejet ». Les itérations se terminent quand il n'existe plus de nœuds feuille du statut « réalisable ». À l'issue de l'évaluation de tous les nœuds, la meilleure solution réalisable est la solution optimale du problème.

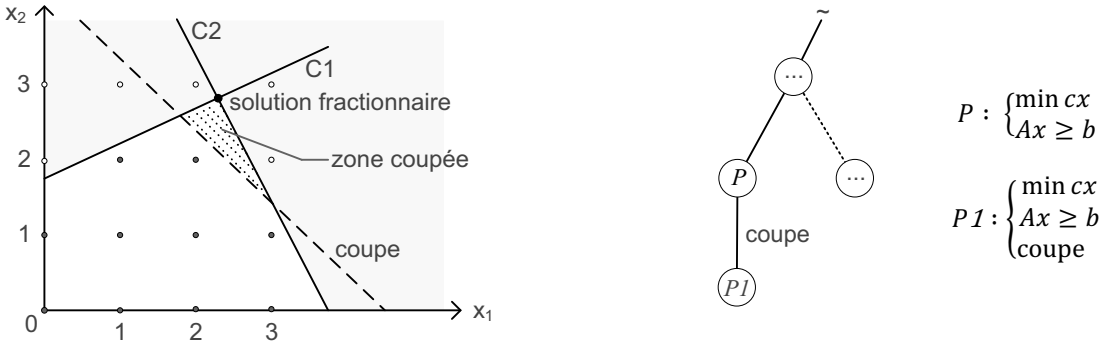
L'efficacité du *branch-and-bound* dépend de la qualité des bornes qui sont utilisées pour restreindre l'arbre de recherche. Christofides *et al.* [Chr81] utilisent la relaxation lagrangienne pour obtenir une borne inférieure. De cette manière, ils ont résolus des instances de CVRP de 25 clients. Fisher [Fis94] utilise un arbre couvrant (*spanning tree*) dans une approche dite *M-tree*. Il a réussi à résoudre une instance de 100 clients. Durant les années 90, la plupart des méthodes exactes pour le CVRP utilisent le *branch-and-bound*, combiné avec une méthode de relaxation. On pourra se référer à l'état de l'art de Laporte et Nobert [Lap87], de Toth et Vigo [Tot02, Tot02b], dans lesquels est donnée une explication plus détaillée sur la résolution exacte du CVRP par *branch-and-bound*.

Pour la résolution des problèmes de *Pickup and Delivery*, Kalantari *et al.* [Kal85] ont proposé initialement un algorithme de *branch-and-bound*. Leur algorithme est basé sur l'algorithme de Little *et al.* [Lit63] utilisé pour le TSP. Le principe de cet algorithme est d'éliminer tous les arcs qui conduiraient à la violation d'une contrainte dans chaque branche de l'arbre. Yano *et al.* [Yan87] utilisent cette méthode pour le VRPB (VPR *with Backhauls*). Ils ont généré une solution optimale avec 4 clients *linehaul* et 4 clients *backhaul* par tournée, en considérant des fenêtres de temps sur la destination, un temps maximal pour chaque trajet et des véhicules de capacité limitée. Gélinas *et al.* [Gél95] ont utilisé cette méthode pour le VRPB avec fenêtre de temps (VRPBTW). Ils ont réussi à résoudre des instances d'une centaine de clients. Toth et Vigo [Tot99] ont présenté une formulation mathématique du VRPB basée sur diverses relaxations pour la borne inférieure. Ils ont résolu la plupart des instances utilisées dans [Goe89, Tot96, Tot99] grâce à un *branch-and-bound*. Cordeau *et al.* [Cor06] ont proposé un algorithme pour le problème de *Pickup and Delivery* avec fenêtre de temps et le chargement de type *Last-In-First-Out*. Ils ont présenté trois formulations du problème et plusieurs contraintes valides. On pourra se référer à [Ber07, Cor07, Par08].

(ii) Méthode *branch-and-cut*

La méthode *branch-and-cut* est une combinaison de deux méthodes : la méthode des plans sécants (*cutting-plane*) et le *branch-and-bound*. Le *cutting-plane* améliore les sous-problèmes en ajoutant des contraintes valides afin de renforcer la relaxation. La partie *branch-and-bound* est utilisée pour trouver la solution optimale en nombres entiers. Dans la partie *cutting-plane*, les contraintes à ajouter sont dite « coupes » (*cut*). Une coupe associée à la solution fractionnaire courante doit respecter les deux critères suivants : (i) toutes les solutions réalisables entières doivent être réalisables pour la coupe ; (ii) la solution fractionnaire courante ne doit pas être réalisable pour la coupe. La Figure 1.9 (a) donne un exemple simple de la technique de *cutting-plane* : on considère que le problème est en deux dimensions ; il contient deux variables (x_1 et x_2). La solution fractionnaire se trouve sur le point d'intersection

des deux contraintes (C1 et C2). On peut alors ajouter une coupe (ligne en pointillé sur la figure) qui sépare la solution fractionnaire, ce qui réduit l'espace des solutions.



(a) ajoute d'une contrainte de coupe (b) sous-problèmes liée aux nœuds

Figure 1.9 : Contrainte de coupe

À chaque itération, on réalise l'opération coupe sur les nouveaux nœuds feuilles de type «réalisable» après les opérations de séparation et d'évaluation. C'est la différence avec les méthodes de *branch-and-bound*. Comme on a vu sur la Figure 1.9 (b) : le nœud *P* est créé par le *branch-and-bound*. On réalise sur ce nœud *P* une opération de coupe qui génère une nouvelle branche. Cette branche contient un arc lié à une contrainte de coupe et un nœud associé à un sous-problème. Ce sous-problème (*P1*) est donc le sous-problème (*P*) augmenté d'une contrainte de coupe. On refait une nouvelle fois l'opération d'évaluation pour le(s) nouveau(x) nœud(s) généré(s). L'algorithme de *branch-and-cut* s'arrête quand l'arbre de recherche ne contient plus de nœuds feuille de type « réalisable ».

Cette méthode a été utilisée initialement par Chvatal [Chv73] pour le TSP. Elle est introduite par Laporte *et al.* [Lap85] pour le CVRP. Plusieurs types de contraintes valides ont été introduits par Hill [Hil95], qui a résolu des instances de CVRP d'une centaine de clients. Naddef et Rinaldi [Nad00] ont fait une étude poussée sur cette méthode. Plus récemment, on pourra se référer à l'algorithme développé par Lysgaard *et al.* [Lys04] pour le CVRP.

Cette méthode est aussi utilisée pour la résolution des problèmes de *Pickup and Delivery* (PDP). Ruland et Rodin [Rul97] ont présenté un algorithme de *branch-and-cut* pour le PDP avec un seul véhicule. Ils appliquent une génération de coupe après chaque opération de séparation, ainsi qu'une procédure de construction de route qui génère la première borne supérieure. Cordeau [Cor06] a proposé un algorithme de ce type basé sur une formulation à trois indices pour le DARP (*Dial-a-Ride Problem*). La plus grande instance résolue contient 36 clients. Ropke *et al.* [Rop09] ont étudié un algorithme de *branch-and-cut* en utilisant deux formulations à deux indices pour le DARP. Ils ont résolu une instance d'une centaine de clients. Le *branch-and-cut* est actuellement considéré comme la meilleure méthode exacte pour les PDP de type M-M [Par08]. Hernandez-Pérez et Salazar-Gonzalez [Her03, Her04] ont introduit un algorithme de *branch-and-cut* qui utilise une heuristique constructive pour obtenir une solution faisable du sous-problème lié au nœud courant. Ils ont réussi à résoudre des instances de PDTSP avec 75 clients. Pour davantage de détails sur l'application de ces méthodes aux PDP, on pourra se référer à [Cor06, Cor07, Par08].

1.2.1.2 Programmation dynamique

La programmation dynamique est une stratégie de résolution ascendante, qui a été introduite par Bellman dans les années 50. Elle cherche la solution optimale d'un problème à partir des solutions des sous-problèmes. Chaque sous-problème est associé à un état dans une étape. La programmation est similaire aux méthodes de type branchement dans le sens où une solution du problème courant dépend des solutions des sous-problèmes précédents. Les différences sont : (i) le type de résolution est ascendant alors qu'il est descendant pour les méthodes de type branchement ; (ii) la programmation dynamique permet aux sous-problèmes de se superposer. La Figure 1.10 montre les différences dans la résolution des sous-problèmes entre ces deux types de méthodes.

(a) méthodes branchement (b) programmation dynamique

Figure 1.10 : Schémas arborescents

La programmation dynamique cherche à minimiser le coût des trajectoires dans un espace d'états. Elle s'applique à des problèmes dont l'évolution est caractérisée par des états successifs $s_0, s_1, .., s_n$ aux étapes 0,1, .., N. Chaque étape est associée à une transition entre deux états. La transition de l'état courant (s_i) à l'état suivant (s_j) se fait selon la décision (x_{ij}) qu'on va prendre. Un coût $c_k(s_i, s_j)$ est associé au changement d'état de s_i à s_j pour la décision x_{ij}. La décision est liée à la solution optimale $f^*_{k+1}(s_0, s_j)$ de l'état s_j, qui est égale à $\min_{p \in S} \{f_k(s_0, p) + c_k(p, s_j)\}$ où S est l'ensemble des états qui peuvent avoir une transition à l'état s_j à l'étape $k+1$, et $f_k(s_0, s_i)$ est la solution optimale du sous-problème lié à l'état i à l'étape k. Les transitions des états successifs avec les décisions associées entre les étapes initiales et finales forment des trajectoires. La Figure 1.11 montre la structure basique pour la programmation dynamique.

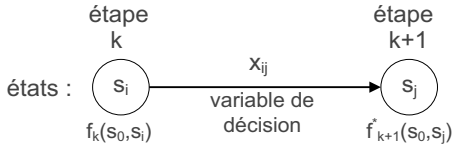
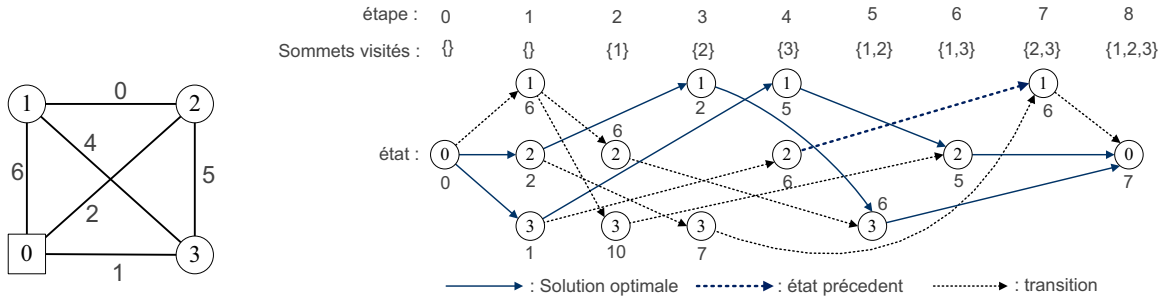


Figure 1.11 : Structure basique de la programmation dynamique

Dans la programmation dynamique, la modélisation du problème contient quatre éléments essentiels : (i) les états ; (ii) les transitions ; (iii) les décisions ; (iv) les coûts. Pour un problème de tournées de véhicules, un état est un couple (F, i) où F est l'ensemble des sommets clients déjà visités, i est le sommet à visiter. Une transition correspond à la visite d'un sommet. La décision à prendre consiste à déterminer les états précédents pour tous les états de la transition courante. Le coût de transition est le coût de transport pour aller d'un sommet à un autre. La résolution est faite de façon à prendre des décisions successives de la transition initiale jusqu'à la transition finale. La Figure 1.12 illustre un exemple simple sur la

résolution d'un problème de tournées de véhicules : la figure à gauche est une instance du CVRP sur un graphe non orienté ; la figure à droite montre la résolution de cette instance avec un seul véhicule. Il existe deux solutions optimales symétriques $(0, 2, 1, 3, 0)$ et $(0, 3, 1, 2, 0)$ pour cette instance, de valeur 7.



(a) graphe (b) résolution en programmation dynamique

Figure 1.12 : Résolution d'un problème de tournées

La programmation dynamique a été appliquée initialement par Eilon *et al.* [Eil71] pour le CVRP. L'efficacité de cette méthode est liée au nombre d'états. L'utilisation d'une procédure de relaxation qui permet de réduire le nombre d'états est importante pour développer une telle méthode. Christofides *et al.* [Chr81] introduit une procédure dit *State-space relaxation* qui vise une réduction importante des états, afin d'obtenir des bornes inférieures valides. Il utilise une nouvelle formulation pour calculer la solution optimale $f^*_{k+1}(s_0, s_j)$ de l'état s_j à la transition $k+1$. Dans cette nouvelle formulation, il remplace un état s par $g(s)$ où g est une fonction qui donne une réflexion de l'espace des solutions de l'ancienne formulation vers un nouvel espace de solutions, afin de réduire le nombre des états cardinaux dans le système. Cette formulation peut s'écrire comme $f_{k+1}\left(g(s_0),\ g(s_j)\right) = min_{p\in\Delta}\{f_k(g(s_0),\ p),\ c^*_k(p,\ g(s_j))\}$ avec $c^*_k(p,\ g(s_j))$ égale à $min_{x,y\in S}\{c_k(x,y) \mid g(x) = p\ et\ g(y) = g(s_i)\}$, Δ est l'ensemble d'états qui peuvent avoir une transition vers l'état $g(s_j)$ à la transition $k+1$. Christofides [Chr85] montre qu'il peut ainsi résoudre des instances du CVRP de 50 clients avec cette procédure. Pour plus de détails sur cette méthode, on peut se référer aux travaux de [Lap92, Sec00, Nov09].

La programmation dynamique a été utilisée par Psaraftis [Psa80] pour le DARP, il a étudié le cas de « demande immédiate » dans lequel une liste de demandes doit être servie aussitôt que possible. Dans son algorithme, il considère que la fenêtre de temps n'est pas spécifiée par les utilisateurs. Cet algorithme possède une complexité de O ($n^2 3^n$) et il ne peut résoudre que des petites instances. Psaraftis [Psa83b] a présenté ensuite une extension de cet algorithme qui traite les fenêtres de temps pour les points origine et destination d'un client. Cette extension possède la même complexité algorithmique. Desrosiers et al [Des86] ont proposé un algorithme de programmation dynamique pour le VRPPD mono-véhicule, en tenant compte de fenêtres de temps. Chaque état est associé à un temps de passage et à une solution courante. Cet algorithme permet d'éliminer efficacement des états non-réalisables/dominés. Il est capable de résoudre des instances avec un maximum de 40 clients. Par ailleurs, la programmation dynamique est souvent intégrée dans des méthodes approchées pour trouver des solutions exactes dans les sous-problèmes. C'est le cas des heuristiques proposées dans [Psa95, Car02]. On la retrouve aussi souvent dans la résolution des sous-problèmes dans les méthodes de génération de colonnes.

1.2.1.3 Programmation linéaire en nombres entiers

(i) Formulation de type *set-partitioning*

La formulation de type *set-partitioning* est proposée initialement par Balinski et Quandt [Bal64] pour le CVRP. On considère $T = \{1, 2, .., K\}$ l'ensemble des tournées de véhicules réalisables. Pour chaque tournée *k*, on associe une variable binaire x_k qui vaut 1 si la tournée k se trouve dans la solution optimale et 0 sinon. Soit C_k le coût de la tournée k et A_{ik} l'indicateur de présence du sommet i dans la tournée k. La formulation ($CVRP$ 2) montre un problème de tournées véhicules formulé comme un problème de *set-partitionning* :

$$(CVRP\ 2)\begin{cases} \text{Minimiser } \sum_{k\in T} C_k x_k & & (2.1) \\ \text{s.c.} & & \\ \sum_{k\in T} A_{ik} X_k = 1 & \forall i \in V \setminus \{0\} & (2.2) \\ \sum_{k\in T} x_k \leq K & & (2.3) \\ x_k \in \{0,1\} & \forall k \in T & (2.4) \end{cases}$$

La contrainte (2.2) impose qu'un sommet client est visité une fois et une seule. La contrainte (2.3) limite le nombre de véhicules disponibles. Dans cette formulation, le nombre de variables de décision est lié au nombre de tournées possibles, qui est souvent exponentiel. Cela rend le problème difficile à résoudre de manière directe. Notons qu'une tournée k correspond à un sous-ensemble de sommets clients S_k qui respecte la contrainte de capacité : $\sum_{j\in S_k} D_j \leq Q$ où D_j est la demande liée au sommet j et Q est la capacité de véhicule. La valeur C_k est obtenue en résolvant un sous-problème de TSP sur S_k.

Dans la littérature, Toregas *et al.* [Tor72] ont introduit une relaxation linéaire de ($CVRP$ 2), en relaxant la contrainte (2.3). Ils définissent $C_k = 1$ pour tous les $k \in T$, afin de minimiser le nombre de véhicules. Cette relaxation donne souvent une solution entière si le nombre de variables est relativement petit. Orloff [Orl76] introduit une contrainte valide basée sur la technique de *cutting plane* à la relaxation de [Tor72]. Elle est très efficace pour le cas où la solution est fractionnaire. Agarwal *et al.* [Aga89] ont proposé un algorithme pour le VPR basé sur la formulation de *set-partitioning*. On peut ainsi trouver plusieurs applications de cette technique pour le CVRP dans [Des95]. Hadjcostantinou *et al.* [Had95] ont utilisé le dual de la relaxation linéaire du modèle ($CVRP$ 2) pour obtenir une borne inférieure dans une méthode heuristique. Mingozzi *et al.* [Min94] ont développé une procédure de relaxation linéaire basée sur la dualité de cette formulation, qui est utilisée avec la méthode *branch-and-bound*. Ils ont réussi à résoudre des instances de 50 clients avec cette combinaison. Pour les PDP, la formulation de *set-partitioning* est souvent utilisée avec une stratégie de génération de colonnes, comme dans [Des88, Gél95, Ioa95, Xu03].

(ii) Génération de colonnes

La méthode de génération de colonnes est souvent utilisée pour résoudre les formulations linéaires de grande taille comme celle de *set-partitioning*. Le principe de la méthode est de résoudre à chaque itération un problème réduit, dit « Problème Principal Restreint » (PPR),

qui utilise seulement un sous-ensemble des variables. Après chaque résolution, on introduit une colonne (une nouvelle variable) qui n'appartient pas au sous-ensemble courant, afin d'améliorer la qualité de la solution. Par exemple, on cherche à résoudre le ($CVRP3$) qui est une formulation de *set-partitionning* pour le problème de tournées de véhicules. Le ($PPR3$) est le problème réduit du ($CVRP3$). Il travaille avec un sous-ensemble de colonnes $S \subset T$ tel que $|S|$ doit être supérieur au nombre de contraintes.

$$(CVRP\ 3)\begin{cases} \text{Minimiser} \sum_{j\in T} C_j x_j & \\ \text{s.c.} & \\ \sum_{j\in T} A_{ij} x_j = 1 & \forall i \in V\backslash\{0\} \\ x_j \in \{0,1\} & \forall j \in T \end{cases} \qquad (PPR\ 3)\begin{cases} \text{Minimiser} \sum_{j\in S} C_j x_j & \\ \text{s.c.} & \\ \sum_{j\in S} A_{ij} x_j = 1 & \forall i \in V\backslash\{0\} \\ x_j \geq 0 & \forall j \in S \end{cases}$$

À chaque itération, soit x^* et * respectivement la solution primale et la solution duale du ($PPR3$) courant. Dans un sous-problème (SP), on cherche la colonne i qui possède le plus petit coût réduit C_i^* où $C_i^* = \min_{j\in T/S}\{C_j - \lambda^* E_j\}$, E_j est la colonne j de la matrice A du ($CVRP3$) et C_j est le coefficient de la variable x_j dans la fonction objectif du ($CVRP3$). Si la valeur de C_i^* est positive ou nulle, la solution du ($PPR3$) courant est alors la solution optimale pour le ($CVRP3$). Sinon, on introduit la colonne j dans ($PPR3$) et on passe à l'itération suivante. La Figure 1.13 montre le processus général de résolution.

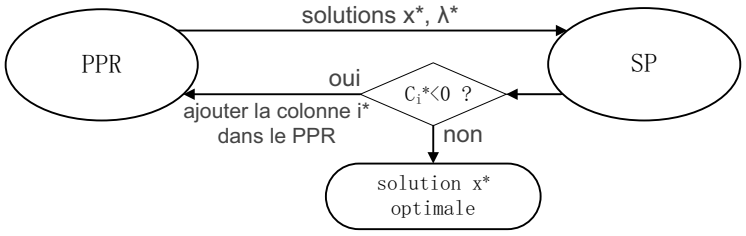


Figure 1.13 : Processus général de la génération de colonne

La génération de colonnes a été introduite initialement par Dantzig et Wolfe [Dan60]. Les livres de Wolsey [Wol98], Desaulniers *et al.* [Des05a] donnent une explication précise sur cette méthode. Elle a été appliquée en particulier par Foster *et al.* [Fos76], Orloff [Orl76], Agarwal *et al.* [Aga89], Desrochers *et al.* [Des91], Chabrier [Cha06], Jin *et al.* [Jin08] pour le CVRP. Pour assurer que la solution soit en nombres entiers pour le CVRP, la génération de colonnes doit être utilisée en conjonction avec un algorithme de *branch-and-bound*.

Dumas *et al.* [Dum91] ont proposé une génération de colonnes pour le VRPPD avec fenêtres de temps. Ils utilisent la programmation dynamique pour chercher la colonne qui possède le plus petit coût réduit. Ils ont résolu efficacement des instances inférieures à 50 clients. Gélinas *et al.* [Gél95] ont proposé un algorithme de *branch-and-bound* basé sur la génération de colonnes pour le VRPPD avec fenêtres de temps. Ils utilisent une formulation de *set-partitioning* pour le VRPPD. Dans l'opération de séparation, ils séparent des variables de ressources comme la capacité ou le temps, à la place de la variable de flot d'objets sur un arc. Savelsbergh *et al.* [Sav98] ont développé un algorithme de génération de colonnes similaire à celui qui se trouve dans [Dum91]. Cet algorithme est embarqué dans le système d'aide à la décision d'une entreprise de distribution, capable de traiter des demandes apparues dynamiquement.

La génération de colonnes est aussi utilisée pour les PDP. Elle donne des solutions optimales pour des sous-problèmes. Cullen *et al.* [Cull81] l'ont utilisé pour résoudre des sous-problèmes de *cluster* et de *routing* dans une heuristique pour le DARP. Desrosiers *et al.* [Des88], Ioachim *et al.* [Ioa95] ont utilisé la génération de colonnes pour traiter le sous-problème de génération des *mini-clusters* dans une méthode heuristique pour le DARP. Ils ont utilisé d'abord une formulation de *set-partitioning* pour le sous-problème *mini-clusters*, puis l'ont résolu avec la génération de colonnes. Xu *et al.* [Xu03] ont proposé un algorithme heuristique basé sur la génération de colonnes pour le VRPPD. Ils résolvent d'abord le problème maître qui utilise la formulation de *set-partitioning*. Puis, les sous-problèmes restant sont traités par l'heuristique. Cet algorithme est capable de résoudre des instances de 200 clients. Pour plus de références sur la génération de colonnes, on peut citer les états de l'art de [Ber07, Cor07, Par08].

1.2.2 Méthodes approchées pour les problèmes de tournées de véhicules

Nous distinguons les méthodes approchées en deux classes : (i) heuristique ; (ii) méta-heuristique. Une heuristique correspond à un algorithme dédié à un problème NP-difficile, capable de trouver efficacement une solution réalisable. La solution est rarement optimale. Une méta-heuristique est associée à une méthode approchée générique, indépendant du problème considéré.

La résolution d'un problème de manière approchée comporte souvent deux phases (voir l'algorithme 1.1) : (i) phase initialisation ; (ii) phase d'amélioration itérative. La phase (i) a pour l'objectif de générer une/des solution(s) initiale(s) qui sera/seront la/les solution(s) courante(s) pour la phase (ii). Puis dans la phase (ii), la/les solution(s) courante(s) est/sont améliorée(s) itérativement.

Algorithme 1.1 : Schéma de la méthode approchée

(i) phase d'initialisation
Générer une solution S_0 (ou un ensemble de solutions Ω_0)
Solution(s) courante(s) $S' \leftarrow S_0$ (ou Ω_0)

(ii) phase d'amélioration itérative
Tant que le(s) critère(s) d'arrête non satisfait(s) **faire**
| Générer une solution (ou des nouvelles solutions) à partir de S'
| Mettre à jour S'
Fin tant que
Retourner S'

1.2.2.1 Heuristiques

Les heuristiques dédiées aux problèmes de tournées de véhicules sont souvent dérivées de celles définies pour le TSP. Il existe des méthodes d'insertion ou des procédures d'améliorations, qui peuvent s'appliquer directement aux problèmes de tournées sans aucune modification. Selon les classifications dans [Lap02] et [Cor07], les heuristiques pour les problèmes de tournées peuvent être classées dans trois groupes : (a) heuristiques de construction ; (b) heuristiques en deux phases ; (c) heuristiques d'amélioration.

(a) Heuristiques de construction

Dans les problèmes de tournées de véhicules, les heuristiques de construction commencent par une solution vide et construisent des tournées en insérant progressivement un ou plusieurs clients dans la solution en privilégiant les insertions de coût minimal. Le processus d'insertion se termine quand tous les clients sont insérés dans la solution. Le principe algorithmique d'une telle méthode est caractérisé essentiellement par trois critères : (1) critère d'initialisation ; (2) critère de sélection des clients ; (3) critère de construction. Il existe deux manières de sélectionner les clients (critère 2) qui mènent à deux types de méthodes : méthode d'économies et méthode d'insertion.

- Méthode d'économies

La méthode d'économies est basée sur la notion de *saving* (économie). L'idée principale de cette notion consiste à fusionner deux tournées en une même tournée en maximisant le coût de réduction apportée par cette opération. On suppose que i est le dernier client d'une tournée et j est le premier client d'une autre tournée. Le coût de réduction associé à la fusion des deux tournée est défini comme : $S_{ij} = C_{i0} + C_{0j} - C_{ij}$ où C_{ij} est le coût de parcours pour aller de i à j. Si la valeur de S_{ij} est positive, alors la fusion des deux tournées est profitable. Dans l'algorithme de Clarke et Wright [Cla64], le coût de réduction est calculé pour tous les couples (i, j) possibles. Ils sont triés dans l'ordre décroissant. L'algorithme commence par construire une solution initiale dans laquelle une tournée est réalisée pour chaque client. À chaque itération, on fusionne les deux tournées correspondant à un couple (i, j) tel que S_{ij} est maximal et positif. Dans le cas où on minimise le nombre de tournées, un coût de réduction négatif peut être accepté. L'algorithme 1.2 montre la méthode de Clarke et Wright.

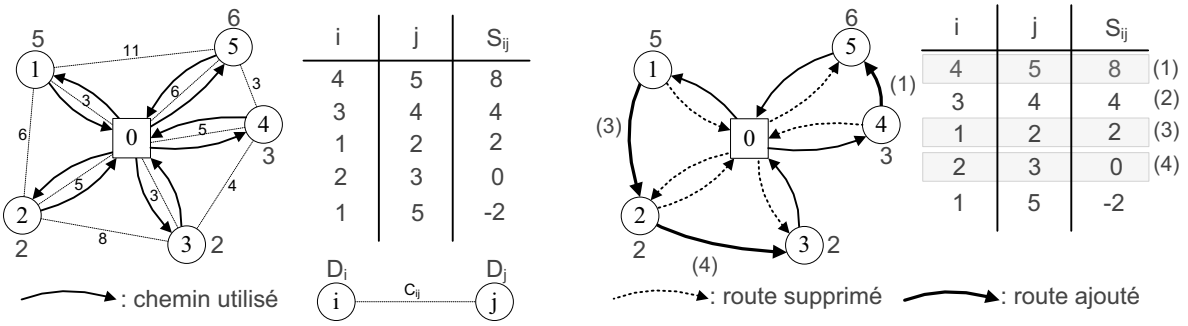
Algorithme 1.2 : Principe de l'algorithme de Clarke et Wright

(i) phase d'initialisation
Initialiser la liste des économies $L \leftarrow \{S_{ij} \mid (i,j) \quad E\}$;
Trier L dans l'ordre décroissant ;
Initialiser l'ensemble des $|V\backslash\{0\}|$ tournées $T \leftarrow \{T_i \mid i \quad V\backslash\{0\}\}$;

(ii) phase d'amélioration itérative
$S_{ij} \leftarrow$ première élément dans L ;
Tant que (L non vide) **et** ($S_{ij} > 0$) **faire**
| **Si** la tournée T_i **et** la tournée T_j peuvent être fusionnées **alors**
| | $T \leftarrow T \setminus \{T_i, T_j\}$;
| | $T \leftarrow T$ U Fusion(T_i, T_j) ;
| **Fin si**
| $L \leftarrow L \setminus \{S_{ij}\}$;
| $S_{ij} \leftarrow$ premier élément dans L ;
Fin Tant que
Retourner T ;

La Figure 1.14 donne un exemple de cette méthode. Le graphe et la solution initiale sont présentés Figure 1.14 (a). On considère que la capacité des véhicules Q est égale à 10. Le chiffre sur les arcs pointillés reliant 2 sommets i et j représente le coût C_{ij} de déplacement du véhicule entre i et j. Le chiffre indiqué près d'un sommet i est la quantité D_i demandé par le client i. La solution obtenue est illustrée dans la figure à droite. Dans l'itération (2), la fusion

liée à S_{34} est rejetée à cause de la capacité de véhicule. La solution présentée cette figure contient deux tournées : (0, 4, 5, 0) et (0, 1, 2, 3, 0).



(a) solution initiale et graphe (b) solution obtenue

Figure 1.14 : Illustration de la méthode de Clarke et Wright

La méthode d'économies est initialement proposée par Clarke et Wright [Cla64]. Elle peut être implémentée de façon parallèle ou séquentielle. Golden *et al.* [Gol77], Nelson *et al.* [Nel85], Paessens [Pae88] ont apporté des améliorations en utilisant des structures de données plus adaptées. Desrochers *et al.* [Des89], Altinkemer *et al.* [Alt91], Wark *et al.* [War94] ont proposé d'utiliser des algorithmes de *matching* pour déterminer les fusions des tournées. Les résultats obtenus sont meilleurs, mais ils sont plus couteux en temps de calcul.

- Méthode d'insertion

Le principe d'une méthode d'insertion consiste à insérer des clients dans des tournées existantes, en satisfaisant un critère sur l'augmentation du coût d'insertion ou sur la distance entre le dépôt et le client à insérer. Si on souhaite insérer le client x entre les clients i et j adjacents dans une tournée existante, le coût (i,x,j) est défini comme : $(i,x,j) = C_{ix} + C_{xj} - \quad C_{ij}$ où C_{ij} est la distance pour aller du sommet i au sommet j, et est un paramètre de contrôle. Dans certains cas, on tient compte aussi de la distance entre le client x et le dépôt : $\beta(i,x,j) = \mu C_{0x} - \quad (i,x,j)$ où μ est un paramètre de contrôle. Dans l'algorithme de Mole et Jameson [Mol76], chaque itération est divisée en deux étapes. On cherche d'abord l'emplacement d'insertion, c'est-à-dire deux clients adjacents i^* et j^* dans une tournée tels que $(i^*,x,j^*) = min\{\ (i,x,j)|(i,j) \in T\ et\ l \neq T\}$. S'il existe un tel couple (i^*,j^*), on cherche ensuite le client x^* à insérer tel que $\beta(i^*,x^*,j^*) = max\{\ \beta(i^*,x,j^*)|\ x \neq T\}$. Après l'insertion du client x^* entre i^* et j^*, une procédure d'amélioration de type 3-opt [Lin65] est appliquée sur la tournée courante. Si on n'a pas trouvé de couple (i_x,j_x), alors on crée une nouvelle tournée $(0,x,0)$ qui devient la tournée courante.

Christofides *et al.* [Chr79] ont apporté une amélioration à la méthode d'insertion. Ils proposent un algorithme en deux étapes : dans la première étape, un algorithme d'insertion séquentiel est appliqué pour obtenir des tournées réalisables. La deuxième étape repose sur des insertions parallèles : N tournées mono-client sont initialement construites, où N est le nombre de tournées obtenues dans la première étape. Les clients restant sont insérés en utilisant un critère de regret, où la différence entre les deux meilleurs coûts d'insertion est prise en compte. Une procédure de « 3-opt » est appliquée après chaque insertion pour améliorer la qualité de la tournée actuelle. Cordeau *et al.* [Cor07] indiquent que cet algorithme est meilleur que l'algorithme de Mole et Jameson.

(b) Heuristiques en deux phases

L'heuristique en deux phases repose sur l'idée que l'on peut décomposer un problème de tournées de véhicules en deux sous-problèmes séparés : partitionnement (*clustering*) et routage (*route*). Le « partitionnement » partitionne l'ensemble des clients en plusieurs sous-ensembles, chaque sous-ensemble étant desservi par une seule tournée. Le « routage » consiste à déterminer la séquence de visite des clients dans une tournée. Il existe deux types de méthodes combinant ces deux sous-problèmes : « *cluster-first, route-second* » et « *route-first, cluster-second* ».

- « cluster-first, route-second »

Dans la méthode « *cluster-first, route-second* », les clients sont d'abord regroupés dans des clusters, les tournées sont ensuite construites en déterminant l'ordre des clients dans chaque cluster. Dans cette méthode, la partie partitionnement joue un rôle plus important que la partie routage. La détermination du routage des clients dans un cluster se ramène à la résolution d'un TSP. Il existe plusieurs techniques pour le partitionnement. On peut distinguer trois classes principales : (1) méthode de regroupement élémentaire ; (2) méthode *branch-and-bound* tronquée ; (3) algorithmes de pétale.

La méthode de regroupement élémentaire consiste à affecter séquentiellement des clients dans les clusters. Wren [Wre71], Wren *et al.* [Wre72], Gillett *et al.* [Gil74] ont proposé un algorithme de *sweep* pour la construction des clusters. Cet algorithme s'applique sur des instances planaires. Les clusters sont créés de façon séquentielle. Pour chaque cluster, on prend initialement un client arbitraire comme client initial du cluster. À chaque étape, le client qui possède le plus petit angle polaire par rapport au dépôt et au client initial est affecté dans le cluster courant. Dans le cas où le cluster courant ne peut plus accepter un client (contrainte de capacité du véhicule), un nouveau cluster est créé. Fisher et Jaikumar [Fis81] ont introduit un algorithme qui permet de déterminer des clusters en résolvant un *problème d'affectation généralisé* (*Generalized Assignment Problem* - GAP). Il s'agit de chercher l'affectation de coût minimal d'éléments à un ensemble de boîtes de capacité donnée. Chaque élément est caractérisé par un poids et un coût d'affectation à chaque boîte. Les éléments et les boîtes correspondent respectivement aux clients et aux clusters. Cet algorithme utilise un critère géométrique : le coût d'affection d'un client à un cluster est égal à la distance du client *seed* au cluster, où le client *seed* est choisi à l'étape initiale de l'algorithme.

La méthode *branch-and-bound* tronquée est introduite par Christofides *et al.* [Chr79]. Les tournées sont déterminées en utilisant une version limitée de l'algorithme de *branch-and-bound*. L'arbre de recherche dans cette méthode contient autant de niveaux que de clusters, dont le nombre est connu a priori. Au niveau n de l'arbre, on génère un ensemble de clusters possibles, à partir des clients non affectés, en combinant les critères des méthodes d'insertion. Chaque cluster i possède un ensemble de clients S_i. Son coût $f(i)$ est donné par $f(i) = t(S_i \cup \{0\}) + u(F \backslash S_i)$ où F est l'ensemble des clients non affectés, $t(x)$ est la fonction qui donne le coût associé à la solution optimale du TSP pour l'ensemble des sommets x, $u(y)$ est la fonction qui donne le coût de l'arbre couvrant minimum (*Minimum Spanning Tree* - MST) pour l'ensemble des sommets y. Le cluster avec le plus petit coût est affecté à la $n^{ème}$ tournée. On passe ensuite au niveau suivant avec l'ensemble des clients non affectés ($F \backslash Si$). Laporte

et al. [Lap02] ont montré que cette méthode donne souvent des meilleures solutions par rapport à la méthode de regroupement élémentaire.

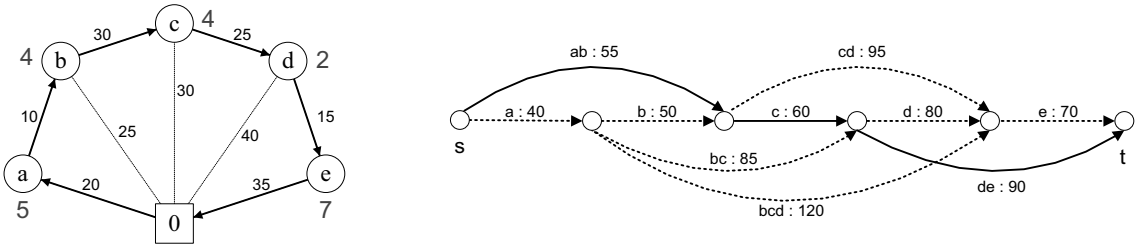
Le principe d'un algorithme de type pétale consiste à générer un grand nombre de tournées réalisables, dites « pétales », puis à sélectionner un sous-ensemble à l'aide d'une méthode de *set-partitioning*. Foster *et al.* [Fos76], Ryan *et al.* [Rya93] ont présenté des règles heuristiques pour déterminer le sous-ensemble à sélectionner, dit « 1-pétales ». Renaud *et al.* [Rena96] ont décrit une extension qui tient compte aussi des configurations des tournées, dite « 2-pétales ». Elle tient compte du cas où les deux tournées s'intersectées (voir la Figure 1.15 (a)) et du cas où une tournée est incluse dans l'autre (voir la Figure 1.15 (b)). Cordeau *et al.* [Cor07] indiquent que la performance de cet algorithme est généralement supérieure à celle de l'algorithme de *sweep* dans la méthode de regroupement élémentaire.

(a) inclusion des deux tournées (b) deux tournées s'intersectées

Figure 1.15 : Solutions avec 2-pétales

- « *route-first, cluster-second* »

Une méthode de « *route-first, cluster-second* » consiste à construire d'abord un tour géant de type hamiltonien en relâchant la contrainte sur la capacité (partie routage). On découpe ensuite ce tour en plusieurs morceaux, chacun correspondant à une tournée de véhicules (partie partitionnement). Cette méthode est présentée initialement par Beasley [Bea83]. Il traite la partie partitionnement comme un problème de plus court chemin. Dans cette approche, la distance/coût entre deux clients i et j est égale à $C_{0i} + C_{0j} + L_{ij}$ où L_{ij} est le coût pour aller de i à j dans la tournée géante. La Figure 1.16 est tirée de [Prin04]. Elle illustre la transformation du partitionnement en un problème de plus court chemin. La partie (a) donne le graphe G initial avec le tour géant $(0, a, b, c, d, e, 0)$. Un graphe auxiliaire G' est construit à partir de G et du tour géant. G' est un graphe à niveaux dans lequel chaque niveau correspond au passage sur un client (dans l'ordre fixé par le tour géant). Chaque sommet correspond à l'insertion d'un retour au dépôt entre les deux clients consécutifs du tour géant. Chaque arc a un coût correspondant au coût de la tournée qu'il représente. Le sommet initial *s* et final *t* permettent de ramener le problème à la recherche d'un plus court chemin de s à t. Les arcs de G' correspondent aux tournées possibles en respectant la capacité. Le libellé sur un arc contient les clients de la tournée correspondante et le coût associé. La partie (b) montre un tel graphe auxiliaire, le libellé « ab:55 » représente la tournée $(0, a, b, 0)$ avec un coût de 55. La solution (arcs en traites pleins) correspond à trois tournées : $(0, a, b, 0)$, $(0, c, 0)$ et $(0, d, e, 0)$ avec un coût total de 205.



(a) tournée géant (b) problème de plus court chemin

Figure 1.16 : Problème de plus court chemin au partitionnement

Haimovich *et al.* [Hai85] ont montré que si les demandes des clients sont unitaires, l'algorithme de Beasley donne des solutions optimales asymptotiquement. Bertsimas *et al.* [Ber96] ont utilisé cet algorithme pour des instances avec des demandes non-unitaires. En général, la performance de l'heuristique « *route-first, cluster-second* » est médiocre par rapport à l'approche inverse. Cependant, Prins [Prin04] a présenté une méta-heuristique (à base de l'algorithme génétique), utilisant le principe de « *route-first, cluster-seconde* ». La transformation du tour géant en tournées est réalisée par une méthode dite *Split*. Les performances de cette approche sont très bonnes. Pour plus de références sur ce type d'heuristique, on peut citer les travaux de [Duh10, Duh12].

(c) Heuristiques d'amélioration

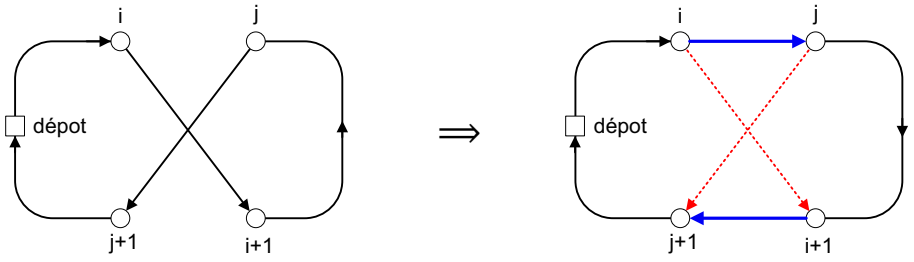
Les heuristiques d'amélioration pour les problèmes de tournées de véhicules consistent essentiellement en des algorithmes de recherche locale. Ils permettent d'améliorer itérativement la solution courante S en sélectionnant un voisin dans son voisinage $N(S)$. L'exploration de $N(S)$ se fait en considérant les échanges d'un certain nombre d'éléments (sommets ou arcs) de S. Dans un processus d'exploration, à chaque itération, la solution courante est remplacée par le meilleur voisin $S' \in N(S)$ au sens du critère d'acceptation. Il existe trois types de critères : (1) *first-available* : on prend le premier voisin S' qui est meilleur que S dans $N(S)$; (2) *best-available* : on prend le meilleur voisin S' dans $N(S)$; (3) *random-available* : on sélectionne aléatoirement un voisin S' dans $N(S)$. Dans la plupart des approches, on ne considère que les voisins améliorants de $N(S)$. Une heuristique d'amélioration est souvent utilisée sur une solution générée par une heuristique de construction.

Les voisinages pour les problèmes de tournées de véhicules reposent essentiellement sur la technique d'échange d'arcs. Cette technique reprend les mouvements de type « *k-opt* » de Lin [Lin65], où k est le nombre d'arcs enlevés/ajoutés dans des tournées de la solution courante. La complexité d'exploration est en $O(n^k)$ pour examiner tous les voisins dans $N(S)$. La plupart des recherches locales utilisent du « 2-opt » ou du « 3-opt ». Selon l'état de l'art fait par Braysy *et al*. [Bra01], on les distingue en deux groupes : *swap* interne et *swap* externe.

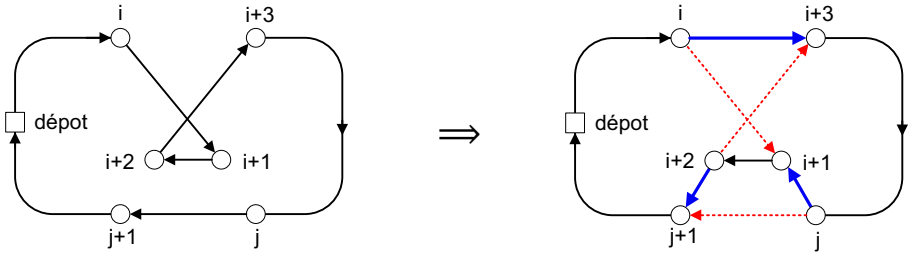
Les mouvements de type *swap* interne réalisent les changements à l'intérieur d'une tournée. Les Figure 1.17 (a) et (b) illustrent des mouvements internes à une tournée : (a) correspond au « 2-opt » et (b) à un « Or-opt » (« 3-Opt » conservant l'ordre de parcours sur le segment déplacé) [Or76]. Ces mouvements ont été initialement conçus pour le TSP.

Les Figure 1.17 (c), (d) et (e) correspondent à des mouvements externes, le « 2-opt* » illustré en (c) a été introduit par Potvin et Rousseau [Pot95]. Il échange les fins de deux tournées et

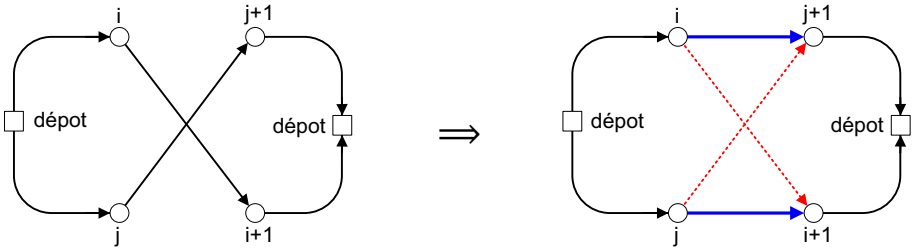
conserve l'ordre de parcours. Le « *relocate* » (Figure 1.17 (d)) consiste à déplacer un sommet dans une autre tournée. C'est un cas particulier de 3-opt, développé par Prosser *et al.* [Pro96]. Le mouvement de la Figure 1.17 (e) combine deux « *relocate* » dans un « 4-opt » spécialisé. Il existe aussi des opérateurs plus complexes comme le transfert cyclique. On pourra se référer aux états de l'art [Kin97, Bra01, Lap02, Cor07].



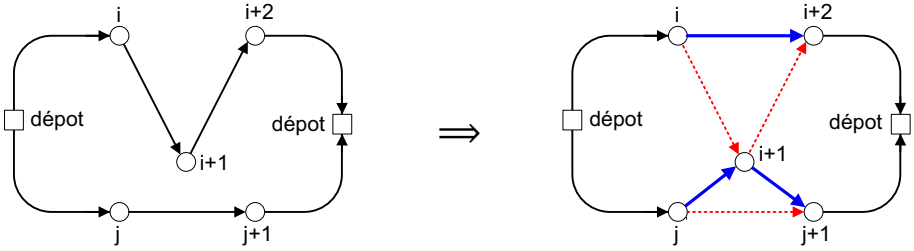
(a) « 2-opt » interne



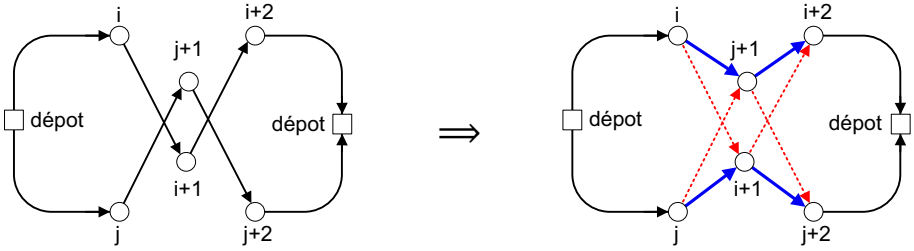
(b) « or-opt » (un cas de «3-opt» interne)



(c) « 2-opt* » externe ou «*cross*»



(d) « *relocate* » (un cas de «3-opt» externe)



(e) échange (un cas de « 4-opt » externe)

Figure 1.17 : Opérateurs de transformation locale

1.2.2.2 Méta-heuristiques

Depuis ces deux dernières décennies, les méta-heuristiques ont été beaucoup étudiées dans la littérature. Par rapport aux heuristiques, les méta-heuristiques présente un schéma de résolution général qui peut être adapté à de nombreux problèmes combinatoires. Les méta-heuristiques comme la recherche tabou et le recuit simulé ont été régulièrement utilisées pour traiter les problèmes de tournées de véhicules. Les travaux de Osman et Kelly [Osm96], de Gendreau *et al.* [Gen98, Gen02], de Cordeau *et al.* [Cor04, Cor05, Cor07] donnent une référence précise sur l'application des méta-heuristiques aux problèmes de tournées. On pourra aussi se référer au livre édité par Glover et Kochenberger [Glo03].

Selon la classification proposée par Cordeau *et al.* [Cor07], on distingue les méta-heuristiques pour des problèmes de tournées de véhicules en trois classes : (a) recherche voisinage ; (b) recherche à base de population ; (c) mécanisme d'apprentissage. Les limites de ces 3 classes sont parfois floues : les approches les plus performantes sont souvent une hybridation de plusieurs méthodes situées dans les différentes classes. La combinaison des méthodes des classes (a) et (b) apparaît fréquemment dans la littérature. On pourra se référer aux méthodes dans [Prin04, Mes05, Cor07].

(a) Recherche par voisinage

Une méta-heuristique de type recherche voisinage consiste à découvrir l'espace des solutions par des déplacements itératifs de la solution courante vers une solution voisine. L'idée principale de ce type de méta-heuristiques est similaire aux algorithmes de recherche locale dans les heuristiques d'amélioration. Soit $V(S_t^*)$ l'ensemble des solutions voisines pour S_t^* où S_t^* est la solution courante à l'itération t, on explore d'abord tous les éléments de $V(S_t^*)$, et ensuite on en choisit une selon le critère de sélection utilisé. La solution choisie à l'itération t deviendra la solution courante (notée S_{t+1}^*) à l'itération $t+1$, puis on recommence l'exploration du voisinage. Les itérations se terminent quand le critère d'arrêt est satisfait. La Figure 1.18 montre le mécanisme de déplacements itératifs d'une recherche voisinage. Les trois critères utilisés par les heuristiques d'amélioration peuvent être adaptés au choix de la solution S_{t+1}^* à l'itération t. On les note comme suit : *first-accept*, *best-accept* et *random-accept*. Dans certaines applications de ces critères, on autorise l'acceptation d'une solution S_{t+1}^* moins performante que la solution courante S_t^*. Dans ce cas, il est possible que la solution acceptée à l'itération t soit la solution de l'itération précédente, *i.e.* $S_{t+1}^* = S_{t-1}^*$. La recherche voisinage risque alors d'être piégée dans une boucle. Il est donc nécessaire d'utiliser un processus d'élimination des cycles entre les itérations. Une stratégie simple est de conserver les solutions explorées (dans une itération) pour les itérations suivantes.

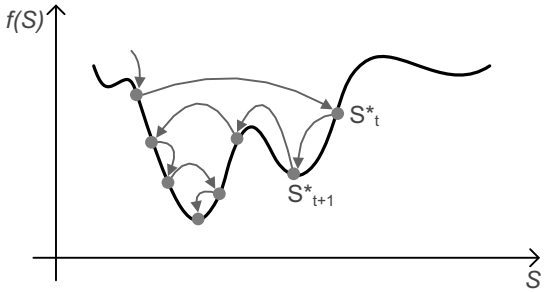


Figure 1.18 : Mécanisme de type recherche voisinage

Les méta-heuristiques comme le *recuit simulé*, la *recherche tabou* et le *recuit déterministe* possèdent une caractéristique forte de type recherche voisinage (descente).

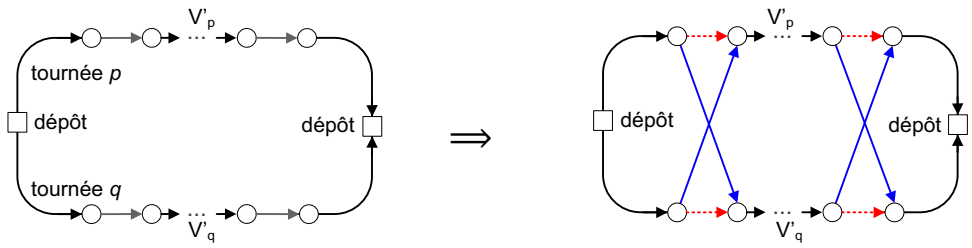
- Le recuit simulé [Osm93] utilise la combinaison des critères *first-accept* et *best-accept* : à l'itération t, si la solution voisine S'_t est meilleure que la solution courante S^*_t, alors $S^*_{t+1} = S'_t$ et on sort de l'itération t. Si on ne trouve pas de solution meilleure que S^*_t parmi toutes ses solutions voisines : alors, on utilise une probabilité $p_t \in [0,1]$ à la sélection de la solution S^*_{t+1}. La valeur de p_t est définie par : $p_t = e^{-\Delta/\theta_t}$ où $\Delta = f(S''_t) - f(S^*_t)$, Θ_t est la température liée à l'itération t, f est la fonction donnant le coût d'une solution et S''_t est la meilleure solution dans $V(S^*_t)$. Avec la probabilité p_t, on accepte la meilleure solution dans $V(S^*_t)$ comme solution courante pour l'itération $t+1$, *i.e.* $S^*_{t+1} = S''_t$. La solution courante S^*_t sera donc gardée pour l'itération $t+1$ avec une probabilité $(1 - p_t)$.
- La recherche tabou possède une liste tabou L_t qui contient toutes les solutions S^*_{t-b} où t est le numéro d'itération actuelle, B est le nombre d'itérations interdites et $b = 1,..,B$. Elle utilise le critère *best-accept*. À chaque itération, la meilleure solution non interdite dans $V(S^*_t)$ est sélectionnée comme la solution courante de l'itération suivante, *i.e.* $S^*_{t+1} = \min_{S \in V(S^*_t)} \{f(S) | S \neq L_t\}$.
- Le recuit déterministe est similaire au recuit simulé. Par contre, le critère utilisé est *first-accept*, qui est un critère d'acceptation déterministe. Il existe principalement deux versions d'algorithmes pour le recuit déterministe : seuil d'acceptation (*threshold accepting*) et déplacement « *record-to-record* ». À l'itération t, une solution S_t est sélectionnée aléatoirement. Dans un algorithme utilisant un seuil d'acceptation, on fait $S^*_{t+1} = S_t$ si $f(S_t) \leq f(S^*_t) + \theta_1$ où θ_1 est le paramètre de contrôle liée à la température. Dans un algorithme de déplacement « *record-to-record* », soit S^* est la meilleure solution connue, alors $S^*_{t+1} = S_t$ si $f(S_t) \leq \theta_2 f(S^*)$ où θ_2 est le paramètre de contrôle. L'algorithme 1.3 montre le schéma général des méta-heuristiques de type recherche voisinage.

Algorithme 1.3 : Schéma des méta-heuristiques de type recherche voisinage

```
(i) phase d'initialisation
    Générer une solution S ;
    S* ← S ;  // mémoriser la meilleure solution

(ii) phase d'amélioration itérative
    Tant que le critère d'arrêt non satisfait faire
    |   stop ← faux ;
    |   Pour toute S' dans voisinage(S) et non stop faire
    |   |   Si solution S' satisfait le critère d'acceptation alors
    |   |   |   stop ← vrai ;
    |   |   Fin Si
    |   Fin Pour
    |
    |   Si non stop alors
    |   |   S' ← extraire(S', S) ;
    |   Fin Si
    |
    |   Si S' est meilleure que S* alors
    |   |   S* ← S' ;
    |   Fin Si
    |   S ← S' ;
    Fin Tant que
    Retourner S ;
```

La performance d'une méta-heuristique de type recherche voisinage dépend de sa définition des solutions voisines. Il existe plusieurs méthodes d'exploration de solutions voisines. Par exemple, Osman [Osm93] a proposé un processus dit « -*interchange* » pour le recuit simulé. Dans ce processus, on considère p et q deux tournées dans la solution courante S_t^*. Soient V_p' et V_q' respectivement un sous-ensemble de clients choisis dans les deux tournées p et q, et $|V_p'| \leq \quad et \; |V_p'| \leq$ où λ est un nombre entier. Les deux sous-ensembles de clients sont échangés entre deux tournées (voir la Figure 1.19). La solution obtenue est considérée comme une solution voisine de S_t^* si le mouvement est faisable. Dans le cas où l'un des deux sous-ensembles est vide, alors ce mouvement devient un déplacement d'un groupe de client d'une tournée vers l'autre. Quand la valeur λ est petite ($\lambda \leq 2$), on a une grande quantité de combinaisons possibles pour V_p' et V_q'. Le processus « -*interchange* » utilise généralement le critère *first-accept*. Il est appliqué souvent dans la recherche *tabou* et le recuit déterministe. Par ailleurs, il existe plusieurs méthodes d'exploration de solutions voisines comme le « 3-opt », la procédure GENI (pour *Generalized Insertion*), l'éjection des chaînes, etc. On pourra se référer à l'état de l'art dans [Lap02, Cor07].



(a) tournées p et q avant un mouvement (b) tournées p et q après un mouvement

Figure 1.19 : Processus « -*interchange*»

L'application des méta-heuristiques de type recherche voisinage sur des problèmes de tournées de véhicules a commencé au début des années 90. Osman [Osm93] a introduit un algorithme de recuit simulé en utilisant un processus de « 2-*interchange* » pour générer des solutions voisines. Au lieu d'utiliser une fonction décroissante, Osman diminue successivement la température θ_t sous la condition que la solution courante S_t^* est modifiée. Dans le cas où la meilleure solution trouvée est égale à la solution courante, il prend la plus grande température entre la moitié de la température actuelle et la température de la meilleure solution voisine. Cet algorithme permet de trouver de bonnes solutions mais il n'est pas encore compétitif avec les meilleurs algorithmes de recherche tabou dans la même période.

De nombreux algorithmes de recherche tabou ont été présentés pendant ces dernières années. Willard [Wil89] a introduit initialement un algorithme tabou pour le CVRP, une solution est présentée sous forme d'un tour géant qui contient plusieurs réplications du dépôt. Les solutions voisines sont obtenues via un opérateur de type « 3-opt ». Taillard [Tai93] a présenté un algorithme tabou qui utilise le « 1-*interchange* » comme processus d'exploration de solutions voisines. À chaque itération, les tournées d'une solution voisine sont optimisées respectivement par un algorithme exact [Vol83] du TSP. Dans la liste tabou utilisée, les solutions sont conservées durant un nombre d'itérations donné. Ce nombre d'itération évolue de manière aléatoire durant l'algorithme. Taillard utilise un mécanisme de diversification afin d'obtenir un espace de voisinage plus large. Cet algorithme est considéré comme l'un des meilleurs algorithmes tabou. Gendreau *et al.* [Gen94] ont amené aussi un algorithme tabou

pour le CVRP, dite *taburoute*. Ils utilisent la procédure GENI pour définir des solutions voisines, dans laquelle un client est enlevé d'une tournée courante, avant d'être inséré dans l'une des K tournées voisines les plus proches. Ce client ne peut pas retourner dans sa tournée d'origine durant les θ prochaines itérations. Ils implémentent un mécanisme de diversification qui est associé à une pénalité pour les solutions infaisables. Rego *et al.* [Reg96] ont introduit un algorithme tabou en utilisant une éjection des chaînes pour générer des solutions voisines. Un mouvement d'éjection des chaînes de l niveaux consiste à remplacer les triplets $(v_{i-1}^k, v_i^k, v_{i+1}^k)$ par les triplets $\left(v_{i-1}^k, v_i^{k-1}, v_{i+1}^k\right)$ pour $k = 1, .., l$ et $(v_{i-1}^k, v_i^k, v_{i+1}^k)$ est remplacé par $(v_{i-1}^k, v_i^l, v_{i+1}^k)$ pour $k = 0$ où v_i^k est le $i^{ème}$ client dans la tournée k. Xu *et al.* [Xu96] ont présenté un algorithme tabou en combinant un mouvement de changement des sommets entre deux tournées et une éjection des chaînes. Cet algorithme donne de bonnes solutions pour certaines instances du CVRP. Toth *et al.* [Tot03] ont développé ainsi un algorithme tabou en combinant l'algorithme de Taillard et le *taburoute*. Ils appliquent des mouvements à la fois intra et inter tournées pour obtenir des solutions voisines.

Le recuit déterministe est appliqué initialement par Golden et al [Gol98] pour le CVRP. Plus récemment, Li et al [Li05] ont combiné le principe de « *record-to-record* » à l'algorithme de Toth *et al.* [Tot03]. Ils génèrent des solutions voisines en utilisant le « 2-opt » intra/inter tournées. Il existe aussi d'autres méta-heuristiques de type recherche voisinage pour le CVRP, comme la VLNS (*Very Large Neighborhood Search*), la VNS (*Variable Neighborhood Search*), l'ELS (*Evolutionary Local Search*), le GRASP (*Greedy Randomized Adaptive Search Procedure*), etc. Pour plus de détails concernant ces algorithmes, on pourra se référer à [Car01, Han01, Lap02, Erg03, Cor04, Cor07].

(b) Recherche population

Une méta-heuristique de type recherche population manipule un ensemble de solutions en parallèle à chaque itération (voir la Figure 1.20). Cet ensemble est appelé « population ». Toutes les solutions dans une population sont connues a priori. Il n'existe pas de définition d'une solution voisine dans une méta-heuristique de type recherche population. L'amélioration des solutions est effectuée à l'aide de techniques de renouvellement de la population. Une nouvelle génération de la population est produite à chaque itération. Les méta-heuristiques comme l'*algorithme génétique*, le *programme à mémoire adaptative* et l'*algorithme mémétique* sont typiquement du type recherche population.

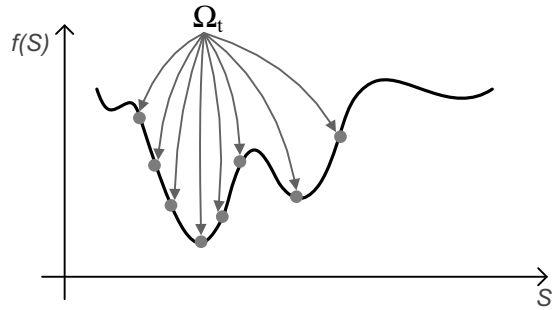


Figure 1.20 : Génération d'une population

Dans un algorithme génétique (GA, pour *Genetic Algorithm*), à une itération t, la population Ω_t se compose d'un ensemble de N chromosomes dont chacun représente une solution faisable. La génération d'une novelle population se réalise par des opérateurs de « *crossover* »

(croisement) et de « *mutation* » : on sélectionne deux chromosomes (parents) dans la population actuelle pour générer deux nouveaux chromosomes (enfants) à l'aide d'un opérateur de « *crossover* », puis on applique aléatoirement un mouvement de mutation sur les deux chromosomes obtenus. On répète k fois la même opération ($k \leq N/2$). Une nouvelle population Ω_{t+1} est obtenue en remplaçant les $2k$ chromosomes les moins intéressants dans $\Omega_t$par des chromosomes nouvellement générés. L'algorithme 1.4 est donne le schéma général d'un algorithme génétique.

Algorithme 1.4 : Schéma de l'algorithme génétique

```
(i) phase d'initialisation
    Générer un ensemble de solutions (population de chromosomes) Ω ;

(ii) phase d'amélioration itérative
    Tant que le critère d'arrêt non satisfait faire
    |   Ω' ← ∅     // ensemble des chromosomes générés
    |   Répéter
    |   |   Choisir deux chromosomes P1 et P2 dans Ω
    |   |   Si crossover (P1, P2) réalisable alors
    |   |   |   Générer deux chromosomes enfants E1 et E2
    |   |   |   Si la condition de mutation est satisfaite pour E1 alors
    |   |   |   |   E1 ← mutation_opérateur(E1)
    |   |   |   Fin Si
    |   |   |   Si la condition de mutation est satisfaite pour E2 alors
    |   |   |   |   E2 ← mutation_opérateur(E2)
    |   |   |   Fin Si
    |   |   |   Ω' ← Ω' U {E1, E2}
    |   |   Fin Si
    |   Jusqu'à |Ω'| >= 2*k
    |   Ω ← nouvelle_population(Ω, Ω') ;   // générer la nouvelle population
    Fin Tant que
    x = extraire(Ω) ;          // choisir la meilleure solution dans la population
    Retourner x ;
```

Dans le cadre de l'utilisation d'algorithme génétique à des problèmes de tournées de véhicules, un chromosome est souvent présenté sous la forme d'une tournée, qui contient une chaîne de nombres entiers de taille fixée où chaque nombre entier correspond à un client. La position des clients représente l'ordre de visite dans la tournée. Il existe plusieurs types d'opérateur de « *crossover* », comme le *crossover* à 1-point et le *crossover* OX. Les deux exemples dans la Figure 1.21 illustrent respectivement ces deux types d'opérations appliquées à un problème de CVRP. La partie (a) correspond au *crossover* à 1-point. Chaque chromosome parent représente une tournée. Dans cet opérateur, on cherche à déterminer un point de croisement dans chaque chromosome parent, on génère ensuite les deux chromosomes $E1$ et $E2$ en échangeant les sous-chaînes entre les deux parents. Les chromosomes générés ne sont pas toujours réalisables. La partie (b) montre une opération de *crossover* OX : on sélectionne aléatoirement deux points de coupe i et j, puis on copie la partie $P1[i]$ à $P1[j]$ du parent $P1$ dans $E1[i]$ à $E1[j]$ de l'enfant $E1$, ensuite le contenu du parent $P2$ est balayé de façon circulaire à partir de la position $j + 1$ pour compléter les cases vides dans $E1$. Le complément dans $E1$ commence aussi à la position $j + 1$. On effectue une opération similaire pour obtenir le chromosome $E2$. Dans l'exemple de la Figure 1.21 (b), on considère i=2 et j=4. Le fils E1 est égale à (- **2 3 4** - - -) après avoir reçu la partie du parent $P1$ (de $P1[2]$ à $P1[4]$). Puis on remplit circulairement $E1$ (à partir de la position E1[5]) par des éléments du parent $P2$ (en commençant à P2[5], P2[k] n'existe pas en $E1$, pour $k \in \{1, .., |P2|\}$). Après la première

opération de complément, *E*1 devient (- **2 3 4** 1 - -). Le complément se termine quand *E*1 est égale à (5 **2 3 4** 1 6 7). En plus de ces deux types de *crossover*, il en existe d'autres comme *crosseover* PMX [Gol85].

Chromosomes	Codages
Parent *P1* :	1 2 3 4 | 5 6 7
Parent *P2* :	**7 3 2 5 | 4 1 6**
Enfant *E1* :	1 2 3 4 **4 1 6**
Enfant *E2* :	**7 3 2 5** 5 6 7

(a) « *crossover* » à 1-point

Chromosomes	Codages
Parent *P1* :	**1 | 2 3 4 | 5 6 7**
Parent *P2* :	7 | 3 2 5 | 4 1 6
Enfant *E1* :	5 **2 3 4** 1 6 7
Enfant *E2* :	**4** 3 2 5 **6 7 1**

(b) « *crossover* » OX

Figure 1.21 : Deux types de « *crossover* »

Une opération de mutation est effectuée aléatoirement sur des chromosomes générés. Elle est souvent un opérateur de *swap* ou un opérateur de type RAR (pour *remove-and-reinsert*). Le *swap* sélectionne deux clients aléatoirement et échange leurs positions dans un chromosome. Un opérateur RAR retire un client et l'insère à une autre position de manière aléatoire. Pour plus de détails sur les techniques de « *crossover* » ou les opérateurs de mutation, on pourra se référer à [Gol85, Bea94, Pot96, Dre03, Prin04].

Dans la littérature, l'algorithme génétique a été présenté initialement par Holland [Hol75]. Thangiah [Tha95] qui introduit cet algorithme pour le CVRP avec fenêtres de temps. Il utilise le *crossover* à 1-point et un opérateur de RAR dans son algorithme. Le chromosome est codé comme une chaîne de bits. Chaque chromosome généré est amélioré par un processus « - *interchange* ». Schmitt [Sch95] a développé une application de l'algorithme génétique pour le CVRP, dans laquelle il s'applique la méthode « *route-first, cluster-seconde* ». Un chromosome est une chaîne de nombres entiers, qui correspond à une tournée géante contenant tous les clients. Une solution du CVRP est obtenue à l'aide d'une procédure de balayage qui commence par le premier client dans la chaîne de chromosome. Une tournée de véhicules est identifiée quand la contrainte de capacité ou la contrainte de durée maximum est violée en rajoutant le client suivant dans la chaîne. Une nouvelle tournée commence par le client suivant. Il utilise le *crossover* OX et le *swap* comme méthode de mutation. Plus récemment, Prins [Prin04] a développé un algorithme généitique, qui utilise aussi la notion de «*route-first, cluster-second*». Dans son algorithme, Prins a développé l'idée de Beasley [Bea83]. Il introduit une technique dite *Split* pour découper une tournée géante en plusieurs tournées de véhicules. Un chromosome est aussi une tournée géante représentée par une chaîne de nombres entiers. Prins utilise le *crossover* OX et un algorithme d'amélioration de type recherche voisinage pour les opérations de mutation.

Outre le GA, Rochat *et al.* [Roc95] ont introduit un algorithme de programme à mémoire adaptative. Une mémoire adaptive est en fait une population qui sauvegarde des solutions de bonne qualité. Le renouvellement de population se fait de façon à remplacer les mauvaises solutions par les meilleures. Pour générer une nouvelle solution, l'algorithme sélectionne plusieurs solutions dans la population et les combine. Tarantilis *et al.* [Tar02] ont proposé aussi un algorithme de programme à mémoire adaptive. Ils utilisent un processus d'extraction des tournées, dit « *bones* » pour générer des nouvelles solutions.

Berger *et al.* [Ber04] ont présenté un algorithme mémétique, qui travaille sur deux populations de taille fixée. Le renouvellement d'une population se réalise de façon à remplacer les

solutions parentes par des solutions générées. Il permet aussi des migrations de solutions entre deux populations. Dans la génération d'une solution fils, on regroupe des tournées dans les deux solutions parentes en retirant les clients qui violent des contraintes, et en insérant des clients non-routés selon un critère de proximité. Mester *et al.* [Mes05] ont introduit un algorithme mémétique en combinant une recherche locale [Vou97] et une stratégie d'évolution [Rec73]. La stratégie d'évolution utilise une règle déterministe pour sélectionner une solution parente dans la population et créer avec une seule solution enfant. La solution générée est ensuite améliorée par une procédure qui consiste en plusieurs algorithmes d'amélioration. Elle remplacera la solution parente si elle est meilleure.

(c) Mécanisme d'apprentissage

L'*algorithme de colonies de fourmis* et le *réseau de neurones* possèdent un mécanisme d'apprentissage. Ce mécanisme repose sur le traitement des informations accumulées sur les arcs tout au long de l'algorithme. La qualité d'une solution est ainsi mesurée par la somme de ces informations. L'algorithme de colonies de fourmis provient de l'observation des méthodes d'exploration des fourmis pour trouver des ressources alimentaires. Dans une telle exploration, des fourmis déposent des phéromones sur leur trajectoire. Ces phéromones contiennent des informations qui attirent d'autres fourmis. Plus le temps passe, plus les chemins vers les ressources alimentaires les plus intéressantes deviennent fréquentés et plus ils sont marqués par une grande quantité de phéromones. En observant ce phénomène chez les fourmis, Colorni *et al.* [Col91] ont proposé initialement l'algorithme de colonies de fourmis. Dans lequel des fourmis artificielles explorent l'espace des solutions en simulant le comportement de fourmis réelles qui cherchent des ressources alimentaires. La qualité d'une solution est mesurée par la quantité de phéromones déposées par les fourmis artificielles. L'algorithme 1.5 montre le schéma général d'un algorithme de colonies de fourmis.

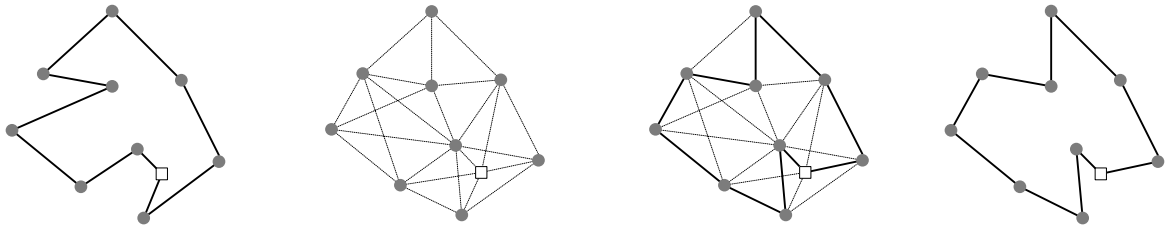
Algorithme 1.5 : Schéma d'un algorithme de colonies de fourmis

```
(i) phase d'initialisation
    M ⟵ nombre d'itérations ;
    Générer une solution S ;
    Initialiser la table de l'intensité phéromone P à partir de S ;

(ii) phase d'amélioration itérative
    Répéter
    |   Choisir des sommets de départ pour les N fourmis ;
    |   Générer une solution pour chaque fourmi ;
    |   Mettre à jours la table de l'intensité phéromone P ;
    |   M ⟵ M – 1 ;
    Jusqu'à M = 0
    S ⟵ extraire(P) ;
    Retourner S ;
```

L'algorithme de colonies de fourmis a été développé initialement pour la résolution du TSP. Dans cet algorithme, chaque arc (i,j) est associé à deux valeurs : l'intensité de phéromone Γ_{ij}, qui évolue durant l'algorithme et la visibilité d'un arc v_{ij}, qui est égale à l'inverse de la longueur de l'arc (i,j). Dans une itération t, on dispose un ensemble de N fourmis artificielles dont chacune est située sur un sommet. Dans l'étape de construction, on dispose d'une probabilité sur la décision de passer du sommet i au sommet j pour une fourmi k. Cette

probabilité est définie par : $P_{ij}^k(t) = (\Gamma_{ij}(t)^\alpha v_{ij}^\beta)/(\sum_{l\in L_i^k} \Gamma_{il}(t)^\alpha v_{il}^\beta) \quad \forall\, j \in L_i^k$ où L_i^k est l'ensemble de déplacements possibles pour une fourmi k sur le sommet i, les deux paramètres et β sont liés à l'importance donnée aux arcs. À la fin d'une itération, la mise à jour de l'intensité des phéromones est définie comme : $\Gamma_{ij}(t+1) = (1-\rho)\Gamma_{ij}(t) + \sum_{k=1,..,N} \Delta_{ij}^k(t)$ où ρ est un paramètre de réglage, $\Delta_{ij}^k(t)$ est la quantité de phéromones déposée par une fourmi k sur un arc (i, j) à l'itération t qui est obtenue par l'équation $\Delta_{ij}^k(t) = 1/\mathrm{C_k(t)}$ avec $C_k(t)$ la longueur de la tournée construite par la fourmi k. En général, le nombre d'itération est fixé dans un algorithme de colonies de fourmis. La Figure 1.22 illustre un exemple d'application de cet algorithme : la partie (a) est une solution initiale (trajectoire initiale des phéromones) ; les parties (b.1) et (b.2) montrent l'évolution de l'intensité des phéromones. Une arrête en couleur foncé (en ligne continue) représente celle qui a accumulé plus de phéromones que les autres. Le graphe sur la partie (c) montre un résultat final en faisant disparaître les arrêtes les moins intéressantes.



(a) tournée initiale (b.1) début des itérations (b.2) fin des itérations (c) résultat final

Figure 1.22 : Application de l'algorithme de colonies de fourmis

L'algorithme de colonies de fourmis est introduit initialement par Colorni *et al.* [Col91]. Kawamura *et al.* [Kaw98] ont proposé une variante hybride de cet algorithme qui utilise un opérateur de « 2-opt » et un algorithme de recuit simulé avec un critère d'acceptation probabiliste. Bullnheimer *et al.* [Bul98] ont développé une application de l'algorithme de colonies de fourmis pour le CVRP dans laquelle les tournées construites à chaque itération sont améliorées par un « 2-opt » avant de faire la mise à jour de l'intensité des phéromones. Plus récemment, Reimann et al [Rei04] ont proposé une version plus performante dite *D-ants*. À chaque itération, ils génèrent des solutions en utilisant une heuristique de construction (Clarke et Wright [Cla64]) et une amélioration de type « 2-opt ». Dans l'heuristique de construction, le coût de réduction (voir Méthode d'économies dans la section 1.2.2.1) est mesuré par une probabilité $P_{ij}^k(t)$ où $P_{ij}^k(t) = (\Gamma_{ij}(t)^\alpha * v_{ij}^\beta)/(\sum_{l\in L_i} \Gamma_{il}(t)^\alpha * v_{il}^\beta)$ avec L_i l'ensemble des arcs possédant les meilleurs coûts d'épargne parmi ceux qui sont liés au sommet i. La meilleure solution de l'itération est ensuite décomposée en plusieurs sous-problèmes. Chaque sous-problème est traité par une même procédure. Dorigo et Blum [Dor05] ont donné des références détaillées sur l'algorithme de colonies de fourmis. On pourra aussi se référer à [Blu04, Dor04, Blu05, Soc08].

Le réseau de neurones possède un même mécanisme d'apprentissage. Il est similaire à l'algorithme de colonies de fourmis du point de vue de l'évolution des informations où la fonction de neurones est équivalente au fonctionnement des fourmis. Le réseau de neurones a été appliqué pour le CVRP par Ghaziri [Gha91] et Matsuyama [Mat91]. Plus récemment, on peut citer les travaux de Potvin *et al.* [Pot03].

1.3 Application des problèmes de flots sur le transport

1.3.1 Problèmes de « synthèse de réseaux » sur le transport

Outre que les problèmes de tournées de véhicules et les problèmes de *Pickup and Delivery*, une application réelle du transport peut aussi être présentée comme un problème de type « synthèse de réseaux ». Les problèmes de « synthèse de réseaux » sont des problèmes d'optimisation combinatoire liés à l'optimisation de réseaux. Ils consistent essentiellement à déterminer la topologie optimale du réseau et le dimensionnement pour écouler un trafic des flots, tout en respectant les contraintes imposées. Ces problèmes sont liés principalement aux applications dans les domaines des télécommunications, des transports et des systèmes de productions. Ils ont pris une importance particulière au cours de ces trente dernières années en *Recherche Opérationnelle*.

Selon l'état de l'art de Gourdin *et al.* [Gour04], ils regroupent des problèmes d'optimisation de réseaux en trois catégories : (i) problèmes de topologie : qui cherchent à identifier la topologie optimale du réseau (le sous-réseau optimal répond à la demande) ; (ii) problèmes concernent au dimensionnement de réseaux, afin de minimiser le coût de infrastructures, tout en assurant un routage efficace ; (iii) problèmes d'allocation de ressources : qui consiste à allouer des ressources disponibles aux différentes demandes qui souhaitent d'employer ces ressources dans le réseau, de manière d'optimiser des divers critères associés. Parmi ces trois catégories, les deux premières catégories peuvent être regroupées dans un cadre dit « conception de réseau », qui s'occupe de la définition du réseau et qui fusionne les caractéristiques des deux catégories. Pour la catégorie (iii), Selon le différent moyen d'affectation, les problèmes d'allocation de ressources sont classés en deux groupes : (a) problèmes de localisation, qui s'agissent de localiser les infrastructures d'un système à l'intérieur d'un espace donné ; (b) problèmes de routage qui cherchent à déterminer les cheminements pour des différentes demande de trafics utilisant l'infrastructure.

Un problème de transport contient deux éléments principaux : des véhicules et des marchandises à transporter. Dans une application de synthèse de réseau à un problème de transport, les véhicules sont traduits en « infrastructures » et les marchandises correspondent aux demandes qui souhaitent d'employer ces infrastructures dans un réseau. La partie correspond au transport d'une marchandise dans un problème de transport peut donc être représenté comme un problème de routage. Pourtant, dans une telle application, la façon d'écoulement de trafic des marchandises ici influe directement sur les « infrastructures » d'un réseau. Cela donne un caractère dynamique au réseau associé (dit réseau dynamique). Elle s'agit de « conception de réseau ». L'application impliquant une telle caractéristique sera présenté dans le chapitre 5.

1.3.1.1 Principaux problèmes de flot

Un problème de synthèse de réseau est souvent modélisé en un modèle de flots. Un modèle de flot classique s'installe sur un réseau de transport $G = (V, E)$ où V l'ensemble de sommets et E l'ensemble d'arcs. Pour modéliser le trafic des objets, on considère que chaque arc peut être traversé par un *flot*. Un *flot simple* f de G est donc décrit par le vecteur $W = (f_{ij} | \forall (i,j) \in E)$. Un composant f_{ij} est la quantité de flot sur l'arc (i,j), qui respecte à la contrainte de

capacité liée à cet arc. Pour les arcs lié au sommet i ($\forall i \in V$), on a $\sum_{i:(j,i)\in E} f_{ji} = \sum_{i:(i,j)\in E} f_{ij}$ (noté $E1$). Cela montre qu'il y a autant de quantité de flots en sortie qu'en entrée sur le sommet, *i.e.* loi de *Kirchhoff*. Dans le cas si le sommet *i* est associé à une demande d'envoie de flot d_i, la loi de conservation devient : $\sum_{i:(j,i)\in E} f_{ji} = \sum_{i:(i,j)\in E} f_{ij} + d_i$. On dirait un flot f achemine une quantité d'objets d du sommet s vers le sommet t, si l'on a $(E1)$ pour tout sommet $k \in V \setminus \{s,t\}$ et les sommets s et t respectent les contraintes suivantes : $\sum_{s:(s,j)\in E} f_{sj} = \sum_{s:(j,s)\in E} f_{js} + d$; $\sum_{t:(j,t)} f_{jt} = \sum_{t:(t,j)} f_{tj} - d$. Le sommet s est correspond à la *source* et le sommet t au *puits* pour la quantité de flot d.

Dans certaines applications, les objets circulant sur le réseau ne sont pas homogènes. Ils concernent de différents types de marchandises. Un flot simple n'est pas capable de traiter ce genre d'applications. On a introduit donc la notion « *multi-flot* ». En considérant qu'il existe M types de marchandises à acheminer. Un vecteur de multi-flot est définit : $W^m = (f_e^m \,|\forall\, e \in E \text{ et } m = 1..M)$. Un flot f_e sur l'arc e est donné par : $f_e = \sum_{m=1..m} f_e^m$ où la valeur de f_e^m soit entière soit fractionnaire.

Un problème de flot est soumis à trois contraintes suivantes : (i) un flot non nul est crée sur la *source* et doit être acheminé au *puits* ; (ii) chaque sommet différent de la *source* et du *puits* doit respecter la loi de conservation de flot ; (iii) chaque arc lié à une contrainte de capacité. Ces trois contraintes sont linéaires. On peut ainsi rajoute d'autres contraintes comme le coût d'utilisation d'un arc. L'objectif est souvent de minimiser le coût d'acheminement. Selon le nombre de flots présentés dans un réseau, on peut avoir essentiellement deux types de problèmes : problèmes du flot de coût minimal et problèmes du multi-flot de coût minimal.

(a) problèmes du flot de coût minimal

Un problème du flot de coût minimal est plus simple relatif à ceux avec multi-flot. Il est défini comme : étant donné un réseau ou graphe orienté $G = (V, E)$ où V l'ensemble de sommets et E l'ensemble d'arcs. Chaque arc $e \quad E$ possède une capacité qui consiste en deux parties : seuil de passage minimal Min_e et seuil de passage maximal Max_e. Il est associé à un coût de parcours C_e. On cherche à déterminer le flot f, qui peut être défini en nombres entiers ou en fractionnaires, tel que :

- un composant f_e avec $e \in E$, il respecte $Min_e \leq f_e \leq Max_e$;
- le coût $Cf = \sum_{e\in E} C_e f_e$ soit minimal.

La formulation ($PL\ 4$) est un modèle du flot de coût minimal. Dans laquelle, la contrainte (4.2) correspond à la capacité d'arc et la (4.3) représente la conservation du flot sur chaque sommet.

$$(PL\ 4)\begin{cases} \text{Minimiser} \sum_{(i,j)\in E} C_{ij} f_{ij} & & (4.1) \\ \text{s.c.} & & \\ Min_{ij} \leq f_{ij} \leq Max_{ij} & \forall i,j \in V, (i,j) \in E & (4.2) \\ \sum_{i:(i,j)\in E} f_{ij} - \sum_{i:(j,i)\in E} f_{ji} = 0 & \forall j \in V & (4.3) \\ f_{ij} \in Z & \forall i,j \in V, (i,j) \in E & (4.4) \end{cases}$$

Si le coût Cf à minimiser est convexe en fonction du flot f, la résolution d'un flot de coût minimal devient un problème d'optimisation convexe [Min81, Min89, Ouo00]. Le flot de coût minimal peut être résolu en temps polynomial par un algorithme d'élimination des circuits de coût négatif (*cycle-canceling algorithm*) ou par un algorithme de plus court chemin successif (*successive shortest path algorithm*) [Min75, Gol89, Ahu98]. Par contre, dans le cas où le coût C*f* est concave, un flot du coût minimal devient alors *NP-difficile* [Min89]. Une solution peut être choisie dans un polyèdre défini par les contraintes de flots et les contraintes de capacités. Cette caractéristique peut être employée pour créer des heuristiques de résolution avec des techniques d'amélioration locale et de contrôle stochastique.

(b) problèmes du multi-flot de coût minimal

Un problème du multi-flot est défini comme : dans un réseau ou graphe orienté $G = (V, E)$, on dispose deux vecteurs de capacités Min et Max indexés sur arc, un vecteur de coût C associé aux arcs, et un ensemble de M types de marchandises. Chaque type de marchandises m contient une source $o(m)$, un puits $d(m)$ et une quantité D_m. On cherche à acheminer un multi-flot $f = \{f_e^m \mid \forall\, e \in E \; et \; m = 1, .., M\}$ qui est positif (siot entière soit fractionnaire) tel que :

- contraintes liées aux demandes : chaque flot f^m achemine la quantité D_m du sommet $o(m)$ vers le sommet $d(m)$;
- contraintes de capacité : un composant f_e avec $e \in E$, respecte $Min_e \leq \sum_{m=1..M} f_e^m \leq Max_e$;
- le coût $Cf = \sum_{e \in E} \sum_{m=1..M} C_e f_e^m$ soit minimal.

Dans le cas où le multi-flot est fractionnaire, alors on peut avoir quatre possibilités dans la résolution d'un tel problème : (i) si le coût Cf est linéaire. Il est souvent de grande taille et de présenter des dégénérescences. La méthode de décomposition Dantzig-Wolfe peut être appliquée dans ce cas ; (ii) si le coût Cf est convexe et on ne tient pas compte des contraintes de capacité, alors l'optimalité par rapport à chacun des sous-problèmes induits en fixant tous les flots f^m sauf un permet de garantir l'optimalité locale. Dans ce cas, un algorithme dit de *Déviation de Flot* peut être mis en jeu ; (iii) si le coût Cf est concave et on ne tient pas compte des contraintes de capacité, alors chacun des flots soit mono-chemin dans ce cas. La solution optimale peut être cherchée à l'intérieur de l'ensemble des sommets du polyèdre. On peut utiliser des heuristiques basées sur l'exploitation d'un mécanisme de transformation locale ; (iv) si le coût Cf est convexe et on tient compte des contraintes de capacité, alors il est devenu un problème d'optimisation convexe. Il existe nombreuse méthodes pour ce problème : algorithme de Déviation de Flot, méthode de points intérieurs [Chi94, Cha02], méthode de décomposition *Dantzig-Wolfe* et méthode par agrégation et coupes [Bend01].

Dans le cas où le multi-flot est entier, alors la résolution d'un problème du multi-flot de coût minimal est en complexité *NP-difficile*. Pour plus de références sur des méthodes de traitement, on peut citer aux travaux dans [Fdi04a, Fdi04b].

1.3.1.2 Problème du flot impliquant le problème de synthèse de réseaux

Le *Capacited Flow Assignment* (CFA) est un problème le plus générique aux problèmes du multi-flot. Il est défini comme suivant : sur un graphe/réseau $G = (V, E)$ où V l'ensemble de sommets et E l'ensemble d'arcs. La topologie et le dimensionnement initial de G sont souvent donnés. Un flot d'infrastructure (ou flot de support) entier F 0 et un multi-flot (entier ou fractionnaire, exemple flot de marchandises) $f = (f_i, i = 1, .., n)$ 0 où n est le nombre de composant du f, tels que :

- (i) contrainte structurelles sur F ;
- (ii) pour tout arc e E, F_e et f_e respectent la contrainte de couplage ;
- (iii) chaque composant f_i de multi flot f achemine une quantité de flot du sommet source vers le sommet puits ;
- (iv) F *et* f minimise un coût $C(F)$ (*coût d'infrastructure*) + $P(f)$ (*coût de qualité des routages*).

La contrainte (i) peut être discrète ou réel, elle rassure le fait de préoccupations de la sécurité. On peut par exemple exiger que le flot F permette l'acheminement d'un certains types de flot f par au moins deux chemins arc disjoints. La contrainte (ii) décrit la relation de couplage entre deux types de flots F et f. Ce couplage s'écrit comme : $\sum_{i=1,..,n} f_i^e \leq \alpha F^e \ \forall e \in E$ où α le coefficient associé au flot F (par exemple, la capacité). La contrainte (iii) définit le sens d'écoulement de multi-flot f. La contrainte (iv) donne la fonction objectif du CFA. Le $C(F)$ est le coût d'installation du flot d'infrastructure F, et le $P(f)$ est liée à la qualité des routages de multi flot f sur un flot d'infrastructure F donné (par exemple, une pénalité liée à l'utilisation de certaines arcs de G).

La famille de problèmes de synthèse de réseaux contient nombreuse extensions du CFA : les problèmes de localisation si la topologie ou le positionnement du réseau ne son pas complètement connue [Coo63, Khu72, Cho94, Dre95, Jai97, Cha99] ; les problèmes de routage si le flot d'infrastructure est fixé [Eco91, Gav92, Bal95, Ouo00, Ben01] ; les problèmes avec réseaux dynamiques ou temporisés [Yag73, Aro89, Char96]. Ces problèmes restent difficiles à résoudre pour des raisons comme : la taille de réseaux sont grande ; ils admettent plusieurs optimisions locales ; la mise en forme des contraintes sur le flot d'infrastructure est risque d'être très complexe. Pour plus de références sur la « synthèse de réseaux », on pourra citer [Ben03, Quil05, Yon05].

1.3.2 Principaux outils pour les problèmes de flots

Dans cette section, on va présenter les méthodes principales pour la résolution des problèmes de synthèse de réseaux. On distingue ces méthodes en deux catégories : (i) méthodes de décomposition en programmation linéaire et polyèdres ; (ii) réseaux de files d'attente et modèles de théorie des jeux.

1.3.2.1 Méthodes de décomposition en programmation linéaire et polyèdres

Un problème de synthèse de réseaux est souvent utilisé pour répondre au besoin des systèmes de grande taille, incluant à la fois des variables fractionnaires et des variables entières. Il est modélisé sous la forme de programmation linéaire et résolu à l'aide des bibliothèques de

programmation linéaire comme CPLEX, Gurobi, XPRESS ou encore GLPK. Dans cette section, nous avons fait une étude sur les principales méthodes de décomposition.

(a) Décomposition de Benders

La décomposition de Benders [Ben62, Geo72] est une méthode de décomposition de type *Maitre/Esclave* pour des problèmes d'optimisation en programmation linéaire avec des variables mixtes. Dans cette méthode, la partie *Maitre* s'occupe de la génération de la topologie du réseau. Plus concrètement, elle traite le flot d'infrastructure (exemple : les parcours de véhicules dans les problèmes de tournées de véhicules) et la partie *Esclave* est utilisée pour l'écoulement des flots objets sur la topologie construite (exemple : objets à transporter dans les problèmes de tournées de véhicules). Dans un modèle linéaire d'un problème de synthèse de réseaux, on distingue les variables en deux groupes : (i) vecteur F : variables entières associées au flot d'infrastructure ; (ii) vecteur f : variables fractionnaires correspondant aux flots d'objets. L'idée principale de cette méthode consiste à décomposer un problème en deux sous-problèmes (partie *maître* et partie *esclave*) en utilisant la théorie de dualité de la programmation linéaire. Chaque sous-problème ne contient qu'un groupe de variables à déterminer. La résolution de chaque sous-problème offre une borne (borne supérieure pour la partie *maître* et borne inférieure pour la partie *esclave*) à la solution optimale du problème initial. On répète alternativement la résolution de ces deux sous-problèmes. Le processus se termine quand la différence entre les deux bornes est à nulle.

On suppose que la formulation ($PL\ 5$) est un modèle linéaire du problème à résoudre, où c et d sont des vecteurs de coût, A et B sont des matrices de coefficients, b est un vecteur et Y est un sous-ensemble d'entiers.

$$(PL\ 5)\begin{cases} \text{Minimiser } cF + df & (5.1) \\ \text{s.c.} & \\ \quad AF + Bf \geq b & (5.2) \\ \quad F \in Y & (5.3) \\ \quad f \geq 0 & (5.4) \end{cases}$$

La formulation ($PR\ 5$) montre le sous-problème lié à ($PR\ 5$) en fixant les valeurs de F, dit *Problème Restreint*. Le dual de ($PR\ 5$) est noté ($DPR\ 5$) avec le vecteur de variables dual u. Le ($DPR\ 5$) est considéré comme le *problème esclave*.

$$(PR\ 5)\begin{cases} \text{Minimiser } df & (5.5) \\ \text{s.c.} & \\ \quad Bf \geq b - AF & (5.6) \\ \quad f \geq 0 & \end{cases} \qquad (DPR\ 5)\begin{cases} \text{Maximiser } (b - AF)u & (5.7) \\ \text{s.c.} & \\ \quad B^T u \leq d & (5.8) \\ \quad u \geq 0 & (5.9) \end{cases}$$

La transformation pour obtenir le *problème maitre* est un peu plus compliquée. On introduit d'abord la formulation ($PR\ 5$) dans ($PR\ 5$). La formulation ($PR\ 5$) devient :

$$\min_{F \in Y}\{cF + \min\{df \mid Bf \geq b - AF, f \geq 0\}\} \qquad (5.10)$$

Selon la théorie de la dualité de la programmation linéaire, la valeur optimale du problème primaire et égale à celle du problème dual. En appliquant cette théorie entre (*PR 5*) et (*DPR 5*), on obtient alors l'équation suivante :

$$\min\{df \mid Bf \geq b - AF, f \geq 0\} = \max\{(b - AF)u \mid B^T u \leq d, u \geq 0\} \qquad (5.11)$$

On introduit l'équation (5.11) en (5.10), cela nous donne :

$$\min_{F \in Y}\{cF + \max\{(b - AF)u \mid B^T u \leq d, u \geq 0\}\} \qquad (5.12)$$

On considère Z l'espace des solutions faisables du problème initial, P l'ensemble des points extrêmes de Z, et K l'ensemble des rayons extrêmes (contraintes bornant l'espace de solutions) de Z. Pour les valeurs F données, le *problème esclave* (*DPR 5*) est facile à résoudre. La résolution obtenue u^* est soit un point extrême u^p de l'espace des solutions/polyèdre convexe, soit un rayon extrême u^k quand le problème est non-borné, avec $u^p \in P$ et $u^k \in K$.

Dans le cas où le *problème esclave* est non-borné, la formulation (5.12) n'a pas de solutions faisables. On doit réduire le domaine compatible de F afin de donner une solution faisable à (5.12). Ce domaine compatible est définit par :

$$R = \{F \mid (b - AF)u^k \leq 0, F \in Y, \forall u^k \in K\} \qquad (5.13)$$

La formulation (5.12) est devenu (5.14) en remplaçant Y par R :

$$\min_{F \in R}\{cF + \max\{(b - AF)u \mid B^T u \leq d, u \geq 0\}\} \qquad (5.14)$$

Dans le cas où le *problème esclave* est borné, on ne considère que les points extrêmes u^p correspondant à la solution obtenue. La formulation (5.12) peut être écrite comme :

$$\min_{F \in R}\{z \mid z \geq cF + (b - AF)u^p, \forall u^p \in P\} \qquad (5.15)$$

Le *problème maître* (*PM 5*) est donc obtenu en combinant les deux formulations (5.13) et (5.15). Il contient donc deux groupes de contraintes : contraintes de type (5.17) et contraintes de type (5.18). Les contraintes de ces deux groupes associent respectivement à la solution du *problème esclave* (l'ensemble des points extrêmes P et l'ensemble des rayons extrêmes K). Le *problème maître* ne contient aucune contrainte de type (5.17) et de type (5.18) au départ. On ajoute une contrainte à chaque itération selon la solution du *problème esclave* obtenue.

$$(PM\ 5)\begin{cases} \text{Minimiser } z & & (5.16) \\ \text{s.c.} & & \\ \quad z \geq cF + (b - AF)u^p & \forall u^p \in P & (5.17) \\ \quad (b - AF)u^k \leq 0 & \forall u^k \in K & (5.18) \\ \quad F \in Y & & \end{cases}$$

L'algorithme 1.6 montre le schéma principal de décomposition de Bender. Dans une itération de la phase (ii), on résout le *problème esclave* (*DPR 5*) avec les valeurs actuelles de *F'* : si la (*DPR 5*) est bornée, alors une contrainte de type (5.17) est ajoutée dans le (*PM 5*) avec u^* la solution optimale du problème esclave obtenue, *i.e.* $P = P \cup \{u^*\}$. La borne supérieure UB du problème initial est mise à jour avec la valeur optimale de (*DPR 5*), et la borne inférieure

LB est égale à valeur optimale de ($PM\,5$) actuel ; si ($DPR\,5$) n'est pas bornée, alors une contrainte de type (5.18) est ajoutée dans ($PM\,5$), *i.e.* $K = K \cup \{u'\}$, où $u' = \max\{(b - AF')u \mid BT\,u \leq d, (b - AF')u \leq 0, u \geq 0\}$. Les itérations se terminent quand la différence entre la borne supérieure et la borne inférieure est plus petite que la tolérance ε, *i.e.* quand $UB - LB \leq \varepsilon$.

Algorithme 1.6 : Décomposition de Benders

(i) phase d'initialisation

$F \longleftarrow$ une solution faisable du flot de véhicules ;
$\varepsilon \longleftarrow$ tolérance ;
$UB \longleftarrow +$;
$LB \longleftarrow -\infty$;
$k \longleftarrow 0$;
$p \longleftarrow 0$;

(ii) phase d'amélioration itérative

Tant que UB – LB > ε **faire**
| Résoudre le problème esclave (*DPR 5*) avec *F* donné ;
| **Si** le problème esclave (*DPR 5*) est infaisable **alors**
| | Retourner faux;
| **Sinon Si** le problème esclave (*DPR 5*) est borné **alors**
| | $u^* \longleftarrow$ solution optimale de (*DPR 5*) ;
| | $UB \longleftarrow$ valeur optimale de (*DPR5*) ;
| | *// ajouter un point extrême dans l'espace de solutions*
| | Mettre à jour (*PM 5*) en ajoutant une contrainte (*5.17*) avec u^* ;
| | $p \longleftarrow p + 1$;
| | Optimiser le problème maître (*PM5*) ;
| | $F \longleftarrow$ solution optimale de (*PM5*) ;
| | $LB \longleftarrow$ valeur optimale de (*PM5*) ;
| **Sinon**
| | Déterminer u' avec *F* donné ;
| | *//ajouter un rayon extrême dans l'espace de solutions*
| | Mettre à jour (*PM 5*) en ajoutant une contrainte (*5.18*) avec u' ;
| | Optimiser le problème maître (*PM 5*) ;
| | $F \longleftarrow$ solution optimale de (*PM 5*) ;
| **Fin Si**
Fin Tant que
Retourner *F* ;

(b) Décomposition Lagrangienne

La décomposition lagrangienne est souvent appliquée pour des formulations qui contiennent une conjonction de deux blocs de contraintes. Elle crée d'abord une copie des variables, et puis sépare le problème initial en deux sous-problèmes en relaxant des contraintes de couplage liées à deux copies de variables. Les contraintes relaxées sont ajoutées dans la fonction objectif à l'aide d'un multiplicateur Lagrangien. On considère que ($PL\,6$) est la formulation en programmation linéaire à résoudre, où c est un vecteur de coûts, A et B sont des matrices de coefficients, a et b sont des vecteurs. ($PL'6$) est la formulation après la duplication des variables.

$$(PL\,6)\begin{cases}\text{Minimiser } cx & (6.1)\\ \text{s.c.}\\ \quad Ax \le a & (6.2)\\ \quad Bx \le b & (6.3)\\ \quad x \in N^n & (6.4)\end{cases} \qquad (PL'\,6)\begin{cases}\text{Minimiser } cx\\ \text{s.c.}\\ \quad Ax \le a\\ \quad By \le b\\ \quad x = y & (6.6)\\ \quad x \in N^n, y \in N^n & (6.7)\end{cases}$$

On introduit le multiplicateur $\mathbb{R}^n$ pour relaxer les contraintes de couplage des deux copies de contraintes (6.6). La formulation lagrangienne est définie :

$$L_{xy}(\lambda) = \min\{cx + \lambda(y - x) \mid Ax \le a, By \le b, x \in N^n, y \in N^n\} \qquad (6.8)$$

La (6.8) peut aussi s'écrire comme :

$$L_{xy}(\lambda) = \min\{(c - \lambda)x + \lambda y \mid Ax \le a, By \le b, x \in N^n, y \in N^n\} \qquad (6.9)$$

On décompose (6.9) en deux sous-problèmes avec $L_{xy}(\ ) = L_x(\ ) + L_y(\ )$:

$$L_x(\lambda) = \min\{(c - \lambda)x \mid Ax \le a, x \in N^n\} \qquad (6.10)$$
$$L_y(\lambda) = \min\{\lambda y \mid By \le b, y \in N^n\} \qquad (6.11)$$

Le dual de la décomposition lagrangienne est défini comme :

$$(DDL\,6)\begin{cases}\text{Maximiser } L_{xy}(\lambda) & (6.12)\\ \text{s.c.}\\ \quad \lambda \in R^{n+} & (613)\end{cases}$$

On considère que $Z_x = \{x \ge 0 \mid Ax \le a\}$ est le polyèdre convexe qui représente l'ensemble des solutions de x, et $Z_y = \{y \ge 0 \mid By \le b\}$ est le polyèdre pour y. Soit K l'ensemble des points extrêmes du polyèdre Z_x, et L est l'ensemble des points extrêmes pour Z_y, $k = |K|$ et $l = |L|$. Alors, les deux formulations (6.10) et (6.11) deviennent :

$$L_x(\lambda) = \max\{w_1 \mid w_1 \le (c - \lambda)x, \forall x \in Z_x\} \qquad (6.14)$$
$$= \max\{w_1 \mid w_1 \le (c - \lambda)x_i, i = 1, .., k\} \qquad (6.15)$$

$$L_y(\lambda) = \max\{w_2 \mid w_2 \le \lambda y, \forall y \in Z_y\} \qquad (6.16)$$
$$= \max\{w_2 \mid w_2 \le \lambda y_j, j = 1, .., l\} \qquad (6.17)$$

La formulation ($DDL'6$) est obtenue en considérant $w = w_1 + w_2$, avec les formulations (6.15) et (6.17). La valeur optimale de ($DDL\,6$) est égale à la valeur optimale de ($DDL'6$) :

$$(DDL'\,6)\begin{cases}\text{Maximiser } w & & (6.18)\\ \text{s.c.}\\ \quad w + \lambda(x_i - y_j) \ge cx_i & \forall i = 1, .., k, j = 1, .., l & (6.19)\\ \quad \lambda \in R^{n+} & & (6.20)\\ \quad w \ge 0 & & (6.21)\end{cases}$$

La solution de ($DDL'6$) donne une borne inférieure de bonne qualité à la solution optimale du problème initial. Mais, la résolution de cette formulation est coûteuse en temps de calcul. Pour

plus de détails concernant cette méthode, on peut se référer aux travaux de [Geo74, Das89, Tou07].

(c) Décomposition de Dantzig-Wolfe

La décomposition de Dantzig-Wolfe est utilisée pour des programmations linéaires qui peuvent être séparées en plusieurs sous-problèmes indépendants. Ces sous-problèmes sont articulés par un petit nombre de contraintes de couplage. On formule le problème maître en utilisant des variables liées au terme de points extrêmes. Le problème maître est souvent de très grande taille, avec un grand nombre de colonnes. Il est résolu par la génération de colonnes (voir la section 1.2.1.3). On suppose que le ($PL\ 7$) est la formulation à résoudre, dans laquelle les x_i sont des vecteurs de variables, C_i sont des vecteurs de coûts, A_i et D_i sont des matrices de données, b_j sont des vecteur de données avec $i = 1..t$ et $j = 0..t$.

$$(PL\ 7)\begin{cases} \text{Minimiser} & C_1x_1 & + & C_2x_2 & + & \dots & + & C_tx_t & & & \\ \text{s.c.} & & & & & & & & & & \\ & A_1x_1 & & & & \cdots & & & = & b_1 & (7.1) \\ & & & A_2x_2 & & & & & = & b_2 & (7.2) \\ & \vdots & & & & \ddots & & & = & \vdots & \vdots \\ & & & & & & & A_tx_t & = & b_t & (7.t) \\ & D_1x_1 & + & D_2x_2 & + & \cdots & + & D_tx_t & = & b_0 & (7.0) \\ & x_1 & , & x_2 & , & \dots & , & x_t & \geq & 0 & \end{cases}$$

Dans le ($PL\ 7$), les contraintes (7.0) correspondent aux contraintes de couplage, et les contraintes (ou blocs) de (7.1) à (7.t) peuvent être décomposées en sous-problèmes. Un sous-problème i est défini comme :

$$(\text{PLS7} - \text{i})\begin{cases} \text{Minimiser } C_ix_i \\ \text{s.c.} \\ \quad A_ix_i = b_i \\ \quad X_i \geq 0 \end{cases}$$

La résolution d'un sous-problème i est relativement simple. La solution obtenue est soit un point extrême (noté x_i^j) du domaine réalisable pour le sous-problème ; soit un rayon extrême (noté w_i^k) quand le sous-problème n'est pas borné. On considère Z_i l'ensemble des solutions réalisables pour le sous-problème i, P_i l'ensemble des points extrêmes et K_i l'ensemble des rayons extrêmes. Une solution x_i peut être écrite comme (7.t+1) avec des variables θ_i^k et λ_i^k où $\theta_i^k \geq 0$ et$\lambda_i^k \geq 0$, et la contrainte de convexité $\sum_{j\in P_i} \lambda_i^j = 1$

$$x_i = \sum_{j\in P_i} \lambda_i^j x_i^j + \sum_{k\in K_i} \theta_i^k w_i^k \qquad (7.t+1)$$

Le problème maître ($PLM\ 7$) utilise la fonction objectif et les contraintes de couplage de ($PL\ 7$). Dans ces contraintes de couplage, les vecteurs de variables x_i sont remplacés par l'équation (7.t+1). On introduit aussi des contraintes de convexité. Le problème maître est défini comme :

$$(PLM\ 7)\begin{cases} \text{Min} & \sum_{j\in P_1} C_1 x_1^j \lambda_1^j + \sum_{k\in K_1} C_1 w_1^k \theta_1^k + \cdots + \sum_{j\in P_t} C_t x_t^j \lambda_t^j + \sum_{k\in K_t} C_t w_t^k \theta_t^k & & & \\ \text{s.c.} & & & & \\ & \sum_{j\in P_1} D_1 x_1^j \lambda_1^j + \sum_{k\in K_1} D_1 w_1^k \theta_1^k + \cdots + \sum_{j\in P_t} D_t x_t^j \lambda_t^j + \sum_{k\in K_t} D_t w_t^k \theta_t^k & = & b_0 & \\ & \sum_{j\in P_i} \lambda_i^j & = & 1 & \forall i = 1..t \\ & \lambda_i^j & \geq & 0 & \forall i,j \\ & w_i^k & \geq & 0 & \forall i,j \end{cases}$$

On résout le problème maître par la génération de colonnes. Dans une itération, π et π' sont les variables duales correspondant à la solution optimale du «Problème Principal Restreint» (PPR) actuel du problème maître, où π correspond aux contraintes de couplage et π' aux contraintes de convexité. Le sous-problème dans la génération de colonnes consiste à identifier la colonne $i \quad T/S$ (où T est l'ensemble des vecteurs de variables et S l'ensemble dans le PPR actuel), qui possède le plus petit coût réduit C^* :

$$C^* = \min_{i\in T/S}\{\min\{(C_i - \pi D_i)x_i \mid A_i x_i = b_i, x_i \geq 0\} - \pi_i'\}$$

Si la valeur de C^* est négative, alors on ajoute la colonne correspondant dans le PPR actuel et on itère. Sinon, la solution actuelle est optimale. Pour plus de détails sur la décomposition de Dantzig-Wolfe, on peut se référer à [Dan60, Las70, Ahu98, Las02].

(d) Autres méthodes

Il existe d'autres méthodes comme la décomposition proximale et les méthodes polyédrales. La décomposition proximale peut être vue comme une méthode combinant la régularisation et la relaxation. Elle est utilisée souvent pour les problèmes de routage non linéaires qui peuvent être décomposés naturellement en une famille de sous-problèmes indépendants reliés par une contrainte de couplage. Pour plus de détails sur cette méthode, on peut se référer à [Spi85, Chi94, Mah98].

Les méthodes basées sur des techniques polyédrales sont aussi appelé méthode de type branchement et coupes (voir la section 1.2.1.1). Un polyèdre *conv(X)* est l'enveloppe convexe de l'ensemble des points x dans l'espace des solutions X. Chaque point $x \quad R^n$ représente une solution réalisable. L'idée principale est d'amener la résolution d'une formulation linéaire à l'optimisation d'une fonction linéaire sur le polyèdre associé. Ce type d'approche a été introduit initialement par *J. Edmonds*. Il l'applique à un problème de couplage (de graphe). Pour plus de détails sur ce type de méthode, on peut consulter les études de [Gro90, Dah94, Cha05, Mah05].

1.3.2.2 Réseaux de files d'attente et modèles de la théorie des jeux

La plupart des problèmes de synthèse de réseaux consistent en un traitement de multi-flot. Ce multi-flot représente souvent des flux moyens correspondant à la circulation d'objets dynamiques comme : messages, véhicules et signaux audio. Les modèles sont donc stratégiques ou tactiques associés à des opérations du système observées sur une période assez

longue. Ils demandent un système dynamique qui permet d'analyser des indices de performance d'un modèle comme la qualité de service. Dans la plupart des cas, un tel système peut être modélisé comme un réseau de files d'attente et être simulé par des logiciels de type QNAP2. Il prend en compte ainsi l'obtention de mesures de délais, taux de perte ou de congestion. Dans un réseau de télécommunication, la fonction de *délai de Kleinrock* est une des fonctions les plus connues pour la mesure de ces taux de congestion. Pour avoir plus de références sur l'application des réseaux de files d'attente dans la synthèse de réseaux, on peut citer [Kle75, Sch77, Ash98, Reb00].

Des modèles de la théorie des jeux sont utilisés comme des outils pour des problèmes de synthèse de réseaux quand ils s'agissent de la tarification des réseaux. On les distingue en deux classes : (i) les jeux coopératifs ; (ii) les jeux non coopératifs. Dans un modèle de jeux coopératifs, la tarification correspond à un processus d'*imputation* (et de décision) des coûts de la part d'un opérateur. Un tel modèle forme des coalitions entre les opérateurs afin de réduire leur coût global. C'est un concept qui n'existe pas dans un modèle de jeux non coopératifs. Les modèles de jeux non coopératifs utilisent un processus naturel d'ajustement des prix et des niveaux de production par des opérateurs concurrents confrontés à une demande élastique aux prix. Ce type de modèles est donc moins une optique décisionnelle que prévisionnelle. Ils peuvent être résolus en recherchant des stratégies optimales de type *équilibre de Nash*, qui correspond à une situation telle qu'aucun des producteurs ne peut modifier sa décision sans voir se dégrader son profit. Pour plus de détails, on peut citer [Nas51, Ber57, Bon63, Arr75, Gui95, Kos05].

1.4 Conclusion

Dans ce chapitre, nous avons réalisé un état de l'art sur les problèmes d'optimisation combinatoire autour de la mobilité et du transport et nous avons passé en revue les différentes méthodes de résolution. Ce chapitre se compose de trois parties : la première partie concerne la présentation des problèmes de tournées de véhicules classiques (VRP), des problèmes de collecte et livraison, ainsi que les principales variantes. Dans la seconde partie, nous avons étudié deux types de méthodes de résolution des problèmes de tournées de véhicules : les méthodes exactes telles que programmation linéaire, les méthodes de recherche arborescente ou encore des méthodes de programmation dynamique, qui permettent de trouver des solutions optimales mais avec un temps de calcul important. Les méthodes approchées comme des heuristiques ou des méta-heuristiques permettent de trouver des solutions rapidement mais sans garantie d'optimalité. Dans la dernière partie, une application des problèmes de multi-flots sur le transport a été proposée. Nous avons présenté aussi les principaux algorithmes de résolution des problèmes de flots.

Chapitre 2

Problème de la Planification du Redéploiement de Véhicules partagés : définition et modèle

Sommaire

2.1 Contexte du problème 63
2.1.1 Problème du Plan de Redéploiement (PPR) 64
2.1.2 PPRV : la version de base des problèmes de type PPR 67
2.1.3 Autres problèmes de type PPR 67
2.2 Définition du PPRV 69
2.3 Aperçu des problèmes proches du PPRV 70
2.4 Modélisation mathématique du PPRV 72
2.4.1 Données d'entrée et paramètres 72
2.4.2 Variables 72
2.4.3 Contraintes 72
2.4.4 Fonction objectif 73
2.4.5 Modèle mathématique du PPRV 73
2.4.6 Contraintes d'élimination de cycles 74
2.5 Expérimentations 77
2.5.1 Environnements de l'expérimentation et instances utilisées 78
2.5.2 Comparaison des contraintes d'élimination de cycles 78
2.5.3 Résultat pour le PPRV 79
2.6 Conclusion 82

Ce chapitre aborde le « Problème de la Planification du Redéploiement de Véhicules partagés » (PPRV), qui est considéré comme un problème de collecte et de livraison de type *Many-to-Many* à plusieurs véhicules. Nous commençons par la présentation du contexte général avant de donner la définition du PPRV. Nous effectuons ensuite une étude bibliographique sur les problèmes apparentés au PPRV. Nous définissons aussi un modèle linéaire et fournissons des résultats expérimentaux à la fin du chapitre.

2.1 Contexte du problème

Nous nous intéressons aux problèmes liés à la gestion des systèmes de véhicules partagés du type location en libre service (au niveau d'outil décisionnel) comme les dispositifs *Vélib* à Paris ou *Vélo'v* à Lyon dans le cadre de la location de vélos, ou encore des projets de mise en partage de voitures électriques comme *Autolib* à Paris. Dans ce type de systèmes, les véhicules sont loués par des clients ou par des abonnés sur des sites implantés généralement dans le tissu urbain/périurbain. La location (ou la restitution) des véhicules évolue de manière non déterministe sur les sites. Un tel système permet à un client d'emprunter (de louer) un véhicule sur un site de services sans avoir besoin de réserver. Après l'avoir utilisé, le client

peut rendre le véhicule sur un site qui n'est pas forcément le même que lors de l'emprunt. En l'absence de politique de gestion proactive, le système conduit presque inévitablement à des déséquilibres de la disponibilité de véhicules sur les sites. De tels phénomènes, dits de « rupture », apparaissent et sont de deux types :

- rupture d'espace libre : un client souhaite louer un véhicule sur un site vide. Il est alors obligé d'attendre le retour d'un véhicule ou d'aller sur un autre site.
- rupture de disponibilité : un client souhaite rendre un véhicule sur un site qui n'a plus d'espace libre. Le client est obligé d'attendre la prise d'un véhicule ou bien d'aller sur un autre site.

Ces situations de rupture sont perçues négativement par les utilisateurs. Elles peuvent réduire le taux d'utilisation du système et, par conséquent, réduire son impact sur le transport urbain. Le Problème de Redéploiement de Convois (PRC) consiste à déterminer **quand** et **comment** effectuer la réorganisation des ressources, tout en respectant des critères associés, afin d'assurer la qualité de service perçue par les clients et le coût associé à la réorganisation. Il concerne donc deux niveaux de décision (quand et comment). Nous nous intéressons plus spécifiquement au deuxième niveau de décision, qui concerne la stratégie de redéploiement (niveau **comment**). Elle conduit au Problème du Plan de Redéploiement (PPR).

La Figure 2.1 illustre la hiérarchie des problèmes liés au redéploiement de convois pour un système de véhicules partagés. Le PPR est considéré comme un ensemble de problèmes d'optimisation combinatoire, qui contient essentiellement les 5 problèmes suivants : PPRV (Problème de la Planification du Redéploiement de Véhicules partagés), PPRV-SCI (PPRV sans Charge Initiale), PPRV-PM (PPRV avec Passage Multiple), PPRV-T (PPRV avec Transfert) et PPRV-D (PPRV Dynamique).

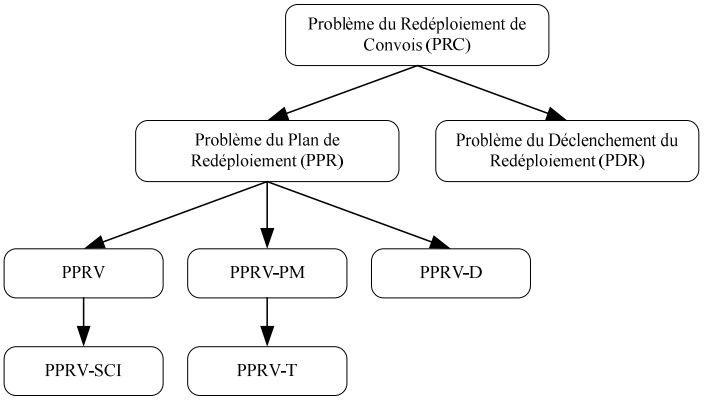


Figure 2.1 : Schéma hiérarchique des problèmes liés au redéploiement de convois

Dans cette section, nous commençons par la présentation du PPR. Ensuite, nous allons décrire les caractéristiques des 5 problèmes qui font partie du PPR. Parmi lesquels le PPRV est considéré comme la version de base.

2.1.1 Problème du Plan de Redéploiement (PPR)

Le PPR est défini comme suit : étant donné un réseau de transport sous la forme d'un graphe (souvent complet et non-orienté) $G = (V, A)$, où V est l'ensemble de sommets qui contient les sites de service et le dépôt des transporteurs et A est l'ensemble des arcs, chaque arc étant associé à un coût lié à la distance spatio-temporelle. Les informations liées à chaque site peuvent être représentées par un triplet (identifiant, quantité de véhicules présents, capacité).

Chaque site possède une situation d'équilibre qui peut être définie comme le nombre souhaitable de véhicules disponibles. Ce nombre peut être dérivé d'une analyse probabiliste des déplacements dans le temps et dans l'espace des usagers et de l'état de chaque site. La demande est obtenue en mesurant la différence entre le nombre souhaité et la quantité actuelle pour chaque site. Un site est en situation « équilibre » si la demande est nulle. Un ensemble de transporteurs est disponible sur le dépôt central. L'objectif consiste à planifier les opérations de redéploiement des véhicules locatifs à l'aide des transporteurs afin de restaurer la situation d'équilibre pour chaque site, tout en optimisant le critère de performance sous le respect des contraintes liées.

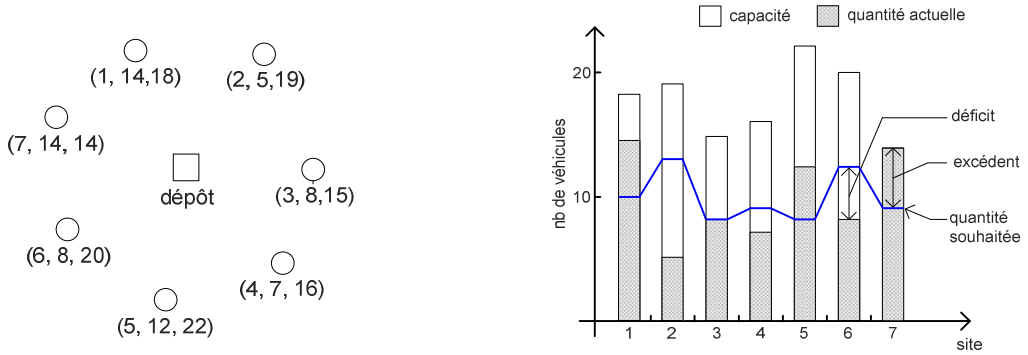
Nous présentons ensuite la description des différents attributs (caractéristiques) autour d'un problème du plan de redéploiement :

- **Caractéristique des objets à transporter**

Les objets à transporter sont des véhicules (par exemple des vélos). Ils sont tous homogènes, sans identification précise ni site (propriétaire) dédié. Aucune continuation n'existe pour ces objets. Avec une telle caractéristique, les objets collectés sur un site (avec une demande positive) pourraient être utilisés pour répondre à la demande de plusieurs sites de livraison (avec demande négative). Au moment de la collecte d'un objet, on ne précise pas sa destination.

- **Estimation de la demande**

La demande d'un site décrit la différence entre le nombre souhaité (situation d'équilibre) et la quantité présente. La Figure 2.2 donne un exemple de l'estimation de la demande. La Figure 2.2 (a) illustre un système locatif sous la forme graphe (complet). Le carré central est le dépôt et chaque cercle représente un site de service. Le libellé en triplet indique les informations d'un site (identifiant, quantité présent, capacité). La figure (b) montre un processus d'estimation de la demande des sites. Nous avons par exemple les sites 1, 5 et 7 en situation d'excédent, les sites 2, 4 et 6 en situation de déficit, et le site 3 en situation d'équilibre. Les sites en situation d'excédent (de déficit) sont dits sites de collecte (livraison). Notons que l'estimation de la demande peut éventuellement tenir compte de la capacité associée aux transporteurs dans certains cas, ainsi que de l'estimation de l'évolution à court/moyen terme.



(a) représentation d'un système locatif (b) principe d'équilibrage des sites

Figure 2.2 : Estimation de la demande

- **Transporteur (agent de transport)**

Nous désignons ici un transporteur par un « agent ». Les agents jouent le rôle opérationnel dans le PPR. Ils réalisent le déplacement des objets (véhicules locatifs) en effectuant des opérations (collecte ou livraison) sur les sites selon la demande. Dans un système locatif de vélos, un agent est associé à une camionnette. Dans le cas de véhicules électriques, l'agent

effectue des opérations de convoyage en conduisant le premier véhicule du convoi. La capacité d'un agent se traduit alors en capacité de la camionnette pour le transport de vélos et en taille maximum du convoi pour le déplacement des véhicules électriques. Nous considérons que les agents sont tous homogènes. C'est-à-dire que les camionnettes sont identiques. Un agent est affecté au traitement d'un certain nombre de sites. Il commence et termine son planning de transport. La trajectoire du déplacement d'un agent forme une tournée.

- **Coût associé à un arc**

Dans un problème PPR, le coût associé à un arc se traduit par une longueur L ou/et par un coût C, qui satisfont tous deux à l'intégralité triangulaire. On considère que le temps nécessaire à un agent pour aller de i à j et effectuer une opération de chargement ou déchargement en j est égale à $L(i,j)$. On considère aussi que le coût associé à un tel déplacement est fixe et égal à $C(i,j)$. On peut aussi introduire une définition du coût plus précise selon les différentes variantes du PPR, comme :

- une fonction de coût spécifique (non linéaire) selon le nombre d'agents employés ;
- des coûts spécifiques liés aux échanges d'objets dans une extension du PPR avec transfert d'objets ;
- des variations des coûts en fonction du temps.

- **Critères d'optimisation liés à l'aspect dynamique**

Dans le PPR, on cherche à retrouver la situation d'équilibre sur chaque site, le terme « équilibre » traduit le **besoin opérationnel** d'un système (locatif). Sachant que dans un système réel, le stockage des sites évolue en fonction du temps, le besoin d'équilibre peut être traduit différemment selon le rythme d'évolution (du stockage des sites) : si l'évolution est relativement lente, on cherche alors un équilibrage pour tous les sites ; si l'évolution est très rapide, alors le besoin d'équilibre concerne essentiellement les sites liés aux phénomènes de « rupture ». Dans ce cas, on peut ne traiter qu'une partie des sites (souvent ceux qui ont les plus grandes demandes). La sélection des sites peut être faite aussi en fonction de la vitesse décroissante d'évolution de la quantité d'objets sur des sites ou encore par un processus d'estimation de la tendance d'évolution de cette quantité. Nous définissons ici le « taux de traitement » comme le pourcentage de sites qui vont être traités selon la situation du système.

Le critère d'optimisation d'un problème du plan de redéploiement considère essentiellement deux éléments : le taux de traitement et le coût de redéploiement. Ce dernier peut être par exemple la durée de redéploiement (la durée de trajectoire la plus longue parmi tous les agents) ou le temps total de redéploiement des objets par les agents. On peut représenter un problème bi-critère dans un repère où chaque axe est associé à un critère. Ainsi dans la Figure 2.3 les abscisses représentent le coût de redéploiement (à minimiser) et les ordonnées le taux de traitement (à maximiser). Chaque solution réalisable correspond à un point.

En multicritère, la dominance permet d'établir une relation d'ordre entre solution. Ici, une solution s_1 domine une solution s_2 si le coût de s_2 est moins bon pour un taux au plus égal. Sur le graphe, l'ensemble des solutions dominées par S correspond au rectangle dont le coin supérieur gauche est S.

La frontière (ou front) de Pareto est l'ensemble des solutions non dominées. Ce sont les solutions que l'on recherche, le choix final dépendant du compromis entre les deux critères. Ainsi S' s'indique la solution de meilleur coût si l'on souhaite traiter toutes les demandes.

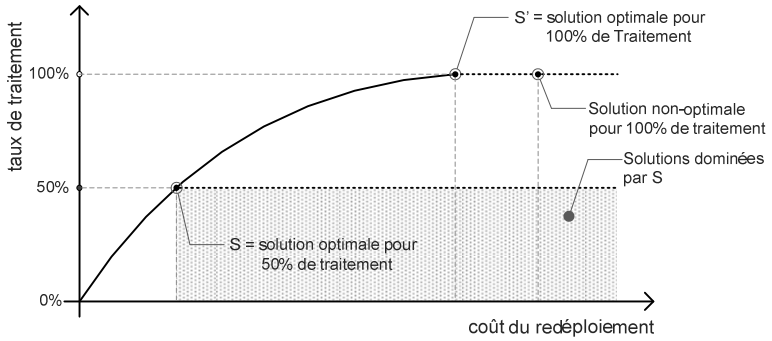


Figure 2.3 : Deux critères dans l'objectif d'un PPR

- **Contraintes générales**

Le problème du plan de redéploiement peut intégrer plusieurs contraintes selon le besoin lié à un contexte précis. Par exemple, le nombre d'agents disponible pour le redéploiement, la durée maximum des trajectoires des agents, l'autorisation du passage multiple sur un site de services, la charge initiale d'un agent lorsqu'il part du dépôt ou encore le transfert des objets entre les agents. La présence des différentes contraintes génère autant de variante du PPR.

2.1.2 PPRV : la version de base des problèmes de type PPR

Dans le cadre de cette thèse, le problème de la planification du redéploiement de véhicules partagés (PPRV) est considéré comme la version de base des problèmes du plan de redéploiement. Le PPRV est posé dans un contexte statique, en supposant que l'information concernant la demande est connue à l'avance et n'évolue pas significativement en fonction du temps durant le redéploiement. Le PPRV vise à restaurer l'équilibre pour tous les sites (*i.e.* 100% du taux de traitement). Le critère d'optimisation du PPRV correspond donc à la recherche de la solution S′ sur la Figure 2.3.

Nous ciblons plus particulièrement le PPRV sur la gestion des systèmes de location de vélos en libre service. Un agent correspond ici à une camionnette capable de transporter un lot de vélos. On fait l'hypothèse que l'estimation de la demande respecte la capacité des agents dans le PPRV. Cela veut dire que la demande estimée ne doit pas dépasser la capacité de camionnettes. Sous une telle hypothèse, le PPRV fait partie des problèmes de collecte et livraison (PDP, *Pickup and Delivery Problem*) en reprenant la nomenclature vue dans le chapitre précédent. Plus précisément, il figure dans la classe des PDP de type *Many-to-Many* en tenant compte du caractère d'homogénéité lié aux objets.

2.1.3 Autres problèmes de type PPR

Les autres problèmes de type PPR sont considérés comme des extensions du PPRV prennent en compte divers facteurs potentiels dans un système de véhicules partagés, y compris surtout des cas difficiles à gérer par le PPRV. Par exemple, le déplacement en « répétitif » entre certains sites : comme l'exemple du quartier de Montmartre pour le *Vélib* à Paris. Les sites au pied de Montmartre sont saturés, alors que ceux qui sont en haut sont vides (car il est plus facile de descendre de Montmartre à vélo que d'y monter). Cela oblige plusieurs allers-retours d'agents pour corriger les déséquilibres de la disponibilité de ces sites. Chaque site est visité plusieurs fois par des agents (le même ou différents agents). Ce cas correspond donc à une extension du PPRV qui autorise le passage multiple sur les sites de service. En plus du

passage multiple, on peut éventuellement considérer les facteurs suivants : la charge initiale des agents en quittant le dépôt, le nombre d'agents envisagés et le transfert d'objets entre agent.

Selon les différents facteurs présentés, on a essentiellement les extensions suivantes pour le PPRV:

- PPRV Sans Charge Initiale (PPRV-SCI) : le dépôt ne possède pas de stock d'objets supplémentaires. Par conséquent, un agent part du dépôt à vide et y retourne à la fin avec une charge vide, *i.e.* la conservation d'objets liés à la demande des sites traités dans une trajectoire est totale. En tenant compte de la contrainte de mono-accès sur les sites de services, la version de PPRV-SCI avec plusieurs agents peut ne pas être réalisable (cela vient du fait qu'il n'est pas toujours possible de partitionner des sites en plusieurs tournées dont la somme des demandes de chaque tournée est nulle). Dans cette thèse, le PPRV-SCI traité est la version à un agent.

- PPRV avec Passage Multiple (PPRV-PM) : cette extension autorise le passage multiple sur les sites de service. Il peut être réalisé par le même agent ou par des agents différents. L'avantage du PPRV-PM est qu'il est toujours réalisable, quelque soit la capacité de l'agent et la possibilité ou non de charge initiale dans le cas à plusieurs agents. Dans le cadre de cette thèse, on s'intéresse particulièrement au PPRV-PM sans charge initiale. Le PPRV-PM que nous abordons après est donc une extension du PPRV sans charge initiale et avec passage multiple à multi-véhicule. Il sera traité dans le chapitre 4.

- PPRV avec Transfert (PPRV-T) : cette extension repose sur le PPRV-PM. En plus des caractères du PPRV-PM, elle autorise le transfert des objets entre différents agents. Le transfert des objets entre deux ou plusieurs agents se réalise par le biais d'un rendez-vous asynchrone. Il traduit en fait une collaboration plus forte et plus directe entre les agents : un agent dépose une partie de sa charge sur un site de service, et un ou plusieurs autre(s) agent(s) les récupère(nt) après. Ce(s) agent(s) doive(nt) attendre le passage du premier agent de décharge avant de repartir s'il(s) arrive(nt) avant lui. Cette extension sera abordée dans le chapitre 5.

- PPRV Dynamique (PPRV-D) : cette extension est destinée au cas où l'évolution de la disponibilité des sites est très forte (en temps réel ou quasi-réel) au cours du redéploiement. Dans cette extension, le taux de traitement des sites n'est plus assuré à 100%. Les agents commencent le redéploiement en suivant la planification initiale (basée sur la disponibilité initiale). Au cours du redéploiement, une mise à jour de la planification est déclenchée lorsque la disponibilité des sites a évolué et l'ancienne planification n'est plus capable d'équilibrer la disponibilité des sites. Au moment de la mise à jour de la planification, les agents restent immobilisés lorsqu'ils sont sur un site ou bien ils attendent sur le prochain site s'ils sont sur la route. La mise à jour de la planification tient compte aussi de la position des agents. Le redéploiement par le PPRV-D peut se réaliser de manière périodique.

2.2 Définition du PPRV

Le Problème de la Planification du Redéploiement de Véhicules partagés (PPRV) est défini sur un graphe orienté complet $G = (V, E)$ où $V = \{0, .., n\}$ est l'ensemble des sommets et $E = \{(i, j) | \forall i, j \in V, i \neq j\}$ est l'ensemble des arcs.

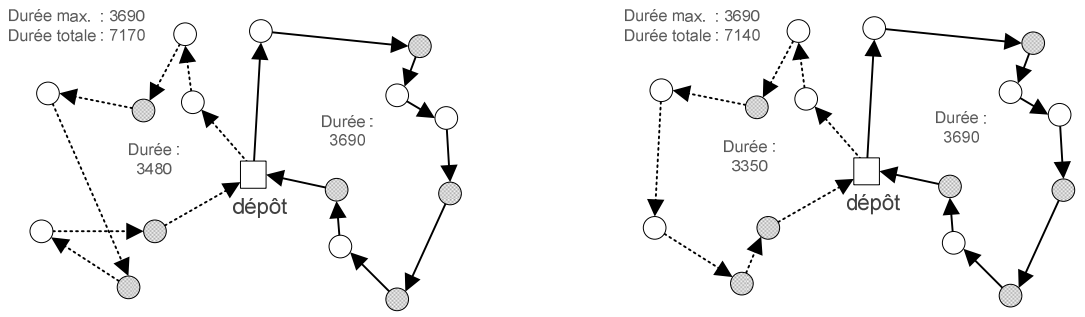
- L'ensemble des sommets est composé du dépôt central (numéroté 0) et des sites de service (numérotés de 1 à n). Chaque site de service i a une demande D_i qui traduit son besoin pour atteindre la situation d'équilibre : D_i est positive si le site i possède une quantité (de véhicules locatifs) en excès, elle est négative si ce site a besoin des véhicules et D_i est nulle s'il est déjà en équilibre. On fait l'hypothèse non restrictive que la somme des demandes est nulle. Si elle ne l'est pas, le dépôt peut compenser l'excédant ou le déficit global, *i.e.* $D_0 = -\sum_{i \in V \setminus \{0\}} D_i$.
- Un arc (i, j) représente le plus court chemin pour se rendre de i à j. On lui associe un coût de parcours T_{ij} qui peut être la distance spatio-temporelle entre deux sites ou encore le besoin en ressources (humaines ou la consommation de carburant) pour le transport de i vers j ou enfin un coût kilométrique.

Un ensemble de K agents homogènes (numérotés de 1 à K) de capacité Q est disponible pour le redéploiement. La capacité d'un agent est la capacité de la camionnette dans le PPRV. On fait l'hypothèse que la demande maximale respecte la capacité d'agents, *i.e.* $\max_{i \in V}\{|D_i|\} \leq Q$. Cette hypothèse est nécessaire pour éviter le passage multiple des agents sur les sommets. Nous rappelons que le passage multiple n'est pas autorisé dans le PPRV.

L'objectif est de déterminer la trajectoire des K agents, de façon à assurer le redéploiement. On cherche à optimiser les critères liés aux durées de redéploiement, en satisfaisant les contraintes suivantes :

(c2.1) une tournée commence et se termine au dépôt ;
(c2.2) un site est visité au plus une fois ;
(c2.3) la charge d'un transporteur ne dépasse pas sa capacité ;
(c2.4) toutes les demandes doivent être satisfaites.

Le PPRV utilise les deux critères d'optimisation suivants : la durée de redéploiement et la durée totale des tournées. Notons que la durée de redéploiement est définie par la plus longue tournée des agents (notée t_{max}). On souhaite que ce premier critère domine le deuxième dans le PPRV. La domination est définie par la présence d'un coefficient α où $0 \leq \alpha \leq 0.00001$ et $\alpha \in \mathbb{R}$. L'utilisation du deuxième critère permet d'éviter le cas sous-optimal : si l'on ne considère que le premier critère, alors une solution obtenue est « optimale » pour la plus grande tournée, alors que les autres tournées auraient encore pu être améliorées comme illustré en Figure 2.4. Le deuxième critère permet essentiellement de garantir que la durée des autres tournées serait aussi minimale.



(a) solution optimale à minimiser t_{max} (b) solution optimale à deux critères

Figure 2.4 : Influence sur les critères d'optimisation

Dans un schéma de redéploiement du PPRV, un agent part du dépôt avec une charge initiale d'objets, effectue une opération (collecte ou livraison) sur chaque site de sa tournée, avant de rentrer au dépôt avec une charge, qui peut être différente de celle du départ. Par contre, la somme des charges initiales des agents est égale à la somme des charges des agents au retour au dépôt, *i.e.* $\sum_{k=1,..,K} y_{0+}^k = \sum_{k=1,..,K} y_{0-}^k$ où y_{0+}^k (y_{0-}^k) est la charge de l'agent k en partant (rentrant) au dépôt. Cette somme est limitée par la valeur R correspondant au nombre d'objets disponibles au dépôt (pour la charge initiale des agents). On fait l'hypothèse que R est suffisamment grande pour permettre toutes les configurations initiales, *i.e.* $R = KQ$.

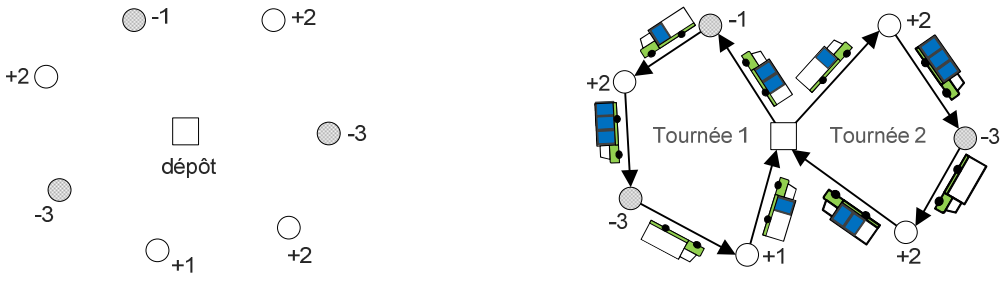
Remarque 2.1

Une tournée est réalisable si la différence entre le maximum et le minimum de charge de l'agent le long de la trajectoire ne dépasse pas sa capacité. Soit S la séquence des sommets de la tournée de l'agent k (commençant par le dépôt), $y_{S_i}^k$ *la charge de l'agent k après avoir visité le site* S_i. *En supposant que* $y_{S_0}^k = 0$ *et* $y_{S_i}^k = y_{S_{i-1}}^k + D_{S_i}$, *alors la tournée k est réalisable si* $\max_{i=0,..,|S|}\{y_{S_i}^k\} - \min_{i=0,..,|S|}\{y_{S_i}^k\} \leq Q$.

Remarque 2.2

Soit une tournée réalisable de l'agent k avec sa charge initiale au départ du dépôt y_{0+}^k *et sa charge finale à la rentrée au dépôt* y_{0-}^k. *Alors la trajectoire de sens inverse est aussi une tournée réalisable en gardant* $Q - y_{0-}^k$ *comme charge initiale et* $Q - y_{0+}^k$ *comme charge finale.*

La Figure 2.5 donne un exemple du PPRV. La figure (a) représente une instance à 8 sommets dont les sommets en gris (blanc) représentent les sites de livraison (collecte). La figure (b) montre une solution avec deux tournées. L'agent de la tournée 1 part du dépôt avec 2 objets chargés et l'agent de la tournée 2 par avec 1 objet. On note que parcourir chaque tournée en sens inverse avec la même charge initiale est aussi réalisable.



(a) une instance du PPRV (b) une solution du PPRV

Figure 2.5 : Exemple du PPRV

Le PPRV possède une caractéristique forte de problème de collecte et livraison (PDP). Il entre dans la catégorie *Many-to-Many* selon la classification proposée par Berbeglia *et al.* [Ber07] car les objets d'un même type sont indifférenciés. Le PPRV est un problème *NP-difficile*.

2.3 Aperçu des problèmes proches du PPRV

Les problèmes apparentés au PPRV concernent principalement les problèmes de collecte et livraison de type *Many-to-Many* (M-M). Le Tableau 2.1 montre la comparaison entre le

PPRV et les principaux problèmes de type M-M. Les colonnes de tableau sont définies comme suit :

1) *Nom* : abréviation/nom du problème ;
2) # *types* : nombre de types d'objets à transporter ;
3) # *agents* : nombre d'agents disponibles ;
4) *Hamiltonien* : oui si les tournées sont hamiltoniennes ;
5) *Multi-accès* : oui si le passage multiple d'agent sur un site est autorisé ;
6) *Charge initiale* : la charge de l'agent en quittant le dépôt. La valeur 0 signifie sans charge initiale ; 0/Q signifie avec charge vide ou charge pleine ; 0-Q pour une charge comprise entre 0 et Q.
7) *Références* : principaux articles associé.

La plus part des problèmes ont été déjà mentionnés dans la section 1.1.4.2. Parmi ces problèmes, le *Swapping Problem* (SP) est un PDP de type M-M avec plusieurs types d'objets à transporter. Le SP est introduit par Anily et Hassin [Ani92]. Il consiste à échanger des objets entre les sites en utilisant un agent de capacité unitaire. Un site possède à la fois une demande de collecte d'un type d'objets et une demande de livraison d'un autre type d'objets. Les demandes sont toutes unitaires dans ce problème. Le passage multiple d'agent sur un site est autorisé. Il existe aussi une version du SP préemptif [Ani92], dans lequel l'agent est autorisé à déposer sa charge temporairement sur un site, servir d'autres sites avant de retourner prendre la charge déposée pour terminer sa livraison.

Nom	# types	# agents	Hamiltonien	Multi-accès	Charge initiale	Références
SP	m	1	Non	Oui	0/Q	Anily *et al.* (1999), Anily *et al.* (1992)
CTSPPD	1	1	Non	Non	0/Q	Anily *et al.* (1999)
QDTSP	1	1	Non	Non	0/Q	Chalasani *et al.* (1999)
1-PDTSP	1	1	Oui	Non	0-Q	Hernandez-Pérez *et al.* (2004)
1-VRPPD	1	K	Oui	Non	0-Q	Martinovic *et al.* (2008)
PPRV	1	K	Oui	Non	0-Q	

Tableau 2.1 : Problèmes proches du PPRV

Le *capacitated TSP with pickups and deliveries* (CTSPPD) [Ani99], le *Q-delivery* [Cha99] TSP (QDTSP) sont des problèmes PDP de type M-M avec un seul type d'objets à transporter. Ces deux dénominations correspondent en fait au même problème. Les demandes sont unitaires et un seul agent de capacité limitée transporte les objets. Un site est considéré comme un client de collecte (livraison) si sa demande est positive (négative). L'agent ne passe pas sur les sites avec demande nulle. La tournée peut être non-hamiltonienne. Le passage multiple de l'agent sur des sites est interdit.

Le *one-commodity Pickup and Delivery TSP* (1-PDTSP) [Her04] est aussi un problème PDP de type M-M avec un type d'objets à transporter. Un agent de capacité limitée est disponible pour transporter des objets entre sites. Les demandes ne sont pas unitaires. Tous les sites doivent être visités dans ce problème. Le 1-PDTSP peut donc être considéré comme le PPRV avec un seul agent en ajoutant la contrainte liée à la tournée hamiltonienne.

Plus récemment, le *Single-commodity VRP with pickup and delivery* (1-VRPPD) [Mar08] est introduit par Martinovic *et al*. en 2008. Selon la définition de [Mar08], le 1-VRPPD est équivalent au 1-PDTSP dans le cas d'un seul agent. C'est-à-dire que tous les sites doivent être visités pour le 1-VRPPD. Le 1-VRPPD est un problème PDP de type M-M à multi-véhicules. Notons que le modèle linéaire dans [Mar08] ne concerne que le cas mono-véhicule, de même que la méthode proposée. Nous n'avons pas pu comparer les modèles linéaires entre le 1-VRPPD et le PPRV. Nous considérons le 1-VRPPD est cohérent au PPRV.

Nous proposons ici d'utiliser la dénomination *One-Commodity Pickup and Delivery* (1-PDP) pour définir l'ensemble des problèmes PDP de type M-M à un seul type d'objets à transporter. Dans ce cas, les problèmes avec la valeur 1 dans la colonne « # *types* » du Tableau 2.1 sont classé en 1-PDP.

Dans la littérature, la plupart des études sur les problèmes de type PDP M-M reste au niveau théorique. Il existe très peu d'applications concrètes à notre connaissance. Hernandez-Pérez *et al.* [Her07] ont donné un exemple de rééquilibrage financier entre les agences bancaires, dans lequel le type d'objets à transporter est de l'argent liquide. Hosny *et al.* [Hos10] propose une application liée au partage de médicaments entre des hôpitaux dans le contexte de certaines circonstances urgentes. Les médicaments à partager peuvent être, par exemple, des vaccins contre la grippe H1N1.

2.4 Modélisation mathématique du PPRV

Nous proposons ici une formulation linéaire pour le PPRV. Elle s'inspire du modèle du problème de collecte et livraison. Afin d'éviter la confusion sur la notation du dépôt (sommet numéro 0), nous utilisons 0^+ (0^-) pour décrire le dépôt lors du départ (retour) des agents.

2.4.1 Données d'entrée et paramètres

- $V = \{0, .., n\}$: ensemble des sommets du graphe ;
- $E = \{(i,j) | \forall i,j \in V, i \neq j\}$: ensemble des arcs du graphe ;
- $Q \in \mathbb{N}$: capacité de chaque agent;
- $K \in \mathbb{N}$: nombre d'agents ;
- $R \in \mathbb{N}$: quantité d'objets disponibles sur le dépôt (pour la charge initiale) ;
- $D_i \in [-Q, Q]$: demande associée au sommet i, $i \in V \backslash \{0\}$;
- $T_{ij} \in \mathbb{N}$: coût pour se rendre de i à j, $(i,j) \in E$;
- α : coefficient de pondération dans la fonction objectif.

2.4.2 Variables

- $x_{ij}^k \in \{0,1\}$: variables de décision pour indiquer si l'arc (i,j) est utilisé par l'agent k ;
- $y_i^k \in \mathbb{N}$: quantité d'objets transportés par l'agent k en quittant le site i, $i \in V \backslash \{0\} \cup \{0^+, 0^-\}$;
- $z_i^k \in \{0,1\}$: variables de décision pour décrire si le site i est visité par l'agent k ;
- $t_{max} \in \mathbb{N}$: durée de la plus longue tournée (durée du redéploiement).

2.4.3 Contraintes

- sur la durée de redéploiement t_{max}, qui est égale à la plus longue tournée :

$$\sum_{(i,j) \in E} T_{ij}^k x_{ij}^k - t_{max} \leq 0 \quad \forall k = 1 \dots K \qquad (2.1-1)$$

- sur la conservation du flot sur un site associé aux agents, *i.e.* un agent repart après avoir visité un site :

$$\sum_{j:(i,j) \in E} x_{ij}^k = \sum_{j:(j,i) \in E} x_{ji}^k = z_i^k \quad \forall i \in V \backslash \{0\}, k = 1 \dots K \qquad (2.1-2)$$

- sur le nombre d'accès à un site :

$$\sum_{k=1}^{K} z_i^k = 1 \quad \forall i \in V\backslash\{0\} \tag{2.2-3}$$

$$z_i^k = 1 \quad \forall i \in \{0^+, 0^-\}, k = 1 \dots K \tag{2.2-4}$$

- sur la conservation de la charge de l'agent k lors du traitement de la demande D_j :

$$y_i^k - y_j^k + D_j \begin{cases} = 0 & \text{si } x_{i,j}^k = 1 \\ \in [-Q + D_j, Q + D_j] & \text{sinon} \end{cases} \quad \forall (i,j) \in E, k = 1 \dots K \tag{2.2-5}$$

Les contraintes $(2.2-5)$ n'est pas linéaire, voici une linéarisation pour ces contraintes :

$$\big(Q - D_j\big)\big(x_{ij}^k - 1\big) \leq y_i^k - y_j^k + D_j \leq \big(Q - D_j\big)\big(1 - x_{ij}^k\big) \quad \forall (i,j) \in E, k = 1 \dots K \tag{2.2-6}$$

- sur la conservation de la charge de l'agent k :

$$y_{0+}^k - y_{0-}^k + \sum_{i \in V\backslash\{0\}} D_i z_i^k = 0 \quad \forall \mathrm{k} = 1 \dots \mathrm{K} \tag{2.2-7}$$

- sur la charge initiale des agents :

$$\sum_{k=1}^{K} y_{0^+}^k \leq R \tag{2.2-8}$$

- définition des variables :

$$x_{ij}^k \in \{0,1\} \quad \forall (i,j) \in E, k = 1 \dots K \tag{2.2-9}$$

$$y_i^k \in \mathbb{N}, 0 \leq y_i^k \leq Q \quad \forall i \in V\backslash\{0\} \cup \{0^+, 0^-\}, k = 1 \dots K \tag{2.2-10}$$

$$z_i^k \in \{0,1\} \quad \forall i \in V\backslash\{0\} \cup \{0^+, 0^-\}, k = 1 \dots K \tag{2.2-11}$$

2.4.4 Fonction objectif

La fonction objectif est de minimiser la durée de redéploiement (t_{max}) et la durée totale des tournées, en tenant compte de la dominance induite par le coefficient α. La valeur de α est fixée de manière à assurer qu'un grain unitaire sur t_{max} domine systématiquement un gain sur le sommet des durées. On la fixe ici à 0.00001. La fonction objectif est donc comme suite :

$$Z = t_{max} + \alpha \sum_{k=1}^{K} \sum_{(i,j) \in E} T_{ij} x_{ij}^k$$

2.4.5 Modèle mathématique du PPRV

La formulation mathématique du PPRV est donnée par :

$$
\text{(PL 2.1)}\begin{cases}
\text{Minimiser } t_{max} + \alpha \sum_{k=1}^{K} \sum_{(i,j)\in E} T_{ij} x_{ij}^k & & & \\
\text{s.c.} & & & \\
\quad \sum_{(i,j)\in E} T_{ij}^k x_{ij}^k - t_{max} \leq 0 & & \forall k = 1 \dots K & (2.1-1)\\
\quad \sum_{j:(i,j)\in E} x_{ij}^k = \sum_{j:(j,i)\in E} x_{ji}^k = z_i^k & & \forall i \in V\backslash\{0\}, k = 1 \dots K & (2.1-2)\\
\quad \sum_{k=1}^{K} z_i^k = 1 & & \forall i \in V\backslash\{0\} & (2.1-3)\\
\quad z_i^k = 1 & & \forall i \in \{0^+, 0^-\}, k = 1 \dots K & (2.1-4)\\
\quad y_i^k - y_j^k + D_j \begin{cases} = 0 \\ \in [-Q + D_j, Q + D_j] \end{cases} & \begin{matrix}\text{si } x_{i,j}^k = 1 \\ \text{sinon}\end{matrix} & \forall (i,j) \in E, k = 1 \dots K & (2.1-5)\\
\quad y_{0^+}^k - y_{0^-}^k + \sum_{i\in V\backslash\{0\}} D_i z_i^k = 0 & & \forall k = 1 \dots K & (2.1-7)\\
\quad \sum_{k=1}^{K} y_{0^+}^k \leq R & & & (2.1-8)\\
\quad \text{Contraintes d'élimination de cycles} & & & (2.1-12)\\
\quad x_{ij}^k \in \{0,1\} & & \forall (i,j) \in E, k = 1 \dots K & (2.1-13)\\
\quad y_i^k \in \mathbb{N}, 0 \leq y_i^k \leq Q & & \begin{matrix}\forall i \in V\backslash\{0\} \cup \{0^+, 0^-\}, \\ k = 1 \dots K\end{matrix} & (2.1-14)\\
\quad z_i^k \in \{0,1\} & & \begin{matrix}\forall i \in V\backslash\{0\} \cup \{0^+, 0^-\}, \\ k = 1 \dots K\end{matrix} & (2.1-15)\\
\quad t_{max} \geq 0 & & & (2.1-16)
\end{cases}
$$

On introduit les contraintes supplémentaires $(2.1-12)$ qui permettent d'exclure des cycles. Plusieurs types de contraintes d'élimination de cycles vont être présentés dans la section 2.4.6.

Ce modèle devient linéaire en replaçant les contraintes $(2.1-5)$ par $(2.1-6)$. Il peut alors être résolu par un solveur linéaire comme CPLEX ou Gurobi. En terme de la complexité spatiale, le modèle linéaire contient $(K(|E|+2|V|+1))$ variables entières et $(K(3|E|+3|V|+1)+2)$ contraintes (sans compter les contraintes d'élimination de cycles).

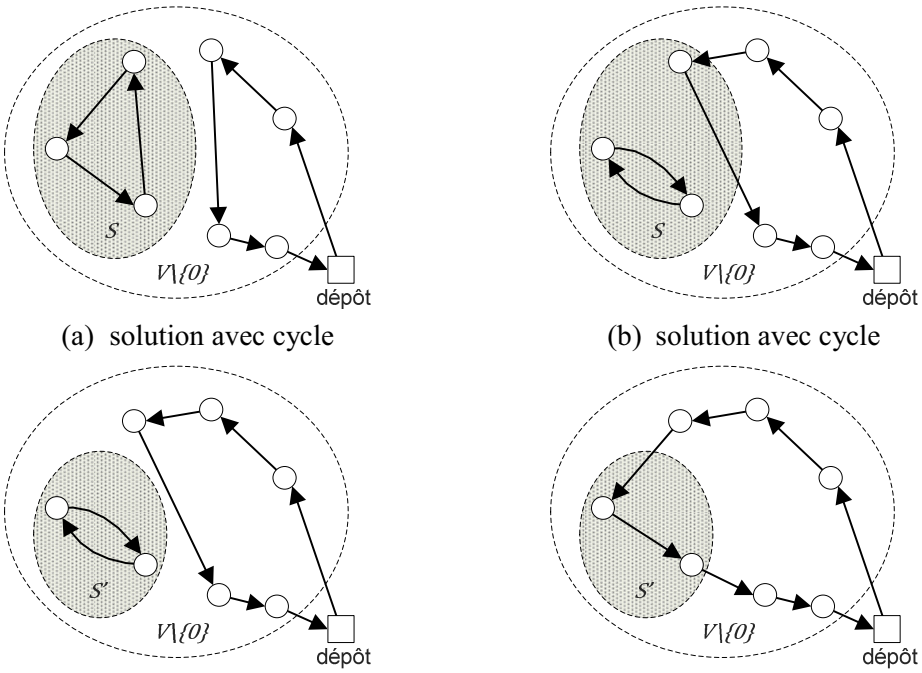
2.4.6 Contraintes d'élimination de cycles

Il existe plusieurs types de contraintes d'élimination de cycles utilisées pour le TSP. Ces contraintes peuvent être adaptées aux problèmes de collecte et livraison en ajoutant un indice k lié aux agents sur les variables correspondantes. Voici les 3 types principaux de contraintes :

1) *Capacity-Cut Constraints* (CCC) : elles éliminent les cycles sur tous les sous-ensembles de sommets S possibles dans $V\backslash\{0\}$ pour chaque agent. Si un sous-ensemble S contient des sommets visités par l'agent k, alors il faut avoir au moins un arc de la tournée k liant les deux parties de sommets S et $V\backslash\{0\}\backslash S$. Les contraintes CCC sont exprimées par $(2.1-17)$. Elles n'introduisent aucune nouvelle variable dans la formulation.

$$\text{(CCC)} \qquad \sum_{i\in S} \sum_{j\notin S} x_{ij}^k \geq z_h^k \qquad \forall S \subseteq V\backslash\{0\}, h \in S, S \neq \emptyset, k = 1,..,K \qquad (2.1-17)$$

La Figure 2.6 donne un exemple d'élimination de cycle par CCC. On suppose que les sommets dans la zone $V\backslash\{0\}$ sont visités par l'agent k, et les 3 sommets dans la zone grise de la figure (a) font partie du sous-ensemble S. La figure (b) illustre une solution après avoir introduit une contrainte CCC liée au sous-ensemble S. Cette contrainte impose d'avoir au moins un arc dans la solution qui lie les deux parties de sommets (S et $V\backslash\{0\}\backslash S$). La solution reste avec un cycle. On a donc besoin d'un autre sous-ensemble (noté S') pour éliminer ce cycle (voir la figure (c)). La figure (d) est une solution sans cycle en ajoutant une contrainte liée à S'. Pour éliminer tous les cycles possibles, il faut prendre en compte toutes les combinaisons de sommets dans $V\backslash\{0\}$. En pratique, CCC introduit un nombre exponentiel de contraintes dans la formulation. Ces contraintes sont très fortes mais elles alourdissent considérablement la résolution et risquent de consommer un temps de calcul important.



(a) solution avec cycle (b) solution avec cycle

(c) solutions avec cycle (d) solution sans cycle

Figure 2.6 : Elimination de cycle par CCC

Dantzig, Fulkerson and Johnson constraint (DFJ) [Dan54]: DFJ est une variante de CCC. On énumère aussi tous les sous-ensembles de $V\backslash\{0\}$. Pour chaque sous-ensemble S, on limite le nombre maximum d'arcs parmi les sommets de ce sous-ensemble. Le nombre est fixé à $|S| - 1$. Cela élimine évidement les cycles dans S (en effet, il faut avoir au moins autant d'arcs que de sommets pour former un cycle entre ces sommets). Dans ce cas, les cycles se trouvant dans les deux figures (a) et (c) de la Figure 2.6 vont être éliminés. Les contraintes DFJ sont exprimées par $(2.1 - 18)$. Le nombre de contraintes engendrées par DFJ est moins grand qu'avec CCC, mais il reste aussi exponentiel en fonction du nombre de sommets dans un problème.

$$\text{(DFJ)} \qquad \sum_{i \in S} \sum_{j \in S} x_{ij}^{k} \leq |S| - 1 \qquad \forall S \subseteq V\backslash\{0\}, |S| \geq 2, k = 1, .., K \quad (2.1 - 18)$$

2) *Miller, Tucker and Zemlin constraints* (MTZ) [Mil60] : CCC énumère tous les sous-ensembles de $V\backslash\{0\}$. Elle demande d'ajouter un grand nombre de contraintes dans la formulation. Cela rend le programme linéaire résultant très difficile à résoudre. Au lieu d'énumérer tous les sous-ensembles de $V\backslash\{0\}$, MTZ repose sur une variable correspondant à la date de passage (ou un indice topologique) de telle sorte que le choix d'un arc lie ces variables entre elles. Cette méthode est proposée initialement par *Miller, Tucker and Zemlin* [Mil60] pour le TSP. Elle est reformulée par *Toth and Vigo* [Tot02] pour le CVRP. La formulation pour le CVRP est définie par $(2.1-19)$, dans laquelle u_i^k sont les variables représentant la charge de l'agent k après avoir visité le sommet i et $D_i \leq u_i^k \leq Q$. Cette formulation ne peut cependant pas être utilisée dans une formulation où il existe des valeurs négatives de demande D_i (*i.e.* problèmes de collecte et livraison).

$$(\mathrm{MTZ}_1) \quad u_i^k - u_j^k + Q x_{ij}^k \leq Q - D_j \qquad \begin{matrix} \forall i,j \in V\backslash\{0\}, i \neq j, \\ D_i + D_j \leq Q, k = 1,..,K \end{matrix} \qquad (2.1-19)$$

MTZ peut être reformulée par $(2.1-20)$ afin d'être adaptée aux problèmes contenant des demandes négatives. Dans l'équation $(2.1-20)$, les variables v_i^k sont des variables entières et $0 \leq v_i^k \leq |V|-1$. Ces variables correspondent à l'ordre de passage de l'agent k sur le sommet i. L'utilisation de MTZ dans un modèle linéaire va donc amener $(K|V|)$ nouvelles variables et $(K|E|)$ contraintes.

$$(\mathrm{MTZ}_2) \quad v_i^k - v_j^k + |V|\, x_{ij}^k \leq |V| - 1 \quad \forall (i,j) \in E, j \neq 0, k = 1,..,K \quad (2.1-20)$$

Il existe plusieurs extensions du MTZ comme par exemple *Desrochers and Laporte constraints* (DL) [Des91] et *Sherali and Driscoll constraints* (SD) [She02]. DL propose des contraintes plus fortes en appliquant une technique de lifting. L'équations $(2.1-21)$ – $(2.1-22)$ donnent l'adaptation de DL au PPRV. L'équation $(2.1-21)$ contient des contraintes plus serrées que celles de $(2.1-20)$. Les contraintes $(2.1-22)$ permettent de réduire les bornes pour les variables v_i^k.

$$(\mathrm{DL}) \begin{cases} v_i^k - v_j^k + (|V|-1)x_{ij}^k + (|V|-3)x_{ji}^k \leq |V| - 2 & \begin{matrix} \forall i,j \in V\backslash\{0\}, i \neq j, \\ k = 1,..,K \end{matrix} \quad (2.1-21) \\ 1 + (|V|-3)x_{i0}^k + \sum_{j \in V\backslash\{0\}} x_{ji}^k \leq v_i^k \leq |V| - 1 - (n-3)x_{0i}^k - \sum_{j \in V\backslash\{0\}} x_{ij}^k & \begin{matrix} \forall i \in V\backslash\{0\}, \\ k = 1,..,K \end{matrix} \quad (2.1-22) \end{cases}$$

SD utilise des variables pour représenter l'ordre des arcs dans des tournées. Nous n'avons pas fait l'adaptation de ce type de contraintes pour le PPRV. On peut se référer à [She02] pour plus de détails.

3) *Gavish and Graves constraints* (GGC) : elles utilisent un réseau de flot auxiliaire. Il existe deux méthodes selon le type de flot : si un flot représente une tournée d'agent qui part de la source (dépôt) et passe sur tous les sites dans la tournée de l'agent, alors ce sont les *Gavish and Graves aggregated flow constraints* (GGA) ; si on envoie un flot pour chaque site de la source vers ce site, alors cela correspond aux *Gavish and Graves disaggregated flow constraints* (GGD).

Dans GGA, on utilise des variables entières f_{ij}^k représentant la quantité de flot k sur l'arc (i,j) où $0 \leq f_{ij}^k \leq (|V|-1)z_i^k$. L'équation $(2.1-23)$ définit des contraintes de conservation du flot sur un sommet. Pour le flot k, le passage sur un site qui est

visité par l'agent k consomme une unité de flot. GGA assure la connexité d'une tournée à l'aide d'une indexation des arcs donnée par les variables de flots f_{ij}^k. Elle contient $(K|E|)$ nouvelles variables et $(K(|V|-1))$ contraintes.

$$\text{(GGA)} \quad \sum_{j:(i,j)\in E} f_{ji}^k - \sum_{j:(i,j)\in E} f_{ij}^k = z_i^k \quad \forall i \in V\backslash\{0\}, k = 1,..,K \qquad (2.1-23)$$

GGD utilise un flot pour chaque site traité. Le flot associé au site l part de l'origine (dépôt) avec une seule unité de flot. On cherche un chemin défini par les variables naturelles pour aller au site l. Le passage sur des sites différents du site l ne consomme pas d'unité de flot. Chaque arc est associé à une capacité limitée par x_{ij}^k. Le flot se termine en arrivant sur le site l. La connexité d'une tournée est déduite du fait qu'il existe un flot (chemin) de l'origine (dépôt) pour tous les sites d'une tournée. Dans le cas où une tournée contient un cycle fermé (exemple : le cycle de 3 sites sur la Figure 2.6 (a)), alors les cheminements des flots du dépôt vers les sommets de ce cycle ne sont plus réalisés. GGD utilise des variables binaires à 4 indices f_{ijl}^k où l est le site destination et $f_{ijl}^k \leq x_{ij}^k, \forall l \in V\backslash\{0\}$. Les contraintes GGD peuvent être définies par les équations $(2.1-24)$ à $(2.1-27)$. Les contraintes $(2.1-24)$ sont des contraintes de conservation de flots. Les contraintes $(2.1-25)$ et $(2.1-26)$ définissent respectivement l'origine et la destination des flots. L'équation $(2.1-27)$ décrit la capacité d'arc pour les flots. GGD contient $(K|E|(|V|-1))$ nouvelles variables et $(K(|E|+|V|+1)(|V|-1))$ contraintes.

$$\text{(GGD)} \begin{cases} \sum_{j:(i,j)\in E} f_{ijl}^k - \sum_{j:(j,i)\in E} f_{jil}^k = 0 & \forall i \in V\backslash\{0\}, l \in V\backslash\{0\}, i \neq l \\ & k = 1,..,K \qquad (2.1-24) \\ \sum_{j:(0,j)\in E} f_{0jl}^k = z_l^k & \forall l \in V\backslash\{0\}, k = 1,..,K \qquad (2.1-25) \\ \sum_{j:(j,i)\in E} f_{jil}^k = z_l^k & \forall l \in V\backslash\{0\}, i = l, k = 1,..,K \qquad (2.1-26) \\ f_{ijl}^k \leq x_{ij}^k & \forall (i,j) \in E, l \in V\backslash\{0\}, k = 1,..,K \qquad (2.1-27) \end{cases}$$

Sur le modèle linéaire du PPRV, nous avons testé MTZ et GGD pour l'élimination des cycles dans les tournées d'agent. La comparaison de ces deux méthodes sera détaillée dans la section 2.5. Par ailleurs, il existe d'autres types de contraintes comme *Multi-commodity flow formulations* (MCF), *Time dependent constraints* (TD) et *Precedence variable based constraints* (PVB). Pour plus de détails sur l'application des contraintes d'élimination de cycles sur des problèmes de tournées de véhicules, on pourra se référer à l'état de l'art dans [Tot02, Orm07, Onc09].

2.5 Expérimentations

Dans cette section, nous allons présenter les résultats donnés par un solveur de programmes linéaires sur le modèle proposé dans la section 2.4. Nous nous intéressons tout d'abord à la résolution des petites instances en utilisant les différentes contraintes d'élimination de cycles. La complexité de ces contraintes s'est analysée essentiellement du point de vue du temps de calcul. La méthode avec le meilleur résultat sera sélectionnée pour les tests suivants. Nous nous intéressons ainsi à la résolution des petites instances avec un nombre d'agents différent.

Le résultat obtenu sera utilisé dans les chapitres suivants pour mesurer la qualité de la méthode approchée.

2.5.1 Environnements de l'expérimentation et instances utilisées

Les tests ont été effectués sur un serveur doté de 2 processeurs AMD Opteron 2.3 GHz, sous Linux (CentOS en 64 bits). Deux solveurs de programme linéaire sont installés : IBM Ilog CPLEX 11.0 (64 bits) et Gurobi Optimizer 4.5.1 (64 bits). Parmi ces deux solveurs, nous avons choisi d'utiliser Gurobi. Le nombre de processus / threads pour le branchement est limité à 8 dans Gurobi. Nous utilisons les interfaces en C++ pour accéder au solveur Gurobi. Les programmes de test sont développés en C++ et compilés par GNU gcc (g++) 4.1.2.

Les instances utilisées sont adaptées de celles de 1-PDTSP (disponible sur le site http://webpages.ull.es/users/hhperez/PDsite/). Pour décrire une instance du PPRV, nous utilisons les notations présentées dans la section 2.2. Une instance du PPRV contient donc :

- un ensemble de sites $V = \{0, .., n\}$ dont un dépôt (numéroté 0) et n sites de services (numérotés de 1 à n). Chaque site i est associé à une demande aléatoire $D_i \in [-10, 10]$ et à une position géographique $(x_i, y_j) \in [-500, 500] \times [-500, 500]$;
- un ensemble d'arc $E = \{(i, j) | \forall i, j \in V, i \neq j\}$: chaque arc (i, j) est associé un coût de parcours T_{ij} qui satisfait l'inégalité du triangle. Le coût T_{ij} correspond à la distance euclidienne arrondie à l'entier le plus proche. Il est calculé par la fonction suivante (fonction utilisée dans le solveur CONCORDE) :

$$T_{ij} = \left\lfloor \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2} + 0.5 \right\rfloor$$

- un ensemble de K agents homogènes de capacité $Q \in \{10, 15, 20, 25, 30, 35, 40\ 45, 50, 1000\}$.

Dans ce chapitre, nous avons utilisé un total de 60 instances de 20 à 40 sommets. Ces instances sont classées en 3 groupes selon le nombre de sommets. Chaque groupe contient 10 graphes différents (notés de A à J). Nous avons testé sur les plus petites capacités : 10 et 20. Le nom d'une instance contient les informations liées au nombre de sommets, à la capacité des agents et au numéro du graphe. Par exemple, « n20q10A » correspond à l'instance correspondant au graphe A du groupe de 20 sommets avec la capacité 10. Notons que la complexité liée à la résolution d'une instance est inversement proportionnelle à la capacité des agents.

2.5.2 Comparaison des contraintes d'élimination de cycles

Nous avons lancé un test avec le solveur Gurobi pour comparer les deux types de contraintes d'élimination de cycles : MTZ et GGD. Nous testons sur les instances de 20 sommets avec le nombre d'agents compris entre 1 et 3. La capacité des agents évolue entre 10 et 20. Le temps de calcul est limité à 432000s (120h). Le résultat obtenu est présenté dans le Tableau 2.2 dont les colonnes sont données comme suit :

- Groupe : groupe d'instances ;
- Q : capacité des agents ;
- K : nombre d'agents ;
- MTZ : résultats avec les contraintes MTZ ;
- GGD : résultats avec les contraintes GGD ;

- Nb. opt. : le nombre de solutions optimales obtenues sur les 10 instances du groupe avant d'atteindre la durée limite ;
- Temps moy. : le temps de calcul moyen des 10 instances du groupe avec Gurobi.

Nous remarquons que MTZ est bien plus efficace que GGD dans les cas où $K \in \{1, 2\}$. Lorsque le nombre d'agents est égal à 3, MTZ devient moins performant. Comme montré dans le Tableau 2.2 : pour le groupe N20Q20, MTZ présente environ 37.8% d'économie en temps de calcul que GGD dans le cas avec 2 agents ; mais dans le cas avec 3 agents, il est 3 fois plus lent que GGD. Nous utilisons MTZ comme contraintes d'élimination de cycles dans la section suivante pour $K \in \{1, 2\}$ et GGD pour $K = 3$.

Instances			MTZ		GGD	
Groupe	Q	K	Nb. opt.	Temps moy.	Nb. opt.	Temps moy.
N20	10	1	10	58.6	10	456.2
N20	10	2	10	797.1	10	3976.6
N20	10	3	9	19472.2	10	**17613.0**
N20	20	1	10	18.9	10	101.0
N20	20	2	10	879.8	10	1415.0
N20	20	3	9	35860.0	10	**11312.9**

Tableau 2.2 : Comparaison des contraintes d'élimination de cycles

2.5.3 Résultat pour le PPRV

Nous nous intéressons ici à la résolution du PPRV de manière exacte en utilisant le solveur Gurobi. Nous avons utilisé les instances de 20 à 40 sommets avec le nombre d'agents de 1 à 3. Le temps de calcul est limité à $120k$ heures où k est le nombre d'agents. Les résultats sont présentés respectivement dans les Tableau 2.3, Tableau 2.4 et Tableau 2.5 pour les tests sur 1 agent ($K = 1$), 2 agents ($K = 2$) et 3 agents ($K = 3$). Les deux derniers tableaux seront utilisés pour comparer aux résultats de la méthode approchée dans le chapitre 2.

Le Tableau 2.3 donne le résultat des petites instances avec 1 agent. Nous étudions l'influence liée à la quantité d'objets R disponibles initialement sur le dépôt (pour la charge initiale) en prenant les différentes valeurs de R dans $\{0, Q/2, Q\}$. Notons que dans le cas général (*i.e.* $R \geq Q$), le résultat du PPRV à un agent est cohérent avec celui du 1-PDTSP. D'ailleurs, la charge initiale est interdite lorsqu'on utilise le paramètre $R = 0$. Le résultat avec un tel paramètre sera utilisé aussi dans le chapitre 3 pour évaluer le gain apporté par le passage multiple sur les sommets.

Les colonnes de Tableau 2.3 sont définies comme suit :

- Grp. : groupe d'une instance ;
- Id : identifiant du graphe ;
- $Q = \{10, 20\}$: capacité des agents, qui distingue les deux instances basées sur un même graphe ;
- $R = \{0, Q/2, Q\}$: paramètres sur la quantité d'objets disponible initialement sur le dépôt (pour la charge initiale) ;
- Val. : solution obtenue par Gurobi ;
- Temps : temps de calcul en seconde avec Gurobi.

Le Tableau 2.3 montre que la présence de la charge initiale permet de réduire le coût de redéploiement. Plus précisément, le coût moyen du groupe de 20 sommets avec la capacité 10 (20) augmente 0.396% (1.381%) en réduisant R de Q à 0. Pour le groupe de 40 sommets avec

la capacité 10 (20), l'augmentation est de 1.497% (1.620%) en moyenne. Par contre, cette augmentation est beaucoup moins importante lors qu'on réduit la valeur R de Q à $Q/2$: nous avons obtenu les mêmes résultats pour les groupe de 20 et de 30 sommets. Le coût moyen a augmenté 0.128% (0.013%) pour le groupe de 40 sommets avec la capacité 10 (20). Au niveau du temps de résolution, le solveur Gurobi utilise en moyenne 30.93 (13.13) secondes pour le groupe de 20 sommets avec la capacité 10 (20), et 54.47 (13.60) heures pour le groupe de 40 sommets avec la capacité 10 (20).

		$Q = 10$						$Q = 20$					
		$R = 0$		$R = 5$		$R = 10$		$R = 0$		$R = 10$		$R = 20$	
Grp.	Id	Val.	Temps	Val.	Temps	Val.	Temps	Val.	Temps	Val.	Temps	Val.	Temps
N20	A	4963	13	4963	20	4963	21	3816	2	3816	2	3816	3
	B	4976	11	4976	16	4976	13	4224	6	4224	10	4224	14
	C	6333	25	6333	42	6333	148	4492	53	4492	30	4492	69
	D	6334	9	6280	10	6280	91	4767	8	4706	14	4706	42
	E	6415	19	6415	39	6415	19	4673	9	4673	14	4673	17
	F	4805	12	4805	22	4805	58	4262	12	4118	2	4118	4
	G	5119	9	5119	12	5119	19	4399	6	4369	5	4369	11
	H	5734	17	5594	6	5594	95	4372	2	4159	2	4159	1
	I	5130	17	5130	26	5130	106	4138	4	4116	9	4116	10
	J	4430	3	4410	14	4410	16	3815	11	3700	4	3700	18
N30	A	6403	45140	6403	13369	6403	32741	5067	469	4918	132	4918	108
	B	6646	413	6603	856	6603	430	5109	867	5109	2021	5109	5588
	C	6486	134411	6486	52972	6486	121400	4901	1162	4901	2271	4901	104667
	D	6652	838	6652	1957	6652	4497	5467	1008	5385	9533	5385	25552
	E	6070	71	6070	400	6070	444	5011	440	4916	121	4916	282
	F	5737	269	5737	1070	5737	989	4583	56	4459	26	4459	28
	G	9371	7678	9371	23081	9371	32085	6672	432000	6672	341448	6672	432000
	H	6638	1224	6431	1827	6431	1629	4689	57	4684	161	4684	46
	I	5821	688	5821	850	5821	1849	4487	16	4483	7	4483	16
	J	6271	1423	6187	1672	6187	4082	4936	298	4645	126	4645	362
N40	A	7297	432000	7173	397969	7173	101876	5481	14626	5481	14567	5481	23769
	B	6647	349353	6557	207357	6557	211611	5462	108	5334	326	5334	290
	C	7528	37286	7528	42658	7528	45298	5799	141700	5782	193642	5775	250439
	D	8300	191666	8163	252040	8109	358967	6253	288783	6054	89337	6054	48705
	E	7142	288927	6928	197187	6928	222264	5778	16845	5598	13233	5598	2470
	F	7591	402419	7528	338294	7506	118004	5491	993	5491	13226	5491	8826
	G	7757	208556	7624	147489	7624	182598	5588	3063	5588	3442	5588	3860
	H	6794	64262	6791	83763	6791	108410	5306	1330	5141	806	5141	91004
	I	7223	13792	7215	28826	7215	65331	5348	3358	5262	1023	5262	10028
	J	6741	323328	6528	432000	6512	29305	5386	15963	5277	3415	5277	209656

Tableau 2.3 : Résultat sur les petites instances du PPRV avec un agent ($K = 1$)

Le Tableau 2.4 et le Tableau 2.5 montrent respectivement les résultats avec 2 agents et 3 agents. Dans ces deux tableaux, les colonnes « Grp. », « Id », « $Q = 10/20$ » et « Temps » possèdent la même définition que celles dans le Tableau 2.3. Les autres colonnes sont définies comme suit :

- Tour. $\{1, 2, 3\}$: longueur de chaque tournée ; la colonne « Tour. 1 » correspond à la durée de redéploiement (*i.e.* la plus longe tournée).
- Total : la somme du coût des K tournées.

Dans cette partie, la valeur R est fixé à KQ, où K est le nombre d'agents et Q est la capacité. L'augmentation du nombre d'agents réduit la durée de redéploiement, mais la somme des

trajectoires augmente. Par exemple, lorsqu'on passe de 2 agents à 3 agents, la durée de redéploiement moyenne pour le groupe de 20 sommets avec la capacité 10 (20) diminue de 27.19% (21.90%) mais le coût total augmente de 7.78% (14.33%).

		$Q = 10$				$Q = 20$			
Grp.	Id	Tour. 1	Tour.2	Total	Temps	Tour. 1	Tour.2	Total	Temps
N20	A	2628	2617	5245	2017	2235	2145	4380	525
	B	2590	2397	4987	127	2372	2349	4721	362
	C	3247	3174	6421	2075	2537	2523	5060	6026
	D	3162	3039	6201	160	2434	2364	4798	244
	E	3362	3344	6706	1284	2448	2387	4835	84
	F	2444	2407	4851	136	2326	2207	4533	432
	G	2833	2769	5602	1325	2448	2447	4895	448
	H	2829	2829	5658	97	2299	2226	4525	16
	I	2591	2479	5070	217	2257	2247	4504	192
	J	2398	2274	4672	533	2116	2098	4214	469
N30	A	3330	3329	6659	396468	2584	2499	5083	5981
	B	3598	3593	7191	432000	2589	2554	5143	9593
	C	3452	3404	6856	263154	2696	2684	5380	403202
	D	3565	3561	7126	573011	2713	2631	5344	139281
	E	3256	3205	6461	130001	2674	2610	5284	74472
	F	3053	3032	6085	227701	2526	2504	5030	35132
	G	4742	4731	9473	403207	3479	3467	6946	241211
	H	3380	3362	6742	403202	2561	2516	5077	5706
	I	3000	2956	5956	403205	2379	2360	4739	684
	J	3224	3112	6336	237109	2532	2532	5064	11314

Tableau 2.4 : Résultat sur les petites instances du PPRV avec deux agents ($K = 2$)

		$Q = 10$					$Q = 20$				
Grp.	Id	Tour. 1	Tour. 2	Tour. 3	Total	Temps	Tour. 1	Tour. 2	Tour. 3	Total	Temps
N20	A	1837	1741	1693	5271	17453	1666	1586	1479	4731	4586
	B	1990	1957	1870	5817	6477	1833	1829	1822	5484	9765
	C	2304	2234	2227	6765	14037	1916	1806	1757	5479	36553
	D	2261	2233	2170	6664	5370	1882	1840	1738	5460	52285
	E	2331	2309	2185	6825	46953	1992	1943	1933	5868	25576
	F	1978	1963	1960	5901	150132	1846	1819	1785	5450	78406
	G	1992	1948	1863	5803	10539	1858	1758	1708	5324	49315
	H	2182	2015	1962	6159	8647	1990	1933	1704	5627	69444
	I	1853	1830	1808	5491	6507	1693	1671	1420	4784	4807
	J	1720	1714	1593	5027	9439	1656	1642	1618	4916	34269
N30	A	2268	2248	2219	6735	432000	1890	1875	1790	5555	20060
	B	2440	2407	2315	7162	429692	2078	2070	2022	6170	403260
	C	2435	2420	2193	7048	431865	1986	1971	1951	5908	367298
	D	2405	2388	2255	7048	403208	2066	2046	2022	6134	368502
	E	2372	2321	2243	6936	279781	2068	2058	2010	6136	91166
	F	2214	2078	1988	6280	403258	1989	1986	1921	5896	38287
	G	3227	3225	3224	9676	791116	2439	2393	2352	7184	432000
	H	2368	2338	2254	6960	432000	2008	1994	1882	5884	192543
	I	2113	2074	1976	6163	415011	1757	1756	1752	5265	22243
	J	2286	2267	2188	6741	411655	1966	1954	1858	5778	394534

Tableau 2.5 : Résultat sur les petites instances du PPRV avec trois agents ($K = 3$)

On constate que le temps de calcul augmente très fortement en fonction du nombre de sommets/agents. Par exemple, le temps de calcul en moyenne est à 797.1 secondes à 2 agents et à 27555.4 secondes (environ 7.6 heures) à 3 agents pour le groupe de 20 sommets avec la capacité 10. Pour le groupe de 30 sommets avec 2 agents de capacité 10, le temps moyen augmente à 356907.6 secondes (environ 99.1 heures). C'est aussi la raison pour laquelle les tests n'ont été effectués que sur les petites instances.

2.6 Conclusion

Dans ce chapitre, nous nous sommes intéressés au problème de la planification du redéploiement de véhicules partagés (PPRV), qui est considéré comme la version de base pour les problèmes de plan de redéploiement (PPR). Le PPRV est un problème de collecte et de livraison (PDP : *Pickup and Delivery Problem*) du type *Many-to-Many* à multi-agents. À notre connaissance, aucun modèle mathématique n'existe pour ce type de problème. Nous proposons ici un modèle linéaire pour la résolution exacte du PPRV à l'aide d'un solveur linéaire comme Gurobi ou Cplex. Nous avons étudié l'efficacité des diverses contraintes d'éliminations de sous-cycle utilisées par les problèmes de tournées de véhicules classiques.

Dans la partie expérimentation numérique, les tests sont effectués sur des petites instances, avec différents nombres d'agents. Nous avons comparé plus particulièrement les deux contraintes d'élimination de sous-cycles suivantes : MTZ et GGD. Le modèle linéaire utilisant les contraintes GGD offre des temps de résolutions plus faible lorsque le nombre d'agent dépasse 3 pour les instances à 20 sommets. Quand la taille des instances augmente, les contraintes MTZ devient plus efficace. Par ailleurs, la taille des modèles générés en utilisant MTZ est beaucoup plus faible. Par conséquent, nous utilisons le MTZ comme contrainte d'élimination de cycles pour les tests avec 1 et 2 agent(s) et GGD pour 3 agents.

Le temps de résolution exacte avec un solveur mathématique est très long. Dans notre cas, une partie des instances n'a pas pu être résolue optimalement (ou la preuve d'optimalité n'a pas pu être effectuée) et le solveur linéaire a atteint le temps maximal de calcul que nous lui avions alloué et qui valait 120 heures. Le résultat présenté dans ce chapitre sert de référence pour évaluer la performance des méthodes approchées dans les chapitres suivants.

Chapitre 3

Résolution approchée du PPRV

Sommaire

3.1 Étude des méthodes approchées 83
3.1.1 Méthodes approchées pour les problèmes de type 1-PDP 84
3.1.2 Relaxations de contrainte pour la résolution du PPRV 86
3.2 Présentation de la méthode approchée 88
3.2.1 Schéma ILS/VND 89
3.2.2 Schéma général de la méthode proposée 91
3.2.3 Procédure de construction d'une tournée géante 92
3.2.4 Procédure de perturbation 94
3.2.5 Transformation d'une tournée géante en une solution du PPRV 96
3.2.6 Recherches locales 101
3.3 Expérimentations 106
3.3.1 Qualité des tournées géantes 107
3.3.2 Qualité des solutions du PPRV en multi-agents 111
3.4 Conclusion 112

Dans ce chapitre, nous nous intéressons à la résolution approchée du PPRV. Nous proposons une méthode approchée de type « *route-first, cluster-second* ». Le principe de la méthode consiste à construire d'abord une tournée géante, puis à la découper en plusieurs tournées d'agents à l'aide de la procédure Split. La méthode proposée utilise le schéma ILS/VND. Elle utilise les opérateurs adaptés à la relaxation de la contrainte de capacité (RCC). L'application de la RCC pour traiter le 1-PDP à un seul agent est une approche fréquente [Her04, Her08, Hos10a]. D'un autre côté, Split a permis d'obtenir des performances remarquables pour la résolution de problèmes du type VRP [Prin95]. Nous proposons ici une application de ces deux techniques à la résolution du PPRV : la RCC pour l'amélioration de la tournée géante et Split pour la transformation de la tournée géante en une solution du PPRV.

La section 3.1 fournit une revue de littérature sur les méthodes approchées pour la résolution des problèmes de type 1-PDP. Nous étudions l'utilisation des relaxations de contrainte pour traiter le PPRV. La section 3.2 présente en détail la méthode proposée. Enfin, nous présentons les résultats obtenus dans la section 3.3.

3.1 Étude des méthodes approchées

Nous commençons par présenter les méthodes approchées pour les problèmes de type 1-PDP. Le PPRV est en fait un problème de type 1-PDP. Plus précisément, le PPRV est équivalent au 1-VRPPD. Nous nous intéresserons sur l'étude des méthodes approchées pour la résolution du PPRV dans la seconde partie.

3.1.1 Méthodes approchées pour les problèmes de type 1-PDP

Les problèmes de type 1-PDP n'ont pas été beaucoup explorés dans la littérature et il n'existe que très peu d'articles les concernant. Parmi ceux-ci, nous nous intéressons plus particulièrement à ceux qui traitent des problèmes dans lesquels la demande et la capacité des agents ne sont pas unitaires, comme le 1-PDTSP et le 1-VRPPD. La demande non-unitaire complexifie la résolution de ces deux problèmes. En particulier, trouver une solution réalisable devient un problème à part entière lorsque la capacité des agents est proche de la demande maximale. Les méthodes approchées pour la résolution de ces deux problèmes utilisent alors souvent une relaxation de la contrainte de capacité.

Le 1-PDTSP est le problème de type 1-PDP le plus étudié. Hernandez-Pérez et Salazar-Gonzalez [Her04] sont les premiers à l'aborder et ils proposent deux heuristiques. La première heuristique utilise le taux d'« *infaisabilité* » lors de l'évaluation d'une solution. Ce taux décrit le déficit au niveau de la capacité de l'agent pour rendre la solution réalisable. Sa valeur est négative lorsqu'une solution est réalisable. Elle est positive si cette solution n'est pas réalisable et sa valeur indique le déficit en capacité pour que la solution soit réalisable. Cette heuristique utilise une procédure de démarrages multiples (multi-start). À chaque itération, elle construit d'abord une solution initiale en appliquant l'heuristique NN (*Nearest Neightbor*), qui favorise l'utilisation des arcs liés avec les deux sommets de types différents en pénalisant légèrement le coût de l'utilisation selon la demande des sommets. Une solution construite peut être non réalisable. Elle est ensuite améliorée par une recherche locale à base de *2-opt* et de *3-opt*. Le critère d'acceptation d'une solution tient compte à la fois du taux d'*infaisabilité* et du coût de la tournée. L'heuristique garde la meilleure solution trouvée. Cette méthode utilise une relaxation sur la capacité de l'agent. La deuxième méthode proposée dans [Her04] est une procédure d'optimisation incomplète. Elle commence par la construction d'une solution initiale en utilisant la même heuristique de construction que précédemment. La structure de voisinage utilisée pour explorer les solutions voisines est de type *k-neighborhood* où k est le nombre d'arcs différents (ou «distance») entre la solution actuelle et chaque voisin. Cette structure peut se traduire dans le modèle PLNE du 1-PDTSP [Her03] avec les contraintes additionnelles suivantes : $\sum_{e \in E_s}(1 - x_e) \leq k$ où E_s est l'ensemble des arcs dans la solution S et x_e est la variable booléenne pour décrire la présence de l'arc e dans une solution voisine de S. C'est le mécanisme fondamental du local branching. Afin de parcourir le voisinage à distance k de la solution courante, il résout de manière exacte ce modèle avec les contraintes additionnelles (en utilisant l'algorithme «*branch-and-cut*»). L'amélioration est faite de manière itérative. Chaque itération consiste à explorer le voisinage de la solution courante à une «distance» inférieure ou égale à k. Le meilleur voisin améliorant est utilisé ensuite comme solution courante à l'itération suivante. Les itérations se terminent lorsqu'il ne trouve plus d'amélioration. La valeur de k est fixée à 3 dans cette méthode. Notons que cette approche correspond à une descente gloutonne dans le local branching.

Hernandez-Pérez *et al.* [Her08] ont proposé ensuite une heuristique hybride GRASP/VND, qui combine le GRASP (*Greedy Randomized Adaptive Search Procedure*) et la VND (*Variable Neighborhood Descent*). Le GRASP est basé sur la répétition d'une phase de construction et d'une phase de recherche locale. La VND repose sur l'utilisation de plusieurs structures de voisinages prédéfinies. Elle explore les solutions d'un seul voisinage à chaque itération. En cas de succès, on revient à la première structure de voisinage. Sinon on passe à la structure suivante. On s'arrête lorsqu'on échoue à trouver un voisin améliorant dans la dernière structure. L'hybride GRASP/VND proposé est en fait un GRASP embarquant une VND (dotée des voisinages définis sur l'échange d'arcs) comme recherche locale. Une deuxième VND (avec les voisinages sur le déplacement de sommets) est utilisée en phase de post-

optimisation. L'objectif de GRASP/VND est de trouver une solution réalisable à l'aide d'une relaxation sur la capacité de l'agent. Puis la post-optimisation essaie d'améliorer la solution trouvée sans relaxation.

Martinovic *et al.* [Mar08] ont présenté un algorithme de recuit simulé (IMSA : *Iterative Modified Simulated Annealing*) pour le 1-VRPPD. IMSA est basé sur l'algorithme de recuit simulé modifié (MSA : *Modified Simulated Annealing*) avec une phase d'amélioration supplémentaire. Le principe de MSA est de créer plusieurs voisins réalisables pour une solution courante dans une même température en implémentant un opérateur de voisinage basé sur le changement de la séquence des sommets de manière aléatoire. La mise à jour de la solution courante repose sur une probabilité dépendant de la température. Dans l'algorithme IMSA, on répète un certain nombre d'itérations de MSA. La solution initiale pour IMSA est générée par un algorithme glouton qui utilise une heuristique NN avec un processus de sélection aléatoire parmi les deux sommets voisins les plus proches. Le sommet sélectionné est inséré après le sommet courant. Il sera considéré comme le sommet courant dans l'itération suivante. Après l'application d'IMSA, une procédure de post-optimisation est appliquée. C'est un algorithme de permutation exacte (EPA : *Exact Permutation Algorithm*), qui est appliqué directement sur les petites instances (inférieure ou égale à 15 sommets) et utilisé pour améliorer les chemins sélectionnés (de 15 sommets) pour les grandes instances. Par ailleurs, l'expérimentation de l'algorithme proposé ne considère qu'un seul véhicule ; ce sont donc des instances de 1-PDTSP.

Zhao *et al.* [Zha09] ont introduit un Algorithme Génétique pour la résolution du 1-PDTSP. L'algorithme commence par la création de la population des solutions réalisables en utilisant une heuristique de construction non déterministe basée sur l'heuristique NN. La population générée est ensuite améliorée par une recherche locale à base de *2-opt*. Pour renouveler la population, un opérateur de croisement basé sur la phéromone (liée aux arcs) est utilisé. L'idée de phéromone vient de l'algorithme de colonies de fourmis. Dans la procédure de croisement, la création d'une solution fils se réalise de manière à insérer itérativement des sommets à la fin de la tournée en tenant compte des voisins dans les deux solutions parent et de la phéromone liée au dernier sommet inséré. La procédure commence par un sommet sélectionné de façon aléatoire. À chaque itération, le sommet pour la position suivante est sélectionné de manière suivante : (a) si tous les voisins de ce sommet présent dans les deux solutions parents sont déjà insérés dans la solution fils ou si aucun voisin ne peut être inséré sans violer la contrainte de capacité, alors on applique une règle probabiliste dépendant de la phéromone cumulée sur les arcs liés au dernier sommet inséré pour choisir un sommet qui n'appartient pas à la liste des voisins ; (b) sinon, on prend le sommet voisin le plus proche dans la liste. Une solution fils générée est toujours réalisable. Elle est optimisée ensuite par une recherche locale à base de *2-opt* avant d'appliquer un opérateur de mutation basée sur une procédure de 3-échange. L'application de la mutation avec 3-échange sur les sommets donne 5 solutions possibles. La meilleure est acceptée. Une nouvelle population est générée en remplaçant la moitié des solutions par les solutions fils créées. Les phéromones sont mises à jours après chaque renouvellement de la population des solutions. Cet algorithme travaille sur les espaces de solutions réalisables, sans utiliser les relaxations de contrainte. Notons que les algorithmes de construction dans [Mar08] et [Zha09] ne garantissent pas une solution réalisable pour toutes les instances de [Her04b].

Hosny *et al.* [Hos10a] ont proposé une méthode hybride NVS/SA (*Variable Neighborhood Search/Simulated Annealing*) pour la résolution du 1-PDP à un véhicule (*i.e.* 1-PDTSP). Le principe de VNS/SA est d'utiliser un critère d'acceptation probabiliste basé sur la température

de SA lorsqu'une solution explorée dans VNS est pire que la solution courante. L'utilisation d'un tel critère permet à la recherche locale de sortir d'un optimum local. L'algorithme VNS/SA est lancé de manière itérative. Chacune exécution commence par la meilleure solution obtenue à l'itération précédente. Le nombre maximal de solutions voisines à explorer dans une exécution de VNS/SA n'est pas fixé, il dépend du nombre d'itération courant. Il se réfère en fait au nombre d'itérations maximales dans VNS/SA. Au lieu de recommencer la construction lorsque l'insertion d'un sommet n'est plus possible, ils autorisent l'utilisation de la relaxation sur la capacité de l'agent dans la construction d'une solution.

À notre connaissance, le cas multi-véhicule des problèmes de type 1-PDP n'a pas été traité expérimentalement, puisque même Martinovic *et al.* [Mar08] ne résolvent finalement que des instances à un véhicule.

3.1.2 Relaxations de contrainte pour la résolution du PPRV

Deux relaxations peuvent essentiellement être appliquées pour la résolution du PPRV par une méthode approchée : la relaxation de contrainte de la capacité (RCC) et la relaxation de contrainte de mono-accès (RCA).

Nous rappelons que le PPRV contient 4 contraintes selon sa définition dans la section 2.2 :

(c2.1) une tournée commence et se termine au dépôt ;
(c2.2) un site est visité au plus une fois ;
(c2.3) la charge d'un agent ne peut pas dépasser sa capacité ;
(c2.4) toutes les demandes doivent être satisfaites.

Parmi ces 4 contraintes, les contraintes (c2.2) et (c2.3) rendent difficile la recherche locale pour trouver des solutions réalisables de bonne qualité : dans une solution du PPRV, la visite d'un sommet s'accompagne toujours d'une opération sur l'agent : collecte ou livraison des objets selon la demande (positive ou négative). Le nombre de passages sur un sommet est limité par la contrainte mono-accès (c2.2), *i.e.* un sommet est visité au plus une fois par des agents. Ainsi en mono-accès, tous les objets associés à une demande doivent être collectés/livrés en une seule opération de l'agent. La contrainte (c2.3) limite la charge qu'un agent peut transférer. Dans le cas où la capacité des agents est petite (et souvent proche de la demande maximale), la recherche locale risque d'avoir beaucoup de difficultés à explorer l'espace des solutions réalisable du PPRV (en se bloquant rapidement dans un optimum local très éloigné de l'optimum global). De ce fait, la relaxation de l'une des deux contraintes pendant la recherche des solutions (dans une méthode heuristique) aide à agrandir l'espace des solutions et à augmenter le nombre de solutions réalisables, mais pénalisées. La relaxation peut être même nécessaire pour pouvoir identifier des solutions réalisables pour certaines instances très contraintes du PPRV.

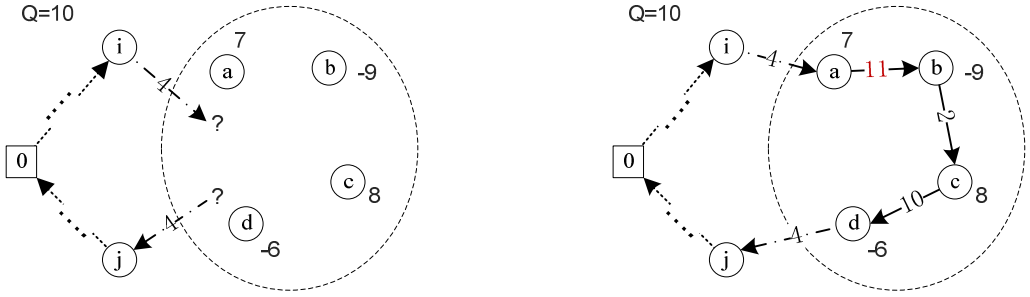
On s'intéresse ici à deux relaxations de contrainte : Relaxation de Contrainte de la Capacité des agents (RCC) et Relaxation de Contrainte mono-Accès d'un sommet (RCA). La RCC autorise la charge d'un agent à dépasser temporairement sa capacité. La RCA autorise le passage multiple sur des sommets et chaque passage doit prendre en compte une partie de la demande (liée au sommet). La livraison sur un sommet de collecte est strictement interdite et vice-versa.

3.1.2.1 Relaxation de contrainte de capacité

La relaxation de contrainte de capacité (RCC) est utilisée essentiellement dans la construction

et l'amélioration de la tournée géante dans notre méthode. Elle autorise la charge de l'agent à dépasser sa capacité durant l'exploration de l'espace des tournées géantes. La quantité de dépassement permet de mesurer la réalisabilité d'une solution. Une solution est dite *réalisable* si le dépassement est nul. Le dépassement est limité par une valeur prédéfinie, dit *seuil de relaxation* (noté tl). Ainsi, la charge d'un agent peut atteindre jusqu'à $(Q + tl)$ dans une solution relâchée, où Q est la capacité (physique) des agents. Le principe de la RCC consiste en fait à augmenter temporairement la capacité des agents, afin d'agrandir l'espace des solutions pour éviter que la recherche locale ne tombe rapidement dans un optimum local. Puis la relaxation peut être progressivement restreinte jusqu'à ne plus autoriser de dépassement.

La Figure 3.1 illustre la RCC : on suppose que les 4 sommets (a, b, c et d) doivent être insérés entre les sommets i et j. La demande de ces 4 sommets est indiquée dans la Figure 3.1 (a). La charge de l'agent sur l'arc (i, j) vaut 4. Évidement, l'insertion des 4 sommets en cette position n'est pas réalisable car la capacité va être dépassée (elle vaut 10 ici). La Figure 3.1 (b) donne une solution relâchée dont la réalisabilité vaut 1, car la charge de l'agent sur l'arc (a, b) vaut 11. Cette solution sera acceptée si le seuil $tl \geq 1$.



(a) cas impossible sans relaxation (b) solution avec la RCC

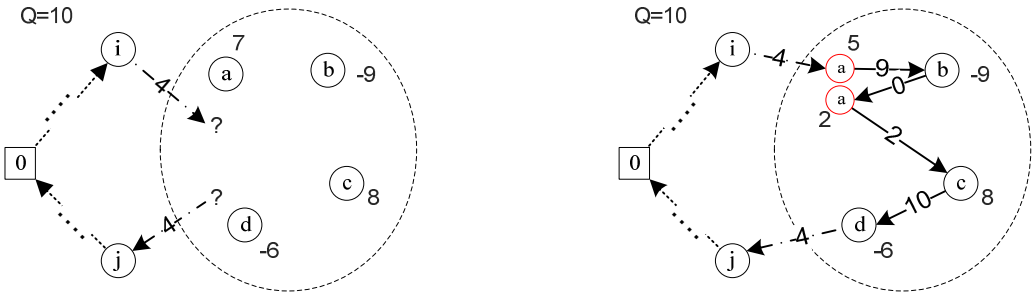
Figure 3.1 : Application de la RCC

Dans la littérature, comme on a mentionné dans la section 3.1.1, la RCC est souvent utilisée pour résoudre des problèmes de type 1-PDP [Her04, Her08, Hos10a].

3.1.2.2 Relaxation de contrainte mono-accès

Le principe de la relaxation de contrainte mono-accès (RCA) consiste à relâcher la contrainte mono-accès sur des sommets. Elle autorise ainsi le passage multiple sur les sommets sous la condition suivante : chaque passage doit s'accompagner d'une opération de livraison ou de collecte sur l'agent selon sa demande (noté c2.5). Par contre, comme les demandes et la capacité des agents sont entières, et qu'il n'y a pas de coût associés à la charge, alors on se restreint à des chargements de quantité entière et non fractionnaire. Ainsi, on ne peut pas avoir de multi-accès lorsque la demande est unitaire.

Le passage multiple sur un sommet en respectant (c2.5) a pour effet de fractionner la demande associée. L'idée d'utiliser la RCA dans une heuristique ou dans une recherche locale vient du fait qu'il est plus facile de respecter la contrainte de capacité lorsque les quantités chargées ou déchargées sont plus faibles, quitte à passer plusieurs fois sur un sommet. Dans la Figure 3.2, la partie de droite illustre une solution relâchée dans laquelle on divise la demande du sommet a en deux parties. Par conséquent, la contrainte de mono-accès est relaxée sur le sommet a.



(a) cas impossible sans relaxation (b) solution avec la RCA

Figure 3.2 : Application de la RCA

À notre connaissance, aucune méthode heuristique basée sur la RCA n'a été proposée pour traiter un tel type de problème.

Remarque 3.1

En principe, les relaxations modifient la structure (et parfois la nature) du problème concerné. En effet, la suppression de blocs de contraintes peut induire une transformation radicale, lorsqu'il s'agit de contraintes de couplage.

Dans le cas de PPRV, la relaxation de contrainte de capacité ou de mono-accès n'a pas de telles conséquences. Dans chacune, la transformation induite peut se traduire par une modification des données sans pour autant altérer la nature du problème. De ce fait, le problème demeure un PPRV, comme on le verra par la suite.

La Figure 3.3 illustre l'impact des relaxations étudiées sur l'espace des solutions du PPRV. Ici, une relaxation de contraintes $ax \leq b$ est présentée comme $ax \leq \gamma b$ où $\gamma > 1$ est une constante. La modification apportée sur la partie droite de l'inégalité permet de définir une espace de solutions intermédiaires entre l'espace des solutions du PPRV initiale et l'espace des solutions du PPRV totalement relâché ($\gamma = +\infty$).

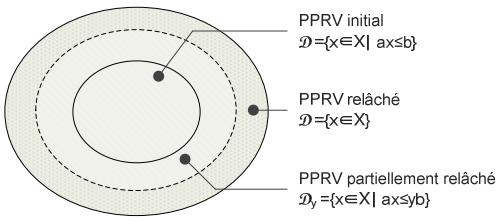


Figure 3.3 : Relaxation du point de vue de l'espace des solutions

3.2 Présentation de la méthode approchée

Nous proposons ici une méthode approchée de type « *route-first, cluster-second* » pour la résolution du PPRV. La méthode repose sur le schéma hybride ILS/VND. Dans un tel schéma, chaque itération commence par la sélection d'un voisin de la meilleure tournée géante trouvée à l'aide d'une procédure de mutation. La tournée voisine est améliorée par une recherche locale avant d'être transformer en une solution du PPRV. Nous utilisons la relaxation de la contrainte de capacité (RCC) dans la recherche locale de la tournée géante, et une adaptation de Split pour l'obtention d'une solution du PPRV à partir d'une tournée géante

Une telle méthode utilise deux espaces de solutions : le premier contient les tournées géantes

et le deuxième les solutions du PPRV. La Figure 3.4 montre le schéma du passage entre les deux espaces. Split établit un passage du premier vers le deuxième. De même, nous disposons aussi d'une procédure qui autorise le passage inverse en concaténant simplement les tournées d'une solution PPRV en une tournée géante. Cela permet une boucle entre deux espaces.

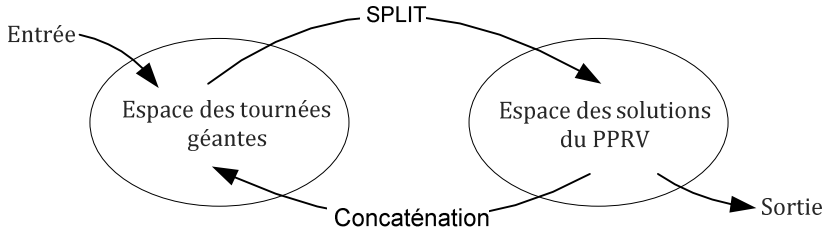


Figure 3.4 : Schéma de passage entre les deux espaces de solutions

Dans cette section, nous commençons par la présentation générale du schéma hybride ILS/VND et l'implémentation de ce schéma dans notre méthode. Ensuite nous présentons la procédure de construction de la tournée géante et l'application de la RCC pour l'amélioration de la tournée géante. Puis nous décrivons en détail l'adaptation de Split et les recherches locales utilisées dans chaque espace de solutions.

3.2.1 Schéma ILS/VND

ILS

ILS (*Iterated Local Search*) a été introduite par Lourenço *et al.* [Lou03]. C'est une méta-heuristique basée sur un schéma itératif dans lequel chaque itération contient deux phases : une phase de perturbation et une phase de recherche locale. La phase de perturbation consiste à choisir aléatoirement un voisin de la solution courante en tenant compte de la liste historique qui contient tous les voisins explorés. Ce voisin est ensuite amélioré par une recherche locale dans la seconde phase. Ces deux phases définissent une stratégie de type diversification/intensification. La phase de perturbation correspond à la diversification et la phase de recherche locale à l'intensification. Ceci permet de trouver des solutions de bonne qualité en parcourant les optimums locaux à l'aide d'un simple schéma répétitif. L'Algorithme 3.1 donne le schéma général de ILS.

Algorithme 3.1 : Schéma de l'ILS

Paramètres d'entrée :
S : solution initiale
Paramètre de sortie :
S : solution améliorée
1 **Début**
2 $S \neg$ améliorer S par la recherche locale ;
3 $historique \neg \{ S \}$;
4
Répéter
5 | $S' \neg$ Perturbation(S, $historique$) ;
6 | $historique \neg historique \cup \{ S' \}$;
7 |
| $S'' \neg$ Améliorer S' par la recherche locale ;
8 | **Si** S'' est meilleur que S **alors**
9 | | $S \neg S''$;
10 | **Fin Si**
11 **Jusqu'à** critère d'arrêt atteint
12 **Fin**

VNS/VND

VNS (*Variable Neighborhood Search*) a été proposée par Mladenović et Hansen [Mla97]. L'idée générale de cette méta-heuristique est d'utiliser itérativement plusieurs structures de voisinage dans la recherche locale. En général, les structures de voisinage sont triées selon la complexité croissante. Au cours de l'exploration de l'espace des solutions, si la structure courante ne permet pas de trouver une solution voisine de meilleure qualité, alors on passe à la structure suivante qui est plus « large ». Si la structure courante fournit un voisin améliorant, alors on recommence l'exploration avec la première structure sur ce voisin. De ce fait, la séquence d'application des structures est « variable », qui dépend en fait la qualité des solutions voisines explorées. Cette caractéristique permet à la recherche locale de sortir des optimums locaux au cours de l'exploration des solutions. Cela vient du fait qu'un optimum local d'une structure de voisinage n'est pas forcément un optimum local pour les autres structures. La solution obtenue à la fin de la méthode est de fait un optimum local pour toutes les structures de voisinage utilisées.

Le schéma classique de la VNS est comme suit : on considère $\mathcal{N}$ l'ensemble des structures de voisinage, $\mathcal{N} = \{\mathcal{N}_k | k = 1, .., k_{max}\}$. On désigne par $\mathcal{N}_k(s)$ l'ensemble des solutions voisines obtenues en appliquant la $k^{ème}$ structure sur la solution s. La VNS reçoit en entrée une solution initiale S souvent obtenue par une heuristique constructive. La solution S devient la solution courante et l'indice de structure k est initialisé à 1. À chaque itération, un voisin S' est sélectionné de manière aléatoire dans $\mathcal{N}_k(S)$, il est amélioré ensuite par la recherche locale. La recherche locale consiste à la recherche d'un optimum local dans $\mathcal{N}_k(S')$ (*i.e.* S' est un optimum local pour la structure $\mathcal{N}_k$ s'il n'existe pas de solution $S'' \in \mathcal{N}_k(S')$ telle que S'' est meilleure que S'). La structure pour l'itération suivante est « variable » (comme mentionné dans le paragraphe précédent) : si S' est meilleure que la solution courante S, alors on fait la mise à jour de S et on utilise ensuite $\mathcal{N}_1$ dans l'itération suivante ; sinon, on passe à $\mathcal{N}_{k+1}$. Ce processus est répété jusqu'à satisfaire le critère d'arrêt. Le schéma de la VNS est donné par l'Algorithme 3.2.

Algorithme 3.2 : Schéma de la VNS

```
    Paramètres d'entrée :
        S : solution initiale
        k_max : nombre de structures de voisinage
    Paramètre de sortie :
        S : solution améliorée
1   Début
2       Tant que critère d'arrêt non atteint faire
3       |   k ← 1 ;
4       |   Tant que k ≤ k_max faire
5       |   |   S' ← Choisir aléatoirement un voisin dans N_k(S) ;   //shaking
6       |   |   Améliorer S' par la recherche locale ;
7       |   |   Si S' est meilleur que S alors
8       |   |   |   S ← S' ;
9       |   |   |   k ← 1 ;
10      |   |   Sinon
11      |   |   |   k ← k +1 ;
12      |   |   Fin Si
13      |   Fin Tant que
14      Fin Tant que
15  Fin
```

La VNS est une méta-heuristique non-déterministe car elle utilise une procédure de *shaking* reposant sur le choix aléatoire d'un voisin dans la structure de voisinage. La VND (*Variable*

Neighborhood Descent) est la simplification du schéma de la VNS en une recherche locale déterministe. Les voisinages sont explorés de manière exhaustive pour obtenir le meilleur voisin. L'Algorithme 3.3 donne le schéma général de la VND.

Algorithme 3.3 : Schéma de la VND

Paramètres d'entrée :
S : solution initiale
k_{max} : nombre de structures de voisinage
Paramètre de sortie :
S : solution améliorée

```
1  Début
2     k ¬ 1 ;
3     Tant que critère d'arrêt non atteint faire
4     |    S' ¬ Choisir le meilleur voisin dans N_k(S) ;
5     |    Si S' est meilleur que S alors
6     |    |    S ¬ S' ;
7     |    |    k ¬ 1 ;
8     |    Sinon
9     |    |    k ¬ k +1 ;
10    |    Fin Si
11    Fin Tant que
12 Fin
```

Le critère d'arrêt de la VND dépend essentiellement de l'indice de la structure de voisinage actuelle. En général, les itérations s'arrêtent lorsqu'aucun voisin amélioré n'est identifié par les k_{max} structures de voisinage. Le nombre de voisinages (k_{max}) est relativement petit dans notre VND (*i.e.* $k_{max} \leq 3$).

Il existe plusieurs variantes/extensions de la VNS comme la VNDS (*Variable Neighborhood Decomposition Search*), la SVNS (*Skewed VNS*) ou encore la PVNS (*parallel VNS*) et la VSS (*Variable Search Space*). La VNDS consiste à réduire l'espace de solutions voisines des structures de voisinage, en disant qu'une structure n'agit que sur un sous-ensemble d'attributs de la solution courante, et chaque structure agissant sur un sous-ensemble différent. La SVNS consiste à autoriser l'acceptation d'un voisin non amélioré dans la recherche locale et la PVNS consiste à lancer parallèlement la VNS. Pour avoir plus de détails sur ces variantes/extensions, on peut se référer à [Mla97, Han03].

ILS/VND

ILS/VND est une hybridation entre ILS et VND. Le principe d'une telle hybridation consiste à remplacer la recherche locale de ILS par VND. L'objectif est de profiter les points forts de chaque méthode, qui garde la diversification de la part de ILS et renforce l'intensification de la recherche locale par VND.

3.2.2 Schéma général de la méthode proposée

La méthode proposée utilise le schéma ILS/VND, qui est donné par l'Algorithme 3.4. Dans un tel schéma, on commence par la construction d'une tournée géante T en utilisant la procédure **Construire_tournée** (voir la section 3.2.3). Puis on l'améliore par **Recherche_locale_1** qui est reposée sur le schéma VND (voir la section 3.2.6.3). Dans la boucle principale de cette méthode, on sélectionne aléatoirement un voisin de la tournée courante à l'aide d'une **Perturbation** (voir la section 3.2.4). On applique **Recherche_locale_1** sur ce voisin avant de lancer la procédure **Transformer_solution** (qui intègre Split et la procédure **Recherche_locale_2**) pour générer une solution du PPRV à partir de la tournée et ensuite l'améliorer (voir la section 3.2.5). L'algorithme se termine lorsque le critère d'arrêt est atteint.

Algorithme 3.4 : Schéma de l'ILS/VND

Paramètres d'entrée :
G : instance
$grain$: graine du générateur aléatoire
It_{max} : nombre d'itérations
Paramètre de sortie :
S : solution du PPRV

Début
  Mettre $Grain$ comme valeur de graine aléatoire;
  $T_0 \leftarrow$ Construire_tournée(G, $grain$) ; *// voir l'Algorithme 3.7*
  $T \leftarrow$ Recherche_locale_1 (G, T_0) ;
  $historique \leftarrow \{T_0\}$;
  $S \leftarrow \emptyset$;
  $it \leftarrow 0$;
  Répéter
    $T' \leftarrow$ Perturbation(G, T,$historique$) ;
    $historique \leftarrow historique \cup \{T'\}$;
    $T' \leftarrow$ Recherche_locale_1 (G, T') ; *// voir l'Algorithme 3.10*
    Si T' est une tournée réalisable **alors**
      $S' \leftarrow$ Transformer_solution(G, T'); *// voir l'Algorithme 3.9*
      Si S' est meilleure que S **alors**
        $S \leftarrow S'$;
      Fin Si
      Si T' est meilleure que T **alors**
        $T \leftarrow T'$
      Fin Si
    Fin Si
    $it \leftarrow it$ +1;
  Jusqu'à $it \geq It_{max}$ et ∃ solution réalisable
Fin

3.2.3 Procédure de construction d'une tournée géante

Nous utilisons deux méthodes de construction différentes dans la procédure **Construire_tournée** : l'heuristique *Nearest Neighbour* randomisée et l'heuristique de construction de [Zha09] modifiée. Ces deux heuristiques autorisent la relaxation de contrainte de capacité dans la tournée géante construite.

3.2.3.1 Heuristique *Nearest Neighbour* randomisée

L'heuristique *Nearest Neighbour* randomisée est une méthode d'insertion itérative. La construction commence par la tournée vide. À chaque itération un sommet est sélectionné puis inséré à la fin de la tournée. Afin d'augmenter la possibilité d'obtenir une tournée réalisable de meilleure qualité, nous utilisons une liste de candidats (dite RCL : *restricted candidates list*). Une telle liste contient l'ensemble de N sommets réalisables (i.e. les sommets non apparus dans la tournée et qui peuvent être insérés **à la fin** de la tournée sans violer la contrainte de capacité). Le nombre N est le minimum de 10 et du nombre de sommets réalisables. Lorsque le nombre de sommets réalisables est supérieur à 10, nous prenons les 10 meilleurs. Si aucun sommet réalisable n'est disponible, alors la RCL reçoit les N sommets les plus proches du dernier sommet de la tournée. La valeur N est dans ce cas le minimum de 10 et du nombre de sommets non insérés. On sélectionne ensuite aléatoirement un sommet dans la RCL pour réaliser l'insertion. La tournée construite peut être non-réalisable. Cette méthode est donnée par l'Algorithme 3.5.

Algorithme 3.5 : Heuristique *Nearest Neighbour* randomisée (NN_Rand)

```
    Paramètres d'entrée :
        G : instance
        Grain : graine du générateur aléatoire
    Paramètre de sortie :
        T : tournée géante
1   Début
2       Mettre grain comme valeur de graine aléatoire;
3       T ¬ {0};                          // premier élément dans la tournée : le dépôt
4       ls ¬ V\{0} ;
5       RCL ¬ ∅ ;                         // liste de candidatures/sommets pour être inséré à la fin de S0
6       Tant que ls ≠ ∅ faire
7       |   lr ¬ ensemble de sommets réalisables (dans ls)
8       |   Si lr ≠ ∅ alors
9       |   |   n ¬ min(10, |lr|) ;
10      |   |   RCL ¬ les x sommets les plus proches du dernier élément de T dans lr;
11      |   Sinon
12      |   |   n ¬ min(10, |ls|) ;
13      |   |   RCL ¬ les x sommets les plus proches du dernier élément de T dans ls;
14      |   Fin Si
15      |
16      |   v ¬ tirer aléatoirement un élément dans RCL ;
17      |   T ¬ T ∪{v};                   // insérer le sommet v à la fin de S0
18      |   ls ¬ ls \ {v};
19      |   RCL ¬ ∅ ;
20      Fin Tant que
21  Fin
```

3.2.3.2 Modification de l'heuristique de Zhao *et al.*

Cette méthode de construction a été proposée par Zhao et al. [zha09] (notée NN2). Elle est basée aussi sur l'heuristique NN (*Nearest Neighbour*). Elle peut être considérée comme une variante de la méthode précédente car la différence vient de la façon de sélectionner un sommet et aussi de la position d'insertion. Dans NN2, lorsque la RCL (pour l'insertion **à la fin** de la tournée) est non vide, on sélectionne le sommet qui possède la plus grande demande (en valeur absolue) et on l'insérer à la fin de la tournée. Si la RCL est vide, alors nous cherchons les sommets non insérés qui peuvent être inséré légalement (*i.e.* sans violer la contrainte de capacité) sur une **position quelconque** de la tournée. Si nous en avons trouvé, alors on applique une méthode alternative pour la sélection d'un sommet ; sinon on recommence la construction avec une tournée vide. Notons que NN2 produit des tournées réalisables. Dans une version modifiée de NN2 (notée NN2_modif), au lieu de recommencer la construction, nous insérons le sommet le plus proche du dernier de la tournée et continuons les itérations. Cette modification permet de construire aussi des tournées non réalisables. Le détail est donné par l'Algorithme 3.6.

Algorithme 3.6 : Heuristique NN2 modifiée (NN2_modif)

```
    Paramètres d'entrée :
        G : instance
        graine : graine du générateur aléatoire
    Paramètre de sortie :
        T : tournée géante
1   Début
2       Mettre graine comme valeur de graine aléatoire;
3       T ¬ {0};
4       ls ¬ ensemble de sommets sauf le dépôt dans G ;
5       v ¬ tirer aléatoirement un élément dans ls;        //premier site à visiter
6       T ¬ T ∪{v};
```

```
7       ls ← ls \ {v};
8       Tant que ls ≠ ∅ faire
9       |   lr ← ensemble de sommets réalisables (dans ls) pour l'insertion à la fin ;
10      |   Si lr ≠ ∅ alors
11      |   |   v ← le sommet avec la plus grande demande (en valeur absolue) dans lr ;
12      |   |   T ← T ∪{v};  // insertion à la fin
13      |   Sinon
14      |   |   lrq← ensemble de sommets réalisables (dans ls) pour l'insertion
        |   |        à une position quelconque ;
15      |   |   Si lrq ≠ ∅ alors
16      |   |   |   v ← sélectionner le sommet le moins coûteux à l'insertion (dans lrq) avec une
        |   |   |       probabilité 0,7 et tirer aléatoirement un sommet dans lrq sinon ;
17      |   |   |   insérer le sommet v en une position réalisable dans la tournée ;
18      |   |   Sinon
19      |   |   |   v ← tirer aléatoirement un élément dans ls ;
20      |   |   |   T ← T ∪{v};
21      |   |   FinSi
22      |   Fin Si
23      |   ls ← ls \ {v};
24      Fin Tant que
25  Fin
```

3.2.3.3 Schéma de la procédure de construction d'une tournée géante

La procédure de construction utilise *Nearest Neighbour* randomisée (**NN_Rand**) et NN2 modifiée (**NN2_Modif**). La sélection de chaque méthode est équiprobable. Le schéma de **Construire_tournée** est donné par l'Algorithme 3.7.

Algorithme 3.7 : Schéma de construction d'une tournée géante (Construire_tournée)

```
    Paramètres d'entrée :
        G : instance
        graine : graine du générateur aléatoire
    Paramètre de sortie :
        T : tournée géante
1   Début
2       p ← tirer aléatoirement un nombre entre 1 et 100 ;
3       Si p ≤ 50 alors
4       |   T ← NN_Rand(G, graine) ;
5       Sinon
6       |   T ← NN2_Modif(G, graine) ;
7       Fin Si
8   Fin
```

3.2.4 Procédure de perturbation

La procédure de perturbation utilise la structure de voisinage « *k-positions* » : deux tournées sont dites distantes de *k-positions* si la somme des différences sur l'ordre de visite de chaque sommet est inférieure ou égale à k, $k \in \mathbb{N}$. Le principe de perturbation consiste donc à choisir aléatoirement un voisin de type *k-positions* de la tournée géante en relaxant la qualité associée. Ceci permet d'explorer des voisins très diversifiés. Tous les voisins explorés sont stockés dans la mémoire, afin d'éviter des doublons. Notons qu'un voisin exploré est une tournée géante réalisable.

Une telle procédure de perturbation repose sur un modèle PLNE. Plus précisément, ce modèle correspond à un problème d'affectation, qui est défini comme suit : étant donné l'ensemble de sommets $V = \{1, .., |V|\}$, avec une demande D_i associée à chaque sommet i. Soit $\omega =$

$\{T_0, T_1, .., T_n\}$ l'ensemble des tournées géantes déjà traitée (chaque tournée géante est un vecteur de taille $|V|$). Le vecteur S de taille $|V|$ stocke l'ordre de visite des sommets de la nouvelle tournée. L'objectif est de trouver une affectation qui donne l'attribution des sommets aux cases du vecteur S, tout en respectant les contraintes suivantes :

- (c3.1) chaque sommet est affecté à une position/case du vecteur S;
- (c3.2) chaque position/case de S accepte un sommet ;
- (c3.3) l'ordre d'affectation doit respecter la capacité de l'agent ;
- (c3.4) la tournée explorée est un voisin de type *k-positions* de la tournée courante (*i.e.* la meilleure tournée) ;
- (c3.5) la tournée explorée est différente de toute les tournées stockées dans $\boldsymbol{\omega}$.

Pour modéliser un tel problème d'affectation, nous avons utilisé deux types de variables :

- $x_i^k \in \{0, 1\}$: =1 si le sommet i est sur la position k, 0 sinon ;
- $q_k \geq 0$: la charge de l'agent en quittant le sommet à la position k.

Les données d'entrée pour le problème sont comme suit :

- $V = \{1, .., |V|\}$: ensemble des sommets ;
- $Q \in \mathbb{N}$: capacité de l'agent ;
- $D_i \in \mathbb{Z}$: demande d'un sommet i, $i = 1, .., |\mathrm{V}|$;
- T^* : la meilleure tournée ;
- $\omega = \{T_0, T_1, .., T_n\}$: ensemble des tournées explorées ;
- α : coefficient définissant la distance entre des tournées voisines de type *k-positions* ;
- β : coefficient définissant la différence par rapport aux tournées stockées dans ω.

La formulation est comme suit :

$$
(\text{PL } 3.1)\begin{cases}
\text{Minimiser} \displaystyle\sum_{i \in V} \sum_{k=1..|V|} x_i^k & & (3.1-1)\\
\text{s.c.} & & \\
\displaystyle\sum_{k=1...|V|} x_i^k = 1 & \forall i \in V & (3.1-2)\\
\displaystyle\sum_{i \in V} x_i^k = 1 & \forall k = 1 \ldots |V| & (3.1-3)\\
q_k - q_{k-1} - \displaystyle\sum_{i \in V} D_i x_i^k = 0 & \forall k = 2 \ldots |V| & (3.1-4)\\
\displaystyle\sum_{k=1...|V|} x_{T^*[k]}^k \geq \lfloor \alpha \cdot |V| \rfloor & & (3.1-5)\\
\displaystyle\sum_{k=1...|V|} x_{T_j[k]}^k \leq |V| - \beta & \forall\, j = 0 \ldots |\omega| & (3.1-6)\\
x_i^k \in \{0, 1\} & \forall i \in V\,, k = 1 \ldots |V| & (3.1-7)\\
0 \leq q_k \leq Q & \forall k = 1 \ldots |V| & (3.1-8)
\end{cases}
$$

Dans la formulation (PL 3.1), la fonction objectif $(3.1-1)$ est une constante puisqu'on cherche seulement une tournée satisfaisant les contraintes ; Les contraintes $(3.1-2)$ et $(3.1-3)$ correspondent respectivement aux contraintes (c3.1) et (c3.2) ; la contrainte $(3.1-4)$ est la conservation de la charge de l'agent dans la tournée géante ; $(3.1-5)$ et $(3.1-6)$ correspondent respectivement aux contraintes (c3.4) et (c3.5). Dans la contrainte $(3.1-5)$, la valeur $(|V| - \lfloor \alpha \cdot |V| \rfloor)$ définit la distance entre deux tournées voisines de type *k-positions*.

Dans notre configuration, la valeur de α dépend de la taille de ω et de la taille d'instance : $\alpha = 0{,}5$ si $|\omega| \leq 10$; $\alpha = 0{,}8$ pour $|V| \leq 60$ et $\alpha = 0{,}9$ pour $|V| > 60$. Dans la contrainte $(3.1-6)$, la valeur de β est fixé ici à 3. Les contraintes $(3.1-7)$ et $(3.1-8)$ définissent les variables. En pratique, le temps de calcul pour la résolution des petites instances est quasi-instantané avec un solveur comme CPLEX ou Gurobi.

3.2.5 Transformation d'une tournée géante en une solution du PPRV

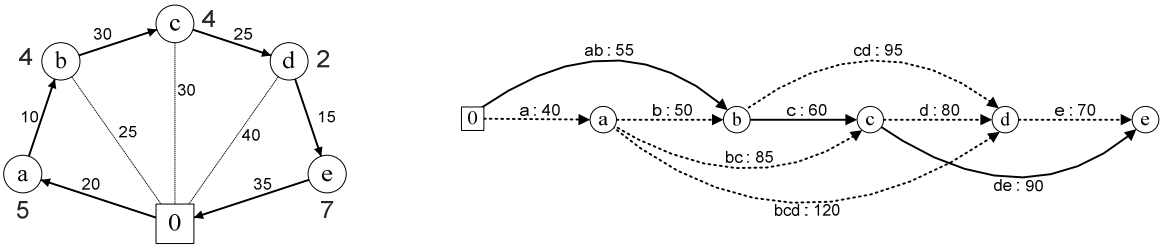
La transformation d'une tournée géante en une solution PPRV utilise une adaptation de Split. Dans cette partie, on commence par la présentation de la version originale de Split, avant de donner la description de la version adaptée au PPRV. La procédure de transformation sera présentée à la fin de cette partie.

3.2.5.1 Split

Split a été popularisée par Prins [Prin04] dans sa méthode de type « *route-first, cluster-seconde* » pour la résolution d'un problème de tournées de véhicules. Ce type de méthodes a été introduit initialement par Beasley [Bea83]. Une telle méthode se présente naturellement en deux phases : la phase *route-first* consiste à construire une tournée géante en minimisant le coût associé et la phase *cluster-seconde* crée une solution (avec multi-véhicules) en découpant la tournée géante en un ensemble de trajectoires séquentielles. Split est une procédure de découpage optimale du tour géant, respectant l'ordre de visite des sommets de la tournée géante, ce qui permet l'utilisation d'algorithmes rapides.

Notons que le principe de Split a déjà été mentionné pour le traitement du problème de tournées de véhicules classique (CVRP) dans la section 1.2.2.1. On reprend l'exemple de la section 1.2.2.1 (tiré de [Prin04]). La Figure 3.5 (a) illustre une tournée géante $T = (0, a, b, c, d, e, 0)$, dans laquelle nous avons les informations liées à la demande des clients et au coût de certains arcs pour appliquer Split. La capacité de véhicules est fixée à 10 et la durée maximale d'une tournée n'est pas limitée.

Pour trouver la meilleure combinaison de découpages sur la tournée géante, Split utilise un graphe auxiliaire $G' = (X, A)$ où X contient tous les sommets de V, et A est l'ensemble des arcs correspondant aux sous-séquences de T. Les sommets de X sont indexés de 1 à $|X|$ selon l'ordre de présence dans T. Un arc (i, j) dans G' représente une coupe (correspondant à une tournée de véhicule) réalisable contenant la séquence des clients $\{T_{i+1}, T_{i+2}, \ldots, T_j\}$, où $1 \leq i(\text{et } j) \leq |X|$ et $i < j$. La Figure 3.5 (b) montre le graphe auxiliaire lié à T. Par exemple l'arc (b, d) avec le libellé « $cd: 95$ » correspond à la tournée $(0, c, d, 0)$ avec le coût 95.



(a) graphe avec la présence d'une tournée géante

(b) graphe auxiliaire pour Split

Figure 3.5 : Split sur un problème de tournées de véhicules

Etant donné l'ordre induit par T, une solution du CVRP correspond à un chemin du sommet T_1 au $T_{|X|}$ sur le graphe auxiliaire. Le but de Split est donc de trouver le plus court chemin dans G' qui représente la meilleure solution pour la tournée géante considérée. Cette évaluation est raisonnable car G' est acyclique [Prin04]. Dans cet exemple, le plus court chemin dans la Figure 3.5 (b) est $(0, b, c, e)$ de coût 205. Ce chemin est converti ensuite en une solution avec 3 tournées de véhicules : $(0, a, b, 0)$, $(0, c, 0)$ et $(0, d, e, 0)$. Cela donne le meilleur découpage pour la tournée géante T de la Figure 3.5 (a).

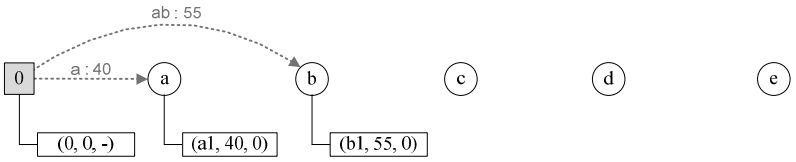
Notons que le graphe auxiliaire est construit par Split à partir de la tournée géante. Au lieu d'attendre la fin de construction avant de lancer un algorithme de plus court chemin, Split propose une stratégie qui lui permet de chercher le plus court chemin en même temps que la construction du graphe. C'est de la programmation dynamique (voir la section 1.2.1.2), sauf que Split respecte l'ordre de visite des sommets (de la tournée géante). À partir de l'ensemble des sommets (fournis par la tournée géante), le graphe auxiliaire est construit en ajoutant successivement les arcs. La construction utilise une boucle parcourant l'ensemble des sommets X (selon l'ordre dans la tournée géante). Chaque itération consiste à identifier les arcs sortant du sommet actuel et à propager une structure d'« étiquette » qui stocke les informations liées aux chemins potentiels après chaque création d'arc. Il dispose donc d'une liste d'étiquettes sur chaque sommet.

Une étiquette est définie par un triplé (identifiant, coût partiel, étiquette père). L'identifiant est utilisé pour distinguer les étiquettes d'une même liste. Le coût partiel est la somme des coûts des tournées parcourues jusqu'au sommet actuel. L'étiquette père indique l'étiquette ayant permis d'obtenir l'étiquette courante. Cette information est nécessaire pour retrouver le découpage optimal à partir de la meilleure étiquette finale.

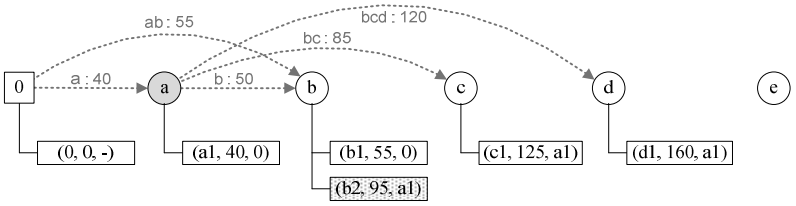
La création des étiquettes a lieu tout au long du parcours du graphe auxiliaire : après la création d'un arc (i, j), Split parcourt la liste d'étiquettes liée au sommet origine de cet arc afin de propager les étiquettes vers le sommet destination. La propagation des étiquettes utilise une règle de dominance qui permet de filtrer des étiquettes les moins intéressantes. La règle de dominance consiste à comparer le coût partiel entre deux étiquettes : celle avec la plus petite valeur domine l'autre. Une nouvelle étiquette est ajoutée si elle n'est pas dominée par celles dans la liste. Au contraire, l'élimination des étiquettes aura lieu lorsque la nouvelle domine les autres dans la liste. Split ne garde que les « meilleures » étiquettes.

La Figure 3.6 montre la construction du graphe auxiliaire qui se trouve sur la Figure 3.5 (b). La Figure 3.6 (a) est le graphe au début des itérations. La Figure 3.6 (b) donne le graphe après l'itération 1. On parcourt dans cette itération le premier sommet (dépôt), ayant le label initial (0, 0, -). Deux arcs sont considérés : $(0, a)$ et $(0, b)$. Ils conduisent aux étiquettes $(a1, 40, 0)$ sur le sommet a et $(b1, 55, 0)$ sur le sommet b. La Figure 3.6 (c) illustre le graphe après l'itération 2. Trois arcs sortant du sommet a sont alors étudiés et la nouvelle étiquette au sommet b est rejetée par la règle de dominance. Dans l'itération suivante (voir Figure 3.6 (d)), on considère les deux arcs sortant du sommet b. La nouvelle étiquette domine l'ancienne. Les itérations s'arrêtent après avoir parcouru l'ensemble des sommets. Pour identifier le plus court chemin, nous prenons l'étiquette dominant sur le dernier sommet, puis remontons les étiquettes pères jusqu'au dépôt. Le chemin parcouru correspond au plus court chemin (voir Figure 3.6 (e)). Dans cet exemple, le plus court chemin trouvé est $(0, b, c, e)$.

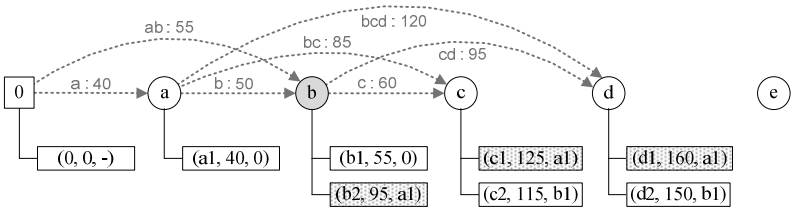
(a) graphe auxiliaire au départ



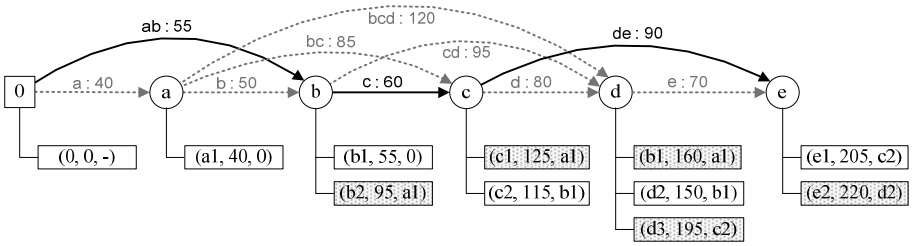
(b) l'itération 1 : sur le sommet 0 (dépôt)



(c) l'itération 2 : sur le sommet a



(d) l'itération 3 : sur le sommet b



(e) graphe auxiliaire avec les étiquettes à la fin des itérations

Figure 3.6 : Recherche du plus court chemin à l'aide des étiquettes

Une fois le plus court chemin trouvé sur le graphe auxiliaire, nous pouvons réaliser le découpage sur la tournée géante. La Figure 3.7 illustre le découpage de la tournée géante par le plus court chemin $(0, b, c, e)$.

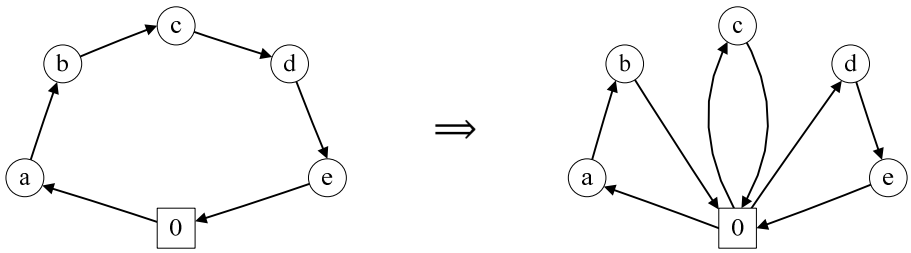


Figure 3.7 : Découpage avec Split

Dans le cas où on doit prendre en compte une limite en ressources (nombre de véhicules, durée maximale…), il faut intégrer dans chaque étiquette la quantité restante en ressources. Ceci complexifie les règles de dominance mais ne change pas le principe général de la méthode.

3.2.5.2 Adaptation de Split au PPRV

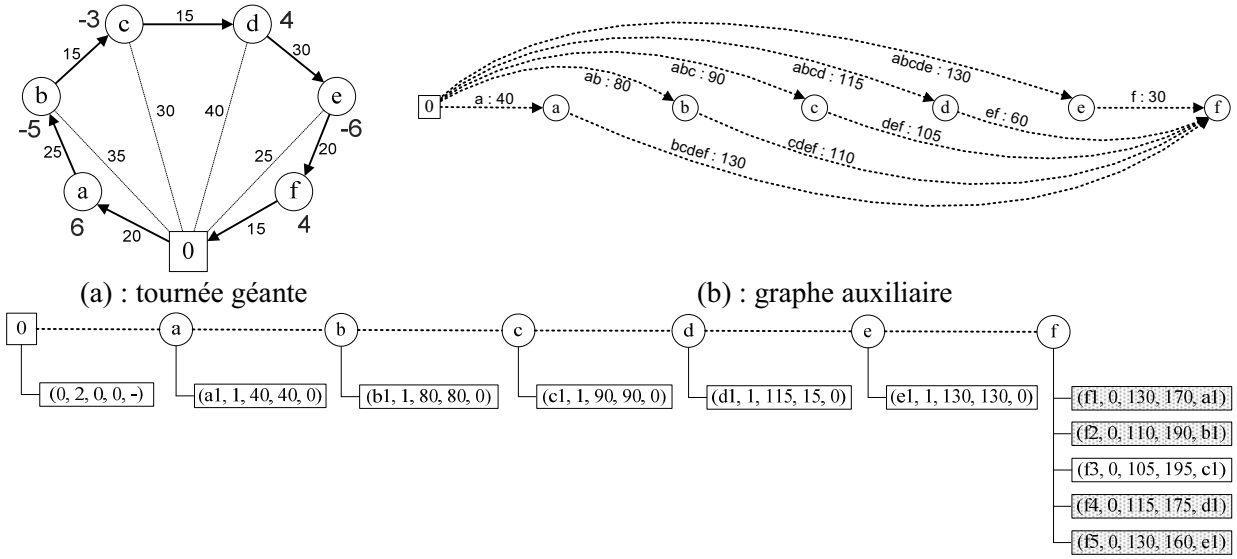
Nous présentons les modifications apportées à Split pour qu'il puisse être adapté au PPRV. Ces modifications se trouvent essentiellement dans la partie de la construction du graphe auxiliaire : un arc doit respecter les contraintes du PPRV (contraintes de capacité, etc..) ; la structure d'étiquettes et la règle de dominance ont aussi été modifiées en considérant le critère d'optimisation du PPRV. Notons que le nombre d'agents K est fixé dans le PPRV. Une solution générée contient donc K tournées. En plus de ces modifications, nous respectons le principe de Split décrit dans la partie précédente.

Une étiquette (modifiée) est représentée ici sous la forme de quintuplé : (identifiant, nombre d'agents libres, coût partiel, durée maximale, étiquette père). Parmi ces cinq attributs, l'identifiant, le coût partiel et l'étiquette père portent la même signification que précédemment. Le nombre d'agents libres décrit le nombre de propagation restant pour l'étiquette : une étiquette ne peut plus être propagée lorsque ce nombre est nul. La durée maximale contient la plus grande durée des tournées d'agent parcourues.

Dans une liste des étiquettes liée à un sommet, on regroupe les étiquettes par classe selon le nombre d'agents libre. Au moment de la propagation des étiquettes, la règle de dominance est appliquée dans chaque classe. La dominance tient compte de la durée maximale et du coût partiel. Dans une classe, une étiquette E_1 domine une étiquette E_2 si l'une des deux conditions suivantes est satisfaite :

- Condition 1 : la durée maximale de E_1 est plus petite que celle de $\boldsymbol{E_2}$;
- Condition 2 : la durée maximale de E_1 est égale à celle de E_2 et le coût partiel de E_1 est plus petit que celui de E_2.

La Figure 3.8 donne un exemple de Split sur le PPRV avec 2 agents. La capacité est égale à 10. La tournée géante est présentée dans la Figure 3.8 (a). Le graphe auxiliaire est illustré par la Figure 3.8 (b). On considère ici que les contraintes du PPRV sont respectées par tous les arcs. Les listes des étiquettes liées aux sommets sont présentées dans la Figure 3.8 (c). Nous nous intéressons à l'étiquette dominante du dernier sommet (ici c'est l'étiquette f). À partir d'elle, on obtient le plus court chemin en remontant au dépôt. Le chemin trouvé dans cet exemple est $(0, c, f)$.



(c) : listes d'étiquettes sur les sommets

Figure 3.8 : Application de Split pour le PPRV avec 2 agents

L'adaptation de Split au PPRV est décrite dans l'Algorithme 3.8. L'ensemble L stocke les listes des étiquettes. La liste liée au $i^{ème}$ sommet de la tournée géante est notée L_i et la $k^{ème}$ étiquette dans la liste du sommet i est notée $L_i^k = (I_i^k,\ A,\ C_{ptl}, D_{max}, P_i^k)$. La signification des attributs du quintuplé a déjà été présentée dans la définition d'étiquette (modifiée). Soit T' la tournée d'agent correspondant à l'arc (i, j) dans le graphe auxiliaire. Alors la réalisabilité de T' est donnée par la fonction tol : on ne s'intéresse qu'aux tournées réalisables. D'autre part, on ne stocke pas la topologie du graphe auxiliaire (plus précisément ses arcs) durant l'exécution de la procédure.

Algorithme 3.8 : Procédure Split adaptée (Split)

Paramètres d'entrée :
G : instance
T : tournée géante
K : nombre d'agents

Paramètre de sortie :
S : solution du PPR

1 **Début**
2 Soit L l'ensemble des listes (d'étiquettes) liées aux sommets ;
3 Ajouter l'étiquette (0, K, 0, 0, -1) dans L_1 ; // *liste d'étiquette pour le premier sommet du T*
4 **Pour** i = 1 à $|T|$ - 1 **faire** // *parcourir les sommets*
5 | $j \leftarrow i + 1$;
6 | $stop \leftarrow$ faux ;
7 | $T' \leftarrow$ {dépôt} ;
8 | **Tant que** non $stop$ **et** $j \leq |T|$ **faire**
9 | | $T' \leftarrow T' \cup T_j$;
10 | | **Si** $tol(T') = 0$ **alors** // *T' est réalisable*
11 | | | $D \leftarrow$ durée de la tournée T' ;
12 | | | **Pour** toutes les étiquettes $L_i^k : (I_i^k, A, C_{ptl}, D_{max}, P_i^k)$ dans L_i **faire**
13 | | | | **Si** $A > 0$ **et** L_i^k non dominé par les étiquettes dans L_j **alors**
14 | | | | | $p \leftarrow$ taille de la liste L_j ;
15 | | | | | Ajouter l'étiquette $(I_j^{p+1}, A - 1, C_{ptl} + D, \max(D_{max}, D), I_i^k)$ dans L_j ;
16 | | | | **Fin Si**
17 | | | **Fin Pour**
18 | | **Sinon**
19 | | | $stop \leftarrow$ vrai ;
20 | | **Fin Si**
21 | **Fin Tant que**
Fin Pour
22
23 $l \leftarrow$ la meilleure étiquette dans la liste $L_{|T|}$;
24 $j \leftarrow |T|$;
25 **Tant que** le père de $l \neq$ - 1 **faire**
26 $l \leftarrow$ père de l ;
27 $i \leftarrow$ sommet dans T lié à l ;
28 $T' \leftarrow$ {dépôt} ∪ séquence $\{0, T_{i+1} \ldots T_j, 0\}$;
29 $S \leftarrow S \cup T'$;
30 $j \leftarrow i$;
31 **Fin Tant que**
32 **Fin**

3.2.5.3 Procédure de transformation

La procédure **Transformer_solution** utilise deux espaces de solutions (voir la Figure 3.4) : celui des tournées géantes et celui des solutions du PPRV. Elle dispose d'un passage dans les deux sens entre ces deux espaces. L'objectif est de trouver la meilleure solution du PPRV à partir

d'une tournée géante. L'Algorithme 3.9 donne le schéma de **Transformer_solution**. Une tournée géante est d'abord découpée en K morceaux en appliquant Split, K est le nombre d'agents. Chaque morceau correspond à une tournée d'agent et l'ensemble de ces tournées donne une solution du PPRV. La solution obtenue est ensuite améliorée par **Recherche_locale_2**. Puis, nous utilisons la procédure **Concaténation** pour concaténer les tournées en une nouvelle tournée géante. On revient donc dans l'espace des tournées géantes grâce à la procédure **Concaténation**. On applique de nouveau Split sur la tournée géante obtenue. La boucle de transformations se termine au bout de $n1$ itérations.

Algorithme 3.9 : Procédure de génération d'une solution PPRV (Transformer_solution)

```
    Paramètres d'entrée :
        G : instance
        T : tournée géante
        K : nombre d'agents
        n1 : nombre d'itérations (par défaut 5)
        n2 : nombre d'échanges dans la mutation (par défaut 10)
    Paramètre de sortie :
        S : solution du PPRV
1   Début
2       T' ¬ T ;
3       Pour i = 1 à n1 faire
4       |   S' ¬ Split(G,T',K) ;                    // voir l'Algorithme 3.8
        |   S' ¬ Recherche_locale_2 (G, S', K) ;    // voir l'Algorithme 3.11
5       |
6       |   Si f(S') < f(S) alors
7       |   |   S ¬ S' ;
        |   Fin Si
8       |
9       |   T' ¬ Concaténation(G, S', K) ;          // concaténation
10      |   T' ¬ swap_intra (T', n2) ;              // mutation
11      Fin Pour
12  Fin
```

3.2.6 Recherches locales

Nous utilisons deux recherches locales dans la méthode proposée. La première, **Recherche_locale_1**, cherche à améliorer la tournée géante et la deuxième, **Recherche_locale_2**, vise à améliorer les solutions du PPRV dans la procédure **Transformer_solution**. Deux types d'opérateurs sont employés : intra-tournée et inter-tournée. La première recherche locale n'utilise que des opérateurs de type intra-tournée reposant sur la relaxation de contrainte de capacité (RCC). La deuxième utilise les deux types d'opérateurs sans relaxation.

On commence par la présentation des structures de voisinage et des opérateurs de recherche locale. Puis nous décrivons les deux recherches locales.

3.2.6.1 Structures de voisinage et opérateurs d'amélioration locale

Une structure de voisinage est définie comme suit : soit $\mathfrak{D}$ l'espace des solutions, une structure de voisinage est une fonction $\mathcal{N}$ qui permet d'explorer un sous-espace de $\mathfrak{D}$ à partir d'une solution S en appliquant un mouvement prédéfini (souvent sur les arcs ou sur les sommets). Elle est caractérisée essentiellement par le type de mouvement considéré.

On note $\mathcal{N}(S)$ l'ensemble des solutions associées au voisinage de la solution S. Une solution $S' \in \mathcal{N}(S)$ est donc un voisin de S. Soit f la fonction d'évaluation. La solution S est dite optimum local relativement à $\mathcal{N}$ si la condition suivante est satisfaite :

$$f(S) \leq f(S') \qquad \forall S' \in \mathcal{N}(S)$$

Un opérateur est un mécanisme reposant sur une structure de voisinage avec un critère d'acceptation. Le but d'un opérateur est de trouver l'optimum local en explorant l'espace des solutions liée à sa structure de voisinage. La démarche d'un opérateur est décrite comme suit : soit S la solution de départ et soit $\mathcal{N}$ la structure de voisinage pour l'opérateur. À chaque itération, nous cherchons à remplacer S par son voisin dans $\mathcal{N}(S)$ qui satisfait la règle de remplacement. Les itérations se terminent lorsque la solution S ne peut plus être remplacée/améliorée à travers $\mathcal{N}$ (*i.e.* S est l'optimum local relatif à $\mathcal{N}$).

Notons qu'avec la RCC, la règle de remplacement tient compte à la fois de la qualité et de la réalisabilité des deux voisins. Soit tol la fonction mesurant la réalisabilité d'une solution et tl le seuil de relaxation. Alors la solution S est remplacée par son voisin $S' \in \mathcal{N}(S)$ si la condition suivante est satisfaite :

$$f(S') < f(S) \;\; et \; tol(S') \leq tl$$

Afin de guider la solution relâchée courante vers les solutions réalisables dans un opérateur avec la RCC, nous appliquons la stratégie suivante : soit Δ une valeur prédéfinie et S' la solution acceptée

- $tl \leftarrow \text{tl} - \Delta$ si $tol(S') > 0$;
- $tl \leftarrow 0$ si S' est réalisable (*i.e.* $tol(S') = 0$).

3.2.6.2 Opérateurs utilisés

Dans les recherches locales, nous utilisons essentiellement quatre opérateurs intra-tournée et deux opérateurs inter-tournée :

- ***2-opt*** **intra-tournée** (2_opt_intra) : les mouvements sont réalisés à l'intérieur d'une tournée. Un voisin est obtenu en supprimant deux arcs et en en ajoutant deux autres. La Figure 3.9 illustre une opération de type *2-opt* : les arcs (a, b) et (c, d) sont supprimés et remplacés par (a, c) et (b, d) ; la séquence de b à c est inversée dans l'opération. Cet opérateur intègre la relaxation de capacité et seuls les voisins satisfaisant la règle de remplacement sont considérés. L'opérateur *2-opt* offre un très bon rapport qualité / complexité – l'exploration du voisinage est en $o(n^2)$ et nous l'appliquons quelle que soit la taille des instances.

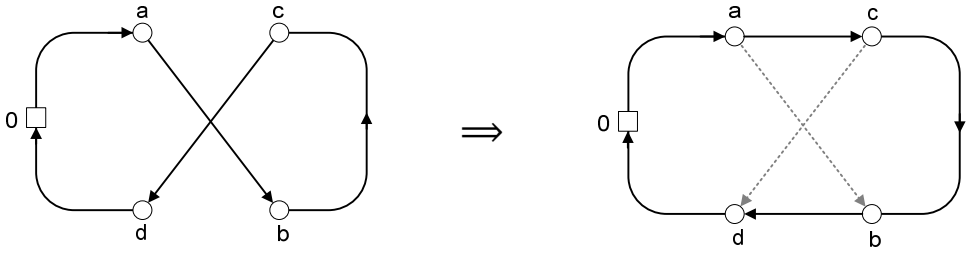
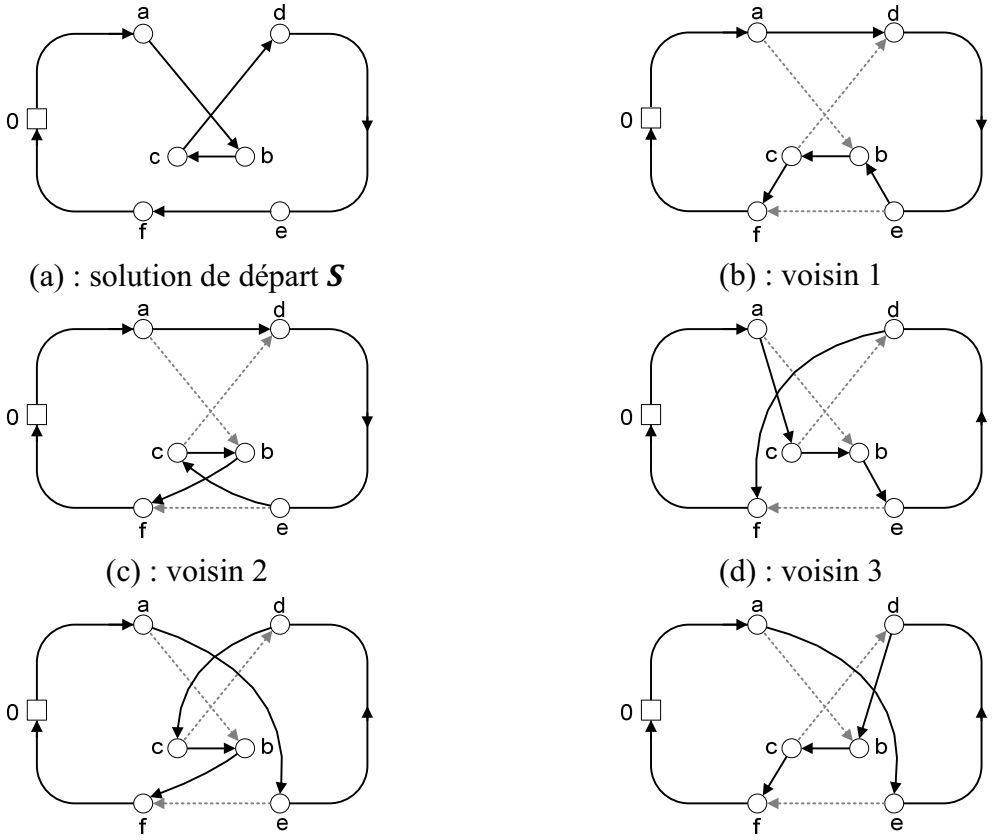


Figure 3.9 : Mouvement de l'opérateur *2-opt* intra-tournée

- ***3-opt*** **intra-tournée** (3_opt_intra) : cet opérateur *3-opt* interne repose sur une structure de voisinage similaire à *2-opt*. Ce type d'opérateurs peut être généralisé en *k-opt* où *k* est le nombre maximum d'arcs à supprimer et à ajouter ensuite à chaque opération. Le *3-opt* inclut donc les mouvements *2-opt*. L'opération de suppression / ajout de 3 arcs peut se traduire par plusieurs combinaisons. La suppression de 3 arcs peut générer 5 voisins et la

Figure 3.10 illustre ces 5 possibilités. L'exploration du voisinage *3-opt* intra-tournée est en $o(n^3)$. C'est supérieur au *2-opt* intra-tournée et nous ne l'utilisons que pour les petites instances (moins de 60 sommets). Cet opérateur intègre également la relaxation de contrainte de la capacité.



(a) : solution de départ ***S*** (b) : voisin 1

(c) : voisin 2 (d) : voisin 3

(e) : voisin 4 (f) : voisin 5

Figure 3.10 : Mouvements de l'opérateur *3-opt* intra-tournée à 3 arcs

- ***Or-opt*** **intra-tournée** (or_opt_intra) : le *Or-opt* est un opérateur proposé initialement par Or [Or76] et popularisé par Golden et Stewart [Gol85]. C'est un cas particulier de *3-opt* intra-tournée qui ne considère que le voisin 3.10 (b). Ce mouvement permet de relocaliser une séquence $\{b, \dots, c\}$ de sa position actuelle en une autre position dans la tournée. *Or-opt* est utilisé à la place du *3-opt* pour les instances de plus de 60 sommets. L'exploration de son voisinage est en $o(n^2)$ et il intègre la relaxation de contrainte de la capacité.

- ***Swap*** **intra-tournée** (swap_intra) : c'est un opérateur dont les mouvements consistent à échanger la position de deux sommets dans une tournée. La Figure 3.11 montre l'échange entre les sommets b et e. Cet opérateur est utilisé pour perturber une tournée géante (la procédure de mutation). Dans ce cas, n itérations d'échange dans une tournée sont effectuées et les sommets sont sélectionnés de manière aléatoire. La règle de remplacement ne tient pas compte de la qualité d'un voisin obtenu. L'exploration de son voisinage est en $o(n^2)$ et il intègre la relaxation de contrainte de la capacité.

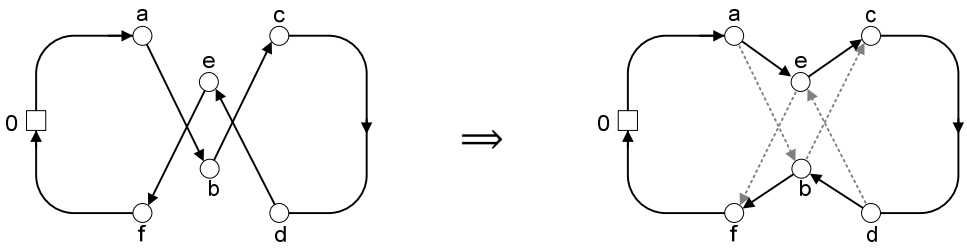


Figure 3.11 : Mouvement de l'opérateur *swap*

- ***2-opt* inter-tournée** (2_opt_inter) : cette version du *2-opt* implique deux tournées. La Figure 3.12 illustre un tel mouvement: l'arc (a, b) de la tournée $(0, .., a, b, .., 0)$ et l'arc (c, d) de la tournée $(0, .., c, d, .., 0)$ sont supprimés. Ils sont remplacés par (a, d) et (c, b) qui donnent deux nouvelles tournées $(0, .., a, d, .., 0)$ et $(0, .., c, b, .., 0)$. Cet opérateur est utilisé pour améliorer une solution du PPRV générée par Split. L'exploration de son voisinage est en $o(n^2)$.

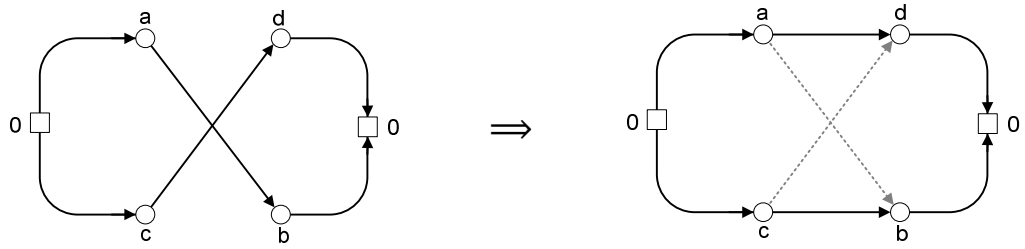


Figure 3.12 : Mouvement de l'opérateur *2-opt* inter-tournée

- ***2.5-opt* inter-tournée** (2.5_opt_inter): il a été introduit initialement par Bentley [Ben92]. Cet opérateur utilise une structure de voisinage dont le mouvement, *relocate*, modifie 3 arcs. Il consiste à déplacer un sommet dans une autre tournée. La Figure 3.13 montre la déplacement du sommet b. D'après [Aar97], le *2.5-opt* offre une amélioration plus de 1% de la qualité avec une augmentation de 50% du temps de calcul par rapport au *2-opt* sur le TSP. Par rapport au *3-opt*, il reste en retrait de 1% sur la qualité mais il est deux fois plus rapide. Dans notre méthode, le *2.5-opt* inter-tournée est utilisé pour l'amélioration d'une solution du PPRV. L'exploration de son voisinage est en $o(n^2)$.

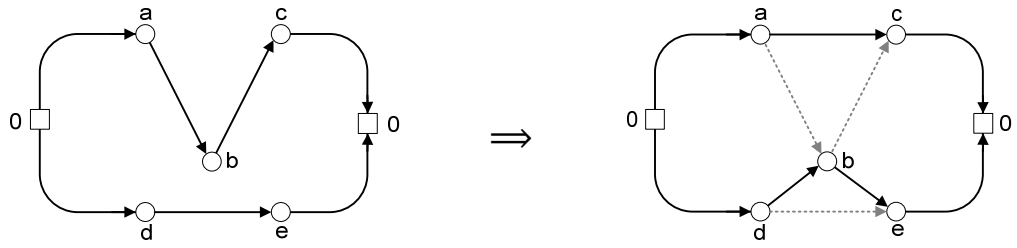


Figure 3.13 : Mouvement de l'opérateur *2.5-opt* inter-tournée

3.2.6.3 Recherche locale pour la tournée géante (VND adaptée)

La procédure **Recherche_locale_1** est utilisée pour l'amélioration d'une tournée géante. C'est une VND avec les 4 opérateurs intra-tournée précédemment décrits. À chaque itération, la tournée géante est d'abord perturbée par le **swap_intra**, puis améliorée par le **2_opt_intra**. Un deuxième opérateur d'amélioration est sélectionné selon la taille du problème : **3_opt_intra** pour les petites instances et **or_opt_intra** pour les grandes.

Le critère d'arrêt dans cette recherche locale tient compte de la réalisabilité de la tournée obtenue. Une tournée non réalisable est autorisée durant la recherche. Le seuil de relaxation tl est initialisé à tol_{init}. Nous adaptons l'équation proposée dans [Her08] pour l'initialisation : $tol_{init} = (3 * \sum_{i \in V : D_i > 0} D_i)/|V|$ *où* V est l'ensemble des sommets et D_i la demande du sommet i. Lorsque la demande des sites est comprise entre -10 et 10, alors la valeur de tl se trouve principalement dans $\{6, 7, 8\}$. Notons qu'une tournée acceptée doit être réalisable. La procédure s'arrête si la tournée courante n'est pas améliorée dans les n_2 dernières itérations ou lorsque la valeur de tol_{init} est nulle. Sinon on diminue tol_{init} afin de réduire l'espace des tournées non-réalisables. Par conséquent, le nombre d'itérations n'est pas fixé dans cette procédure et il peut atteindre $tol_{init} * n_1$. Le paramètre n_1 est fixé à 5 et le paramètre n_2 est fixé aussi à 5. Le schéma de la recherche locale est donné par l'Algorithme 3.10.

Algorithme 3.10 : Schéma de VND adapté (Recherche_locale_1)

Paramètres d'entrée :
G : instance
T_0 : tournée géante
n_1 : nombre d'échanges pour la mutation (par défaut 5)
n_2 : nombre d'itérations (par défaut 5)
Paramètre de sortie :
T : tournée géante améliorée

```
1   Début
2       T ← T0 ;
3       stop ← faux ;
4       it ← 1 ;
5       tol_init ← valeur initiale du seuil de relaxation ;

6       Tant que stop = faux faire
7       |   tl ← tol_init ;
8       |   T' ← swap_intra (T, n1) ;
9       |   Faire
10      |   |   Faire
11      |   |   |   {T',tl} ← 2_opt_intra (G, T', tl) ;
12      |   |   Tant que T' est améliorée
        |   |
13      |   |   Si taille(G) ≤ 60 alors
14      |   |   |   {T', tl} ← 3_opt_intra (G, T', tl) ;
15      |   |   Sinon
16      |   |   |   {T', tl}← or_opt_intra (G, T', tl) ;
17      |   |   Fin Si
18      |   Tant que T' est améliorée
19      |
20      |   Si tol(T') = 0 et T' est meilleure que T alors
21      |   |   T ← T' ;
22      |   Fin Si
        |
23      |   Si it = n2 et (T est améliorée ou tol_init = 0) alors
24      |   |   stop ← vrai ;
25      |   Sinon Si it = n2 alors
26      |   |   tol_init ← tol_init − 1 ;
27      |   |   it ← 1 ;
28      |   Sinon
29      |   |   it ← it + 1 ;
30      |   Fin Si
31      Fin Pour
32  Fin
```

3.2.6.4 Recherche locale pour l'amélioration d'une solution PPRV

La procédure **Recherche_locale_2** est utilisée dans la procédure de transformation des solutions PPRV (voir la section 3.2.5.3). Elle utilise les 5 opérateurs (sauf **swap_intra**). A chaque itération, la solution est améliorée d'abord au niveau inter-tournée puis au niveau intra-tournée. La boucle se termine lorsqu'aucune amélioration n'est trouvée dans l'itération courante. La relaxation de la contrainte de capacité est pas autorisée : le seuil de relaxation dans les opérateurs intra-tournées est nul. L'Algorithme 3.11 montre le schéma algorithmique de cette procédure.

Algorithme 3.11 : Schéma de la recherche locale pour les solutions PPRV (Recherche_locale_2)

```
    Paramètres d'entrée :
        G : instance
        S : solution entrée
    Paramètre de sortie :
        S* : solution améliorée
1   Début
2       S* ← S ;
3       stop ← faux ;
4       Tant que stop = faux faire
5       |   S' ← S* ;
6       |   Faire
7       |   |   S' ← 2_opt_inter(G, S') ;                    // 2-opt externe
8       |   Tant que S' est améliorée
        |
9       |   Faire
10      |   |   S' ← 2.5_opt_inter(G, S') ;                  // 2.5-opt externe
11      |   Tant que S' est améliorée
12      |
13      |   Pour toutes les tournées T dans S' faire
14      |   |   Faire
        |   |   |   T ← 2_opt_intra(G, T, 0) ;               // 2-opt interne
15      |   |   Tant que T est améliorée
16      |   |
17      |   |   Faire
18      |   |   |   Si taille(G) ≤ 60 alors
19      |   |   |   |   T ← 3_opt_intra(G, T, 0) ;           // 3-opt interne
20      |   |   |   Sinon
21      |   |   |   |   T ← or_opt_intra(G, T, 0) ;          // or-opt interne
22      |   |   |   Fin Si
23      |   |   Tant que T est améliorée
24      |   Fin Pour
        |
25      |   Si f(S') < f(S*) alors
26      |   |   S* ← S' ;
27      |   Sinon
28      |   |   stop ← vrai ;
29      |   Fin Si
30      Fin Tant que
31  Fin
```

3.3 Expérimentations

Les algorithmes sont programmés en C++ (avec GNU gcc 4.1.2) et testés sur un PC doté de deux processeurs AMD Opteron 2.3 GHz sous Linux (CentOS 5.4 en 64bits). Nous utilisons Gurobi Optimizer 4.5.1 (64 bits) comme solveur de programmation linéaire, et le nombre de processus / threads pour le branchement est fixé à 8. Nous utilisons le même type d'instances que celles de la section 2.5.1. Les tests sont réalisés sur les instances petites (i.e. nombre de

sommets ≤ 60) et instances (*i.e.* $60 <$ nombre de sommets ≤ 100). Les résultats seront présentés en deux parties : la partie de tournées géantes correspondant au cas mono-agent (sur les petites et les moyennes instances) et la partie de solutions du PPRV à multi-agents (sur les petites instances).

3.3.1 Qualité des tournées géantes

Pour tester la qualité des tournées géantes, nous avons fixé ici le nombre d'agents à 1 ($K = 1$). Nous rappelons que le résultat du PPRV à un agent est cohérent avec celui du 1-PDTSP (voir la définition du PPRV). Nous avons comparé ici aux meilleurs résultats du 1-PDTSP présentés dans la littérature.

Les tests sont réalisés sur les groupes d'instances de {20, 30, 40, 50, 60, 70, 80, 90, 100} sommets avec les petites capacités ($Q \in \{10, 20\}$). Chaque groupe contient 10 instances. Notons que les instances sont généralement plus difficiles à résoudre lorsque la capacité est proche de la valeur absolue de la demande des sommets (qui est comprise dans [-10, 10]). Dans notre expérimentation, on lance 5 fois les calculs en utilisant une graine différente pour le générateur aléatoire à chaque fois. Le nombre d'itérations de ILS est fixé à 300. Nous gardons la meilleure solution trouvée.

Le Tableau 3.1 donne le résultat obtenu sur les petites instances. Les colonnes sont classées en 3 blocs : Instances, Références et Méthode proposée. Elles sont définies comme suit :

- Groupe : information liée au nombre de sommets et à la capacité des agents ;
- Id. : identifiant du graphe dans son groupe ;
- Optimale : valeur de la solution optimale du 1-PDTSP présentée dans [Her08] ;
- Her08 : résultat obtenu par le GRASP/VND dans [Her08] ;
- Moy. : moyenne obtenue sur 5 runs ;
- Best : meilleure solution trouvée sur les 5 runs ;
- %GAP : écart relatif entre la solution optimale et la meilleure solution trouvée ;
- Itr. : numéro d'itération à laquelle on trouve la meilleure solution ;
- Temps[1] : temps de calcul de la procédure de perturbation en secondes (pour 300 itérations);
- Temps[2] : temps de calcul de la procédure d'amélioration locale en secondes.

Le Tableau 3.1 montre le résultat sur les 50 petites instances avec un agent de capacité 10. On peut voir que la méthode proposée est capable de trouver la solution optimale au moins une fois sur les 5 runs pour 49 instances sur 50. En ce qui concerne les heuristiques présentées dans la littérature, les solutions optimales ont été trouvées pour 46 instances dans [Her08] (avec 25 lancements), pour les 50 instances dans [Zha09] (avec 10 lancements) et pour 39 instances dans [Hos10a] (avec 5 lancements). Pour l'instance « N60q10D » qu'on n'a pas réussi à trouver l'optimum, mais l'écart reste très faible (0.02%). Au niveau du temps de calcul, le temps moyen pour la procédure de perturbation (respectivement pour la procédure d'amélioration locale) est de 45.39 secondes (19.23 secondes) pour le groupe de 20 sommets et de 1024.76 secondes (804.08 secondes) pour le groupe de 60 sommets.

Instances		Références		Méthode proposée					
Groupe	Id.	Optimale	Her08	Moy.	Best	%GAP	Itr.	Temps[1]	Temps[2]
N20q10	A	4963	**4963**	**4963.0**	**4963** (5)	0.00	1	29.37	8.39
	B	4976	**4976**	**4976.0**	**4976** (5)	0.00	1	33.45	8.74
	C	6333	**6333**	**6333.0**	**6333** (5)	0.00	1	104.93	50.71
	D	6280	**6280**	**6280.0**	**6280** (5)	0.00	2	77.52	30.99
	E	6415	**6415**	**6415.0**	**6415** (5)	0.00	1	31.87	35.75
	F	4805	**4805**	**4805.0**	**4805** (5)	0.00	1	42.52	11.78
	G	5119	**5119**	**5119.0**	**5119** (5)	0.00	1	26.88	8.03
	H	5594	**5594**	**5594.0**	**5594** (5)	0.00	1	32.73	10.81
	I	5157	**5157**	**5157.0**	**5157** (5)	0.00	11	35.96	15.89
	J	4410	**4410**	**4410.0**	**4410** (5)	0.00	1	38.62	11.25
N30q10	A	6403	**6403**	**6403.0**	**6403** (5)	0.00	3	103.51	100.27
	B	6603	**6603**	**6603.0**	**6603** (5)	0.00	1	98.12	76.25
	C	6486	**6486**	**6486.0**	**6486** (5)	0.00	2	85.04	77.15
	D	6652	**6652**	**6652.0**	**6652** (5)	0.00	7	89.09	137.83
	E	6070	**6070**	**6070.0**	**6070** (5)	0.00	1	87.73	76.76
	F	5737	**5737**	**5737.0**	**5737** (5)	0.00	1	79.87	82.05
	G	9371	**9371**	**9371.0**	**9371** (5)	0.00	1	86.42	144.60
	H	6431	**6431**	**6431.0**	**6431** (5)	0.00	1	80.67	71.64
	I	5821	**5821**	**5821.0**	**5821** (5)	0.00	4	81.01	143.22
	J	6187	6271	**6187.0**	**6187** (5)	0.00	4	94.80	83.74
N40q10	A	7173	**7173**	**7173.0**	**7173** (5)	0.00	1	201.34	221.39
	B	6557	**6557**	6560.0	**6557** (4)	0.00	4	249.09	213.05
	C	7528	**7528**	**7528.0**	**7528** (5)	0.00	14	189.24	223.46
	D	8059	8073	**8059.0**	**8059** (5)	0.00	7	213.88	255.52
	E	6928	6931	**6928.0**	**6928** (5)	0.00	5	204.20	232.12
	F	7506	**7506**	7514.8	**7506** (3)	0.00	17	232.48	255.83
	G	7624	7669	7642.0	**7624** (3)	0.00	35	211.09	390.48
	H	6791	**6791**	6791.0	**6791** (5)	0.00	5	224.20	290.93
	I	7215	**7215**	**7215.0**	**7215** (5)	0.00	2	258.46	327.60
	J	6512	**6512**	**6512.0**	**6512** (5)	0.00	2	205.43	156.94
N50q10	A	6987	**6987**	**6987.0**	**6987** (5)	0.00	4	265.20	232.16
	B	9488	**9488**	**9488.0**	**9488** (5)	0.00	15	677.49	521.52
	C	9110	**9110**	9114.4	**9110** (3)	0.00	8	480.71	542.49
	D	10260	**10260**	10299.5	**10260** (4)	0.00	40	435.20	591.66
	E	9492	**9492**	**9492.0**	**9492** (5)	0.00	17	453.64	837.17
	F	8684	**8684**	**8684.0**	**8684** (5)	0.00	21	486.02	984.81
	G	7126	**7126**	**7126.0**	**7126** (5)	0.00	31	447.54	547.54
	H	8885	**8895**	8912.4	**8885** (3)	0.00	61	423.08	436.45
	I	8329	**8329**	8339.8	**8329** (4)	0.00	31	373.15	786.06
	J	8456	**8456**	**8256.0**	**8456** (5)	0.00	27	639.50	611.44
N60q10	A	8602	**8602**	**8602.0**	**8602** (5)	0.00	35	763.74	697.91
	B	8514	**8514**	**8514.0**	**8514** (5)	0.00	48	1301.75	732.57
	C	9453	**9453**	9476.6	**9453** (1)	0.00	257	1083.57	808.37
	D	11059	11061	11131.8	11061 (1)	0.02	149	1208.36	781.50
	E	9487	9572	9529.0	**9487** (3)	0.00	73	831.26	649.18
	F	9063	**9063**	9105.4	**9063** (2)	0.00	191	1253.59	1232.53
	G	8912	8967	8963.6	**8912** (2)	0.00	104	1246.94	895.69
	H	8424	**8424**	8426.8	**8424** (4)	0.00	33	846.91	694.55
	I	9394	**9394**	9451.6	**9394** (1)	0.00	287	894.97	886.43
	J	8750	**8750**	8765.6	**8750** (4)	0.00	90	816.56	662.09

Tableau 3.1 : Résultat sur les petites instances à un agent ($K = 1$)

Le Tableau 3.2 montre le résultat des instances de taille moyenne (de 70 à 90 sommets) avec toujours un seul agent. Les colonnes ont la même définition que dans le Tableau 3.2, sauf pour les deux colonnes suivantes :

- Her07 : valeur de la solution obtenue par l'heuristique de [Her07] ;
- %GAP : écart relatif par rapport aux résultats de [Her07]. Notons que les solutions optimales ne sont pas connues.

Instances		Références	Méthode proposée					
Groupe	Id.	Her07	Moy.	Best	%GAP	Itr.	Temps[1]	Temps[2]
N70q10	A	**9312**	**9312.0**	**9312** (5)	0.00	25	1295.13	101.15
	B	**10106**	10117.6	**10106** (4)	0.00	74	2798.73	133.13
	C	**9959**	9980.6	**9959** (2)	0.00	107	1324.58	154.55
	D	**10386**	**10386.0**	**10386** (5)	0.00	21	2315.64	176.23
	E	**13055**	13107.2	13069 (1)	0.11	247	2430.43	201.97
	F	**10191**	10224.2	10198 (3)	0.07	64	2571.73	158.81
	G	**9916**	9994.6	**9916** (1)	0.00	240	2106.90	152.64
	H	**8868**	8897.4	**8868** (2)	0.00	245	2127.29	140.63
	I	**10051**	10092.4	**10051** (3)	0.00	138	2820.31	188.81
	J	**10414**	10474.2	**10414** (2)	0.00	16	1437.17	126.47
N80q10	A	11597	11532.8	**11421** (1)	-1.52	298	3042.07	144.23
	B	**12861**	12883.2	**12861** (2)	0.00	203	5264.89	238.28
	C	12471	12414.0	**12358** (2)	-0.91	90	3372.82	273.82
	D	**11050**	11160.0	**11050** (2)	0.00	132	3804.69	201.55
	E	**11185**	11255.4	**11185** (2)	0.00	115	4835.44	332.34
	F	14012	13677.4	**13650** (1)	-2.58	275	3544.13	302.81
	G	11413	11154.0	**11108** (1)	-2.67	280	3292.22	228.64
	H	11307	11112.6	**11075** (1)	-2.05	124	3777.44	235.63
	I	**11063**	11185.0	**11063** (3)	0.00	181	4428.85	283.27
	J	**9263**	9278.2	**9263** (3)	0.00	113	2605.56	144.36
N90q20	A	**8079**	8083.2	**8079** (2)	0.00	24	639.65	73.04
	B	8673	8567.4	**8507** (1)	-1.91	248	954.33	95.10
	C	**8448**	8478.2	8468 (2)	0.24	38	658.10	89.80
	D	**9369**	9504.8	9469 (1)	1.07	272	1045.29	109.86
	E	10072	9749.0	**9674** (1)	-3.95	286	866.55	118.07
	F	10295	10053.4	**9961** (1)	-3.24	250	878.00	119.18
	G	**8339**	8360.6	8355 (1)	0.19	86	580.06	83.27
	H	**9234**	9243.6	**9234** (2)	0.00	165	580.08	116.39
	I	**8601**	8658.6	**8601** (1)	0.00	193	626.25	111.48
	J	**8204**	**8204.0**	**8204** (5)	0.00	8	710.18	84.24

Tableau 3.2 : Résultat sur les instances moyennes à un agent ($K = 1$)

Dans le Tableau 3.2, on constate que la méthode proposée offre une amélioration moyenne de 0.94% en GAP pour les instances à 80 sommets (N80Q10) et de 0.76% pour les instances à 90 sommets (N90Q20) par rapport aux résultats de [Her07]. Le temps moyen pour la procédure de perturbation (pour la procédure de l'amélioration locale) est de 2122.79 secondes (153.44 secondes) pour le groupe de 70 sommets avec la capacité 10 et 753.85 secondes (100.04 secondes) pour le groupe de 90 sommets avec la capacité 20. Il n'y a pas d'instances à 90 sommets avec une capacité de 10.

Le Tableau 3.3 contient le résultat des instances à 100 sommets. Les 6 colonnes suivantes ont une définition différente par rapport au Tableau 3.1 :

- BKR : meilleures solutions connues dans la littérature (*Best Known Result*) ;
- Her08 : résultat présenté dans [Her08] ;
- Zha09 : résultat présenté dans [Zha09] ;
- Hos10 : résultat présenté dans [Hos10] ;
- %GAP[1] : écart relatif entre « BKR » et « Best » ;
- %GAP[2] : écart relatif entre « Her08 » et « Best ».

Instances		Références				Méthode proposée					
Groupe	Id.	BKR	Her08	Zha09	Hos10	Moy.	Best	%GAP[1]	%GAP[2]	Temps[1]	Temps[2]
N100q10	A	11741	11874	11828	11741	11762.2	**11700** (1)	-0.36	-1.47	6619.31	280.75
	B	13066	13288	13114	13066	13128.4	**13003** (1)	-0.25	-1.92	6242.00	383.59
	C	**13893**	14069	13977	**13893**	13975.4	13896 (1)	0.37	-0.89	10274.80	400.09
	D	**14253**	14542	**14253**	14328	14341.6	14257 (1)	0.03	-1.96	11312.30	481.68
	E	**11411**	11650	**11411**	11430	11416.4	**11411** (2)	0.03	-2.03	14003.20	576.32
	F	11644	11734	11644	11813	11689.2	**11635** (1)	-0.08	-0.84	5199.37	340.74
	G	12025	12049	12038	12025	11983.0	**11866** (1)	-0.91	-1.11	8573.25	326.39
	H	12818	12892	12818	12821	12727.2	**12673** (2)	-1.13	-1.70	7449.64	445.39
	I	14025	14048	14032	14025	13929.0	**13834** (1)	-1.01	-1.17	8952.98	479.24
	J	13293	13430	13297	13476	13371.4	**13222** (1)	-0.47	-1.49	5563.60	275.94

Tableau 3.3 : Résultat sur le groupe d'instances N100q10 à un agent ($\boldsymbol{K = 1}$)

Dans le Tableau 3.3, nous avons amélioré le résultat pour 7 instances sur 10 à 100 sommets, avec un GAP d'amélioration moyen de 0.38%. Cela représente une amélioration moyenne de 1.46% par rapport au résultat de [Her08]. Le temps de calcul de la procédure de perturbation (respectivement pour la procédure d'amélioration locale) est en moyenne de 8419.05 secondes (399.01 secondes).

Dans les tableaux précédents, on constate que le temps de la procédure de perturbation est beaucoup plus important que celui de l'amélioration locale, à partir de 70 sommets. Plus précisément, il est en moyenne 2 fois plus élevé pour le groupe à 70 sommets et 5 fois plus élevé pour le groupe à 100 sommets

Dans le Tableau 3.4, nous avons comparé les trois opérateurs d'amélioration de type intra-tournée. Les tests ont été réalisés en appliquant un seul opérateur sur une solution initiale. Les colonnes du Tableau 3.4 sont définies comme suit :

- Groupe : groupe de 10 instances classé par le nombre de sommets et la capacité des agents ;
- %GAP Init. : écart moyen entre les solutions initiales et le meilleur résultat connu pour un groupe d'instances;
- T.A. : moyenne du taux d'amélioration, en pourcentage, entre les solutions initiales et les solutions améliorée par un opérateur ;
- Temps : temps d'exécution moyen d'un opérateur en secondes.

Dans le Tableau 3.4, on constate que le taux d'amélioration du 2-opt inter est meilleur que celui du 3-opt inter[1] à partir du groupe N80q10. Ceci montre que le paramètre Δ utilisé pour la mise à jour du taux de tolérance tl dans l'opérateur 3-opt inter[1] n'est plus adapté lorsque la taille des instances grandit. Par conséquent, la relaxation est rapidement serrée. Les colonnes 3-opt inter[2] montrent le résultat obtenu lorsque la valeur Δ est divisée pour les instances de taille moyenne (*i.e.* nombre de sommets > 60).

		Opérateurs d'amélioration							
		2-opt inter		Or-opt inter		3-opt inter[1]		3-opt inter[2]	
Groupe	%GAP init.	T. A.	Temps	T. A.	Temps	T. A.	Temps	T. A.	Temps
N40q10	193.0%	31.6%	< 0.01	44.8%	< 0.01	48.4%	0.01	48.4%	0.01
N60q10	245.6%	26.2%	< 0.01	28.3%	< 0.01	26.8%	0.05	31.4%	0.07
N80q10	277.6%	28.8%	< 0.01	24.5%	0.01	18.5%	0.12	29.3%	0.15
N100q10	327.2%	18.8%	< 0.01	16.8%	0.03	10.43%	0.28	20.5%	0.32

Tableau 3.4 : Comparaison des opérateurs d'amélioration

3.3.2 Qualité des solutions du PPRV en multi-agents

Dans cette partie, nous présentons le résultat du PPRV en multi-agents obtenu par la méthode proposée. Les tests sont lancés sur les petites instances, avec 2 et 3 agents. Le résultat obtenu est comparé à celui du modèle mathématique présenté en section 2.5.3. Nous rappelons que la fonction objectif du PPRV consiste à minimiser deux critères : la durée du redéploiement et le coût total des parcours. Le deuxième critère est dominé par le premier. Une solution du PPRV en multi-agents possède donc deux valeurs qui correspondent respectivement aux deux critères.

Les Tableau 3.5 et Tableau 3.6 montrent respectivement le résultat des petites instances à 2 agents et à 3 agents. Les colonnes de ces deux tableaux sont définies comme suit :

- {Groupe, Id.} : information correspondant à une instance ;
- {Durée, Coût} : durée du redéploiement et coût total de la solution optimale ;
- {D. Moy., C. Moy.} : durée/coût moyen(ne) des solutions obtenues sur 5 runs ;
- {D. Best, C. Best} : durée/coût de la meilleure solution, avec le nombre de fois que cette solution est trouvée sur 5 runs ;
- %GAP : écart relatif entre « Durée » et « D. Best » ;
- Itr. : numéro d'itération à laquelle on trouve la meilleure solution ;
- {Temps[1], Temps[2]} : temps de calcul en seconde pour la procédure de perturbation et la procédure d'amélioration.

Instances		Sol. Optimale		Méthode proposée							
Groupe	Id.	Durée	Coût	D. Moy.	C. Moy.	D. Best	C. Best	%GAP	Itr.	Temps[1]	Temps[2]
N20q10	A	2628	5245	2682.0	5254.0	**2628** (5)	**5245**	0.00	3	29.05	70.67
	B	2590	4987	2698.0	5259.8	2679 (3)	5223	3.44	39	33.14	59.55
	C	3247	6421	3279.0	6488.6	**3247** (2)	**6421**	0.00	43	104.84	333.17
	D	3162	6201	3162.0	6201.0	**3162** (5)	**6201**	0.00	5	77.30	258.04
	E	3362	6706	3397.4	6725.8	3363 (1)	6575	0.03	36	31.76	219.56
	F	2444	4851	2444.0	4851.0	**2444** (5)	**4851**	0.00	1	42.46	94.92
	G	2833	5602	2834.8	5608.8	**2833** (4)	**5602**	0.00	26	26.78	49.75
	H	2829	5658	2829.0	5658.0	**2829** (5)	**5658**	0.00	1	32.53	99.95
	I	2591	5070	2591.0	5070.0	**2591** (5)	**5070**	0.00	1	35.95	98.46
	J	2398	4672	2398.0	4672.0	**2398** (5)	**4672**	0.00	1	38.28	92.55
N30q10	A	3330	3329	3338.8	6668.8	**3330** (2)	**6659**	0.00	14	94.59	919.31
	B	3598	3593	3604.2	7113.6	**3598** (1)	**7191**	0.00	248	83.95	594.27
	C	3452	3404	3482.6	6892.4	**3452** (1)	6856	0.00	124	80.45	611.81
	D	3565	3561	3600.4	7183.8	**3565** (2)	**7126**	0.00	102	84.79	1076.28
	E	3256	3205	2356.0	6461.0	**3256** (5)	**6461**	0.00	2	85.70	525.46
	F	3053	3032	3053.0	6085.0	**3053** (5)	**6085**	0.00	1	77.03	406.28
	G	4742	4731	4768.0	9469.2	**4742** (1)	9473	0.00	73	79.38	1556.47
	H	3380	3362	3396.8	6762.0	**3380** (2)	**6742**	0.00	105	80.24	716.19
	I	3000	2956	3032.2	5987.0	**3000** (2)	**5956**	0.00	1	79.90	862.52
	J	3224	3112	3224.0	6336.0	**3224** (5)	**6336**	0.00	5	88.37	594.90

Tableau 3.5 : Résultat sur les petites instances avec deux agents ($K = 2$)

Instances		Sol. Optimale		Méthode proposée							
Groupe	Id.	Durée	Coût	D. Moy.	C. Moy.	D. Best	C. Best	%GAP	Itr.	Temps[1]	Temps[2]
N20q10	A	1837	5271	1837.0	5271.0	**1837** (5)	**5271**	0.00	1	29.01	74.60
	B	1990	5817	2036.2	5614.2	**1990** (2)	**5817**	0.00	157	33.12	54.59
	C	2304	6765	2304.0	6791.6	**2304** (3)	**6765**	0.00	50	104.62	330.75
	D	2261	6664	2291.0	6656.4	**2261** (1)	**6664**	0.00	4	77.06	251.27
	E	2331	6825	2331.0	6825.0	**2331** (5)	**6825**	0.00	8	31.72	197.91
	F	1978	5901	1989.0	5468.0	1989 (5)	5468	0.56	9	42.23	85.35
	G	1992	5803	1992.0	5830.0	**1992** (5)	**5803**	0.00	1	26.76	46.48
	H	2182	6159	2229.6	6504.8	2197 (1)	6332	0.69	249	32.54	95.93
	I	1853	5491	1853.6	5483.2	**1853** (4)	**5491**	0.00	2	35.76	93.09
	J	1720	5027	1723.0	5013.8	**1720** (2)	**5027**	0.00	56	38.41	87.63
N30q10	A	2268	2248	2274.4	6749.8	**2268** (4)	**6735**	0.00	20	111.69	834.93
	B	2440	2407	2452.4	7247.6	**2440 (3)**	7162	0.00	4	96.58	576.11
	C	2435	2420	2452.6	7062.2	**2435 (1)**	7048	0.00	258	92.85	550.69
	D	2405	2388	2405.0	7048.0	**2405 (5)**	7048	0.00	2	100.15	932.14
	E	2372	2321	2372.0	6936.0	**2372** (5)	**6936**	0.00	1	103.22	483.84
	F	2214	2078	2218.4	6460.2	**2214** (2)	**6280**	0.00	1	84.74	382.01
	G	3227	3225	3245.4	9695.4	**3227** (1)	9676	0.00	6	91.90	1330.31
	H	2368	2338	2368.0	6971.8	**2368** (4)	**6960**	0.00	5	91.80	624.80
	I	2113	2074	2113.0	6163.0	**2113** (3)	**6163**	0.00	53	90.63	802.73
	J	2286	2267	2289.0	672.8	**2286** (2)	6741	0.00	94	100.37	554.91

Tableau 3.6 : Résultat sur les petites instances avec trois agents ($K = 3$)

Dans le Tableau 3.5, nous avons obtenu 8 solutions optimales (0.17% de gap en moyenne) pour le groupe à 20 sommets et les 10 solutions optimales pour le groupe à 30 sommets. Pour le résultat des petites instances à 3 agents (Tableau 3.6), on a obtenu aussi 8 solutions optimales (0.06% de gap en moyenne) pour le groupe à 20 sommets et 10 pour le groupe à 30 sommets. Au niveau du temps de calcul, la procédure de perturbation ne dépend que de la taille de l'instance. La procédure d'amélioration consiste en deux étapes : la recherche locale de la tournée géante et le découpage par Split. La deuxième étape consomme un temps plus important que la première : le temps moyen de la procédure d'amélioration est de 137.66 secondes (786.35 secondes) pour le groupe à 20 sommets (de 30 sommets) à 2 agents, dont 19.23 secondes (99.35 secondes) pour la première partie et 99.35 secondes (687.01 secondes) pour la deuxième étape.

3.4 Conclusion

Dans ce chapitre, nous avons proposé une approche de type « *route-first, cluster-second* » pour la résolution du PPRV. Le principe de cette approche consiste à trouver d'abord une tournée géante de bonne qualité, puis à la découper en un ensemble de tournées d'agents (*i.e.* une solution du PPRV). Une telle approche utilise donc deux espaces de solutions : l'un pour les tournées géantes et l'autre pour les solutions du PPRV. Elle repose sur un schéma hybride de type ILS/VND dans lequel la VND est utilisée comme recherche locale au sein de l'ILS. Dans un tel schéma, on commence par la construction d'une tournée géante. À chaque itération, un voisin de la tournée géante est sélectionné à l'aide de la procédure de perturbation. Ce voisin est ensuite amélioré par la VND avant d'être découpé en utilisant la procédure Split. Une deuxième recherche locale est ainsi appliquée sur la solution obtenue. Le critère d'arrêt tient compte essentiellement du nombre d'itérations. Par ailleurs, la relaxation de la contrainte de capacité (RCC) est appliquée lors de la recherche de la tournée géante.

La procédure de la construction se compose de 2 heuristiques gloutonnes de type *Nearest Neighbour* randomisée. Elles tiennent compte principalement du critère lié à la qualité de tournées. La réalisabilité d'une tournée géante construite n'est pas assurée par ces deux heuristiques.

La procédure de perturbation utilise la structure « *k-positions* » comme voisinage. Elle se concentre sur la recherche des voisins (tournées gérantes) réalisables sans tenir compte de la qualité associée. Cette procédure repose sur un modèle linéaire, dont la résolution nécessite un solveur de programmation linéaire. Elle permet d'offrir des voisins très diversifiés. Cependant, elle consomme un temps important.

Deux recherches locales sont utilisées dans la méthode proposée : la première pour améliorer la tournée géante (schéma VND) et la deuxième pour l'ensemble de tournées d'agents après le découpage de Split. Dans la première recherche locale, on tient compte de la relaxation de contrainte de la capacité d'agent (RCC) et dans la deuxième on ne travaille que sur les tournées réalisables. Au niveau des opérateurs d'amélioration, nous avons implémenté principalement des opérateurs de type *k-opt* dans les recherches locales.

Dans la partie expérimentation numérique, les tests ont été effectués sur les instances très contraintes (*i.e.* la capacité est proche de la valeur absolue de la demande). Au niveau de la qualité de tournées géantes, notre méthode permet de trouver quasiment toutes les solutions optimales sur les petites instances (nous avons trouvée 59 solutions optimales sur les 60 instances) ; et sur les grandes instances (à 100 sommets), nous avons amélioré 7 des 10 meilleures solutions connues. Au niveau de la qualité des solutions en multi-agents, les résultats obtenus sont comparés avec ceux du chapitre 2. Nous observons un très petit écart sur les instances testées.

Au niveau du temps de calcul, nous avons fait une comparaison simple, en précisant le temps de calcul pour chaque partie par itération. La procédure de perturbation avec le modèle d'affectation et le découpage par Split consomment plus de temps que les autres étapes de la méthode.

Chapitre 4

PPRV avec passage multiple : définition et résolution

Sommaire

4.1 Introduction 115
4.2 Définition du PPRV avec Passage Multiple 117
4.3 Problèmes proches du PPRV-PM 119
4.4 Proposition d'une approche heuristique 121
4.4.1 Création d'une instance auxiliaire 121
4.4.2 Heuristique de résolution 128
4.5 Expérimentations 134
4.5.1 Taille des instances auxiliaires 134
4.5.2 Opérateurs d'amélioration et la recherche locale 135
4.5.3 Résultats avec un agent 136
4.5.4 Résultats avec multi-agents 139
4.6 Conclusion 142

Nous nous concentrons ici sur le Problème de la Planification du Redéploiement de Véhicules partagés avec Passages Multiples (PPRV-PM), dans lequel un ou plusieurs agents peuvent passer plusieurs fois sur un site de service. Le PPRV-PM est motivé par le cas où la capacité des agents est petite ou lorsqu'elle est plus petite que la demande des sites. Ce problème a des similitudes avec les problèmes de type SDVRP (*Split Delivery Vehicle Routing Problem*) et plus précisément le VRP avec la contrainte de division sur collecte et livraison. Dans le PPRV-PM, le passage multiple vient du fait que la demande peut être divisée en plusieurs parties, dont chaque partie est traitée par un agent. En outre, le passage multiple d'un agent sur un même sommet est également autorisé.

Dans ce chapitre, nous commençons par la présentation des Problèmes de Plan de Redéploiement avec la division de la demande avant de donner la définition du PPRV avec Passages Multiples. Nous étudions ensuite dans la section 4.3 les problèmes proches dans la littérature. Nous proposons une approche heuristique en deux phases dans la section 4.4. Les résultats expérimentaux sont présentés dans la section 4.5.

4.1 Introduction

Un Problème de Plan de Redéploiement (PPR) avec la contrainte de division de demande est défini sur un réseau complet $G = (V, E)$ où V est l'ensemble de sommets et E est l'ensemble d'arcs. Pour chaque couple de sommets $(i, j) \in E$, on dispose d'un coût C_{ij} et d'une longueur L_{ij}, qui satisfont tous les deux l'inégalité triangulaire. Un vecteur de demandes $D = \{D_i \in \mathbb{Z} \mid \forall i \in V\}$ est donné. Il peut s'écrire $D = D^+ + D^-$, où D^+ (D^-) est le vecteur positif (négatif) ou nul, de support disjoints, tels que : $|1.D^+| = |1.D^-|$ où 1 est le vecteur

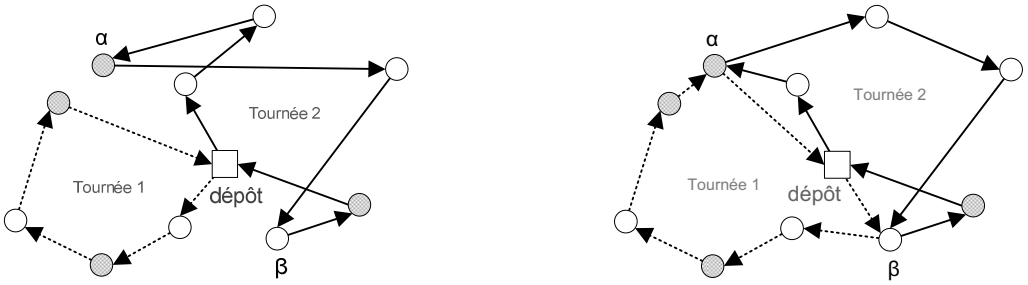
indexé sur V dont tous les éléments sont égaux à 1. On note V_{col} et V_{liv} les supports respectifs de D^+et D^-, *i.e.* $V_{col}(V_{liv})$ est l'ensemble de sommets de collecte (de livraison).

Concrètement, D_i^- (D_i^+) exprime la présence sur le sommet i d'un déficit (excédent) d'un certain type d'objets (noté TO) indifférenciés (par exemple vélos ou véhicules électriques). L'objectif est alors de rétablir l'équilibre, en transportant des objets depuis les sommets de collecte vers les sommets de livraison. On utilise pour ce faire un ensemble de véhicules de transport, supposés identiques de capacité Q. Nous désignons ici un véhicule de transport par un « agent ». Pour simplifier, on suppose que tous ces agents sont initialement basés en un même sommet dépôt et que l'on dispose d'un total de K agents.

Une durée maximale T_{max} est imposée pour le processus de rééquilibrage ou de **redéploiement**. On considère L_{ij}, le temps nécessaire à un agent pour aller de i à j et effectuer une opération de chargement/collecte ou déchargement/livraison en j. On considère aussi C_{ij}, le coût associé à un tel déplacement. On cherche à organiser le processus de rééquilibrage de façon à ce qu'il coûte le moins possible et qu'il respecte les contraintes de durée et de capacité des véhicules, sachant qu'un coût fixe est associé au nombre d'agents utilisés.

Tous les éléments du vecteur D sont divisibles. La division peut s'effectuer jusqu'à l'unité pour les objets TO. Au moment du redéploiement, lorsqu'un agent effectue une opération (de collecte ou livraison) sur un sommet, il est possible que l'agent traite d'abord une partie de la demande associée, puis repasse un peu plus tard pour la partie restante, ou encore laisse les autres agents s'en occuper.

La Figure 4.1 illustre l'influence de la division de demande : on suppose que le sommet α est « difficile » dans le sens où il a une demande importante de livraison. La figure à gauche montre une solution sans division et celle de droite une solution avec division de demande. La division de demande offre une solution plus intéressante dans ce cas.



(a) solution sans division de demande (b) solution avec division de demande

Figure 4.1 : Influence sur la division de demande

Par conséquent, un sommet peut être visité plusieurs fois par le biais de la division de demande. Le passage multiple peut donc être utilisé lorsque la demande du sommet dépasse la capacité des agents ou encore pour réduire le coût de redéploiement en établissant la collaboration entre des agents. Dans ce contexte, la collaboration entre les agents peut prendre plusieurs formes et deux versions du problème peuvent être envisagées :

- **version sans transfert** (PPRV-PM) : un même objet est transporté depuis son origine dans V_{col} jusqu'à sa destination dans V_{liv} par un même agent ; dans ce cas, le coût du

redéploiement intègre le nombre de véhicules et la somme des coûts C_{ij} des arcs (i,j) empruntés par les véhicules;

- **version avec transfert** (PPRV-T : *PPRV avec transfert*) : les objets peuvent être échangés entre les agents lors du passage sur les différents sommets de V. Dans ce deuxième cas, le coût de parcours d'un véhicule doit tenir compte des temps d'attente nécessaire à la synchronisation des échanges : on supposera que le coût lié aux temps d'attente correspond à la somme de ces temps d'attente.

Ces deux versions sont considérées dans un contexte statique (*i.e.* les données liées à la demande sont connues à l'avance). Ces deux problèmes font partie de la deuxième branche de l'ensemble de problèmes du plan de redéploiement avec division de demande (Figure 4.2). Dans ce chapitre, nous nous intéressons exclusivement à la version sans transfert (PPRV-PM).

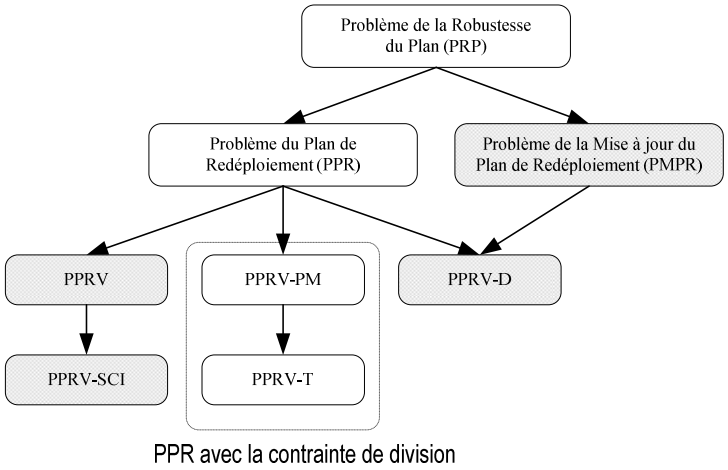


Figure 4.2 : Hiérarchie des problèmes liés au redéploiement de convois

4.2 Définition du PPRV avec Passage Multiple

Le Problème de la Planification du Redéploiement de Véhicules partagés avec Passages Multiples (PPRV-PM) est défini dans le même contexte que le PPRV (voir la section 2.2) : sur un graphe orienté complet $G = (V, E)$ où V est l'ensemble de sommets numérotés de 0 à n et $E = \{(i,j) | \forall i,j \in V \; et \; i \neq j\}$ l'ensemble d'arcs. L'ensemble V est composé du dépôt central (numéroté 0) et des sites de service. Chaque site i est associé à une demande D_i qui exprime la présence en sommet d'un déficit/excédent d'un type d'objets indifférenciés. Un coût de parcours T_{ij} est associé à chaque arc $(i\ ,j)$. C'est le temps nécessaire pour un agent de parcourir le chemin entre deux sommets. Un ensemble de K agents homogènes de capacité Q est disponible pour transporter les objets. Notons que dans le PPRV-PM la capacité Q peut éventuellement être plus petite que certaines demandes. Lorsque la demande $D_i \neq 0$, le passage d'un agent sur le site i doit s'accompagner d'une opération (collecte ou livraison) qui traite une partie de sa demande.

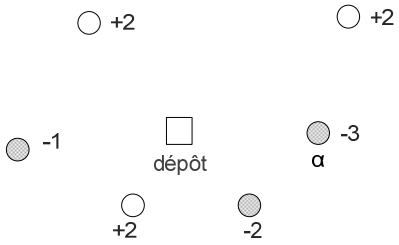
L'objectif du PPRV-PM consiste à planifier les trajectoires de K agents de façon à optimiser les critères liées aux durées de redéploiement en respectant les contraintes suivantes :

(c4.1) la trajectoire d'un agent commence et se termine au dépôt ;
(c4.2) la charge d'un agent ne peut pas dépasser sa capacité ;
(c4.3) toutes les demandes doivent être satisfaites.

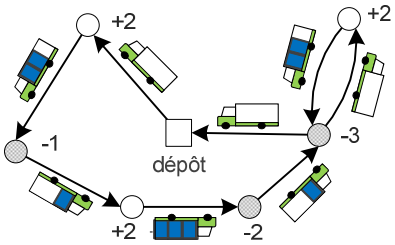
Les critères d'optimisation considérés sont la durée de redéploiement et la durée totale des trajectoires. Nous utilisons ici la même stratégie de domination que celle du PPRV présenté en section 2.2., *i.e.* le premier critère domine le deuxième en utilisant un coefficient α. Par normalisation, on suppose que le coût de transport est égal à la durée de trajectoire pour chaque agent.

Le PPRV-PM ne tient pas compte de la charge initiale. Dans un schéma de redéploiement, un agent part du dépôt avec une charge vide, il effectue une opération (collecte ou livraison) sur chaque sommet de la trajectoire et rentre à la fin au dépôt à vide. L'opération effectuée sur un sommet doit être cohérente avec sa demande, la collecte (livraison) sur un sommet de livraison (collecte) n'est pas autorisée.

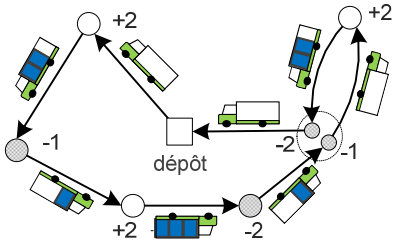
La Figure 4.3 donne un exemple du PPRV-PM : la figure (a) est une instance avec 7 sommets dont les sommets en gris (blanc) représente les sites de livraison (collecte); la figure (b) montre une solution avec un agent de la capacité $Q = 3$; dans cette solution, la demande du sommet α est divisée en 2 parties dont une partie avec 2 objets et l'autre avec 1 objet (figure (c)) ; Notons que toutes les demandes sont divisibles : la figure (d) montre une présentation de cette solution en divisant toutes les demandes en objet unitaire.



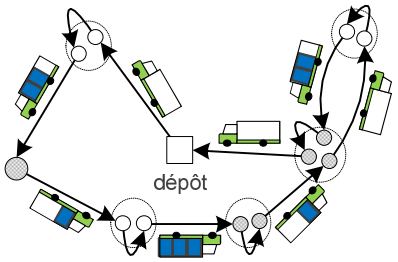
(a) une instance du PPRV-PM



(b) une solution du PPRV-PM



(c) présentation de la solution avec la division de la demande du sommet α



(d) présentation de la solution avec la division unitaire des demandes

Figure 4.3 : Illustration du PPRV-PM

Le PPRV-PM est un problème de collecte et livraison avec une contrainte de division de demande. Il peut se réduire en un problème du type SDVRP (*Split Delivery Vehicle Routing Problem*) avec collecte et livraison. Le PPRV-PM est un problème *NP-difficile*.

Remarque 4.1

Dans l'hypothèse où la demande correspond à une quantité dénombrable d'objets à transporter, alors le PPRV-PM peut être transformé en un problème de collecte et livraison (PDP : *Pickup-and-Delivery Problem*) de type *Many-to-Many* (plus précisément, le 1-PDP : *one-commodity Pickup and Delivery Problem*) en effectuant la division unitaire sur toutes les demandes. Dans une telle transformation, un sommet i avec la demande D_i est divisé en $|D_i|$

copies dont chaque copie a une demande unitaire (Figure 4.4). Notons qu'une demande unitaire est atomique. L'instance obtenue après division unitaire est une instance de PDP. Cela donne la possibilité d'adapter une méthode de résolution du PDP à la résolution du PPRV-PM. Cependant la transformation est pseudo-polynômiale et la taille du problème PDP résultant peut être très importante. De plus, d'un point de vue formel, le PDP obtenu présente de nombreuses symétries (duplication de sites identiques) qui augmentent inutilement la combinatoire et rendent malaisée une résolution exacte.

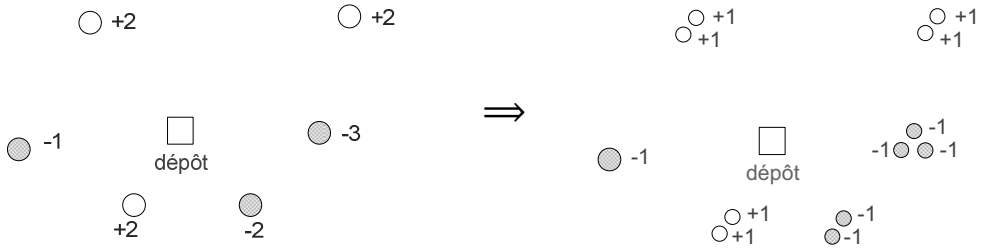


Figure 4.4 : Division unitaire d'une instance PPRV-PM

Remarque 4.2

Avec la présence de la division unitaire de demande, il est possible d'adapter le modèle linéaire du PPRV (présenté dans la section 2.4) au PPRV-PM. On considère que $G' = (V', E')$ est l'instance obtenue en effectuant une division unitaire sur une instance du PPRV-PM. V' contient le dépôt et l'ensemble de sommets (avec la demande unitaire), E' est l'ensemble des arcs. Pour chaque arc $(i, j) \in E'$, le coût de parcours T_{ij} est nul si i et j correspondent à un même sommet initiale (*i.e.* sommet dans G); il correspond au coût de transport pour aller du sommet initial de i au sommet initial de j sinon. L'adaptation du modèle se fait en modifiant deux choses sur le modèle PL 2.1 :

- instance : on utilise ici l'instance auxiliaire G' ;
- charge initiale des agents : les agents partent et rentrent au dépôt avec une charge nulle dans le PPRV-PM, *i.e.* $y_{0+}^k = y_{0-}^k = 0 \quad \forall k = 1, .., K.$

4.3 Problèmes proches du PPRV-PM

Dans la littérature, le PPRV-PM est proche aussi des problèmes de collecte et livraison (PDP) que des problèmes de tournées avec contrainte de division de demande (SDVRP). D'un côté, le PPRV-PM peut se transformer en 1-PDP ; de l'autre côté, il possède une caractéristique forte des problèmes de type SDVRP avec la présence de la contrainte de division de demande. Le Tableau 4.1 regroupe les problèmes qui tiennent compte d'une telle contrainte. Les colonnes de ce tableau sont définies comme suit :

1) *Nom* : abréviation/nom du problème ;
2) *Structure* : structure de requêtes dans un problème [Ber07] ;
3) *# agents* : nombre d'agents disponibles ;
4) *Charge initiale* : charge de l'agent en quittant le dépôt. La valeur 0 signifie sans charge initiale ; 0-Q pour une charge comprise entre 0 et Q ;
5) *Multi-passage par un agent* : possibilité ou non de passage multiple sur un client par un même agent ;
6) *Références* : principaux articles sur le problème.

Nom	Structure	# agents	Charge initiale	Multi-passage par un agent	Références
SDVRP	1-M	K	0-Q	Non	Dror and Trudeau (1989, 1990)
VRPSDP	1-M-1	K	0-Q	Non	Mitra (2005, 2008)
PDPSL	1-1	1	0	Oui	Nowak (2005)
PPRV-PM	M-M	K	0	Oui	

Tableau 4.1 : Problèmes proches du PPRV-PM

Parmi les problèmes présentés dans le Tableau 4.1, le *Split Delivery Vehicle Routing Problem* (SDVRP) est le plus étudié dans la littérature. Le SDVRP est un problème de type *Many-to-One* (M-1), car les objets sont initialement situés sur le dépôt pour être livrés aux clients. Il a été introduit par Dror et Trudeau [Dro89, Dro90]. Par rapport au CVRP, la restriction sur le nombre d'accès à un client (*i.e.* un client est visité une et une seule fois) est retirée dans le SDVRP. La livraison sur un client peut être divisée en plusieurs parties, chaque partie étant traitée par un agent différent. Le passage multiple sur un client par un même agent n'est pas autorisé. Il existe plusieurs variantes du SDVRP en considérant les contraintes comme la fenêtre de temps, la priorité des clients ou encore le type de véhicules. Pour plus de détails sur le SDVPR, on peut se référer aux travaux de [Liu05, Arc08 et Arc11].

Le *Vehicle Routing Problem with Split Deliveries and Pickups* (VRPSPD) [Mit05, Mit08] est considéré comme une variante du SDVRP avec une combinaison des opérations de collecte et de livraison dans chaque tournée. Le VRPSPD est introduit par Mitra [Mit05]. C'est un problème de type *One-to-Many-to-One* (1-M-1) : les objets initialement sur le dépôt doivent être livrés aux clients et ceux qui sont situés initialement sur les clients doivent être collectés au dépôt. C'est une variante du *Vehicle Routing Problem with Backhaul* (VRPB) avec la contrainte de division de demande. L'objectif du VRPSPD est de déterminer le nombre minimum d'agents nécessaire pour satisfaire toutes les demandes de livraison et de collecte et de minimiser le coût total des tournées.

Le *Pickup and Delivery with Split Loads* (PDPSL) est introduit par Nowak [Now05]. C'est un problème de collecte et livraison de type *One-to-One* (1-1) avec la contrainte de division : les demandes de transport concernent des objets à acheminer d'un client (origine) vers un autre client (destination). Les objets liés à une demande peuvent être livrés par des agents différents ou par un même agent avec plusieurs passages, *i.e.* le passage multiple d'un agent sur un client est autorisé (ce qui le différencie des problèmes de type SDVRP). Le dépôt ne participe pas au transport des objets dans ce problème. Nous pouvons se référer les travaux de [Now08, Now09] pour plus d'informations sur le PDPSL.

À notre connaissance, la plupart des études sur les problèmes de tournées de véhicules avec la contrainte de division est focalisée sur le SDVRP et ses variantes. Il existe très peu de travaux sur le PDPSL et les problèmes de collecte et livraison de type Many-to-Many (M-M) comme le PPRV-PM n'ont pas encore été étudiés dans la littérature à ce jour.

4.4 Proposition d'une approche heuristique

Nous proposons ici une approche heuristique en deux phases de type « *divide-first, route-second* » pour le PPRV-PM. L'idée principale de cette méthode consiste à diviser d'abord les demandes (en créant une instance auxiliaire), puis à résoudre par une heuristique. La méthode proposée consiste donc à créer une instance auxiliaire à partir de l'instance initiale du PPRV-PM dans la première phase, puis à résoudre l'instance auxiliaire en appliquant une heuristique dans la seconde phase. La solution obtenue est ensuite projetée sur l'instance initiale. La Figure 4.5 illustre le schéma de la méthode proposée.

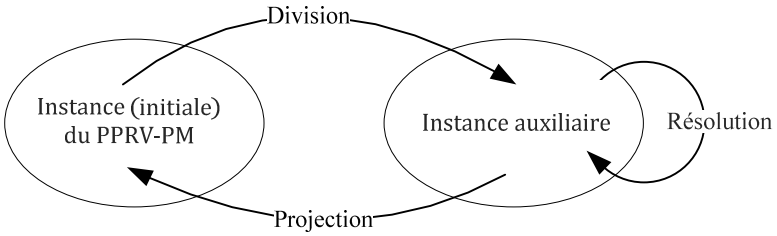


Figure 4.5 : Schéma général de la méthode proposée

Dans la phase de division, nous proposons une stratégie plus sophistiquée que la division unitaire. Elle repose sur un modèle d'affectation pour créer un ensemble de couplages entre les sommets de collecte et les sommets de livraison. La division des demandes est basée sur les couplages obtenus. L'instance auxiliaire hérite des relations de couplage (voir la section 4.4.1). L'heuristique de résolution utilise alors des opérateurs d'amélioration locale de type « *Remove-And-Reinsert* » (RAR) fonctionnant sur les couplages (voir la section 4.4.2). La projection d'une solution de l'instance auxiliaire sur l'instance initiale se réalise en regroupant les sommets divisées (*sommets logiques* : ceux qui se trouvent dans l'instance auxiliaire) en un unique *sommet physique* (*i.e.* le sommet correspondant dans l'instance initiale).

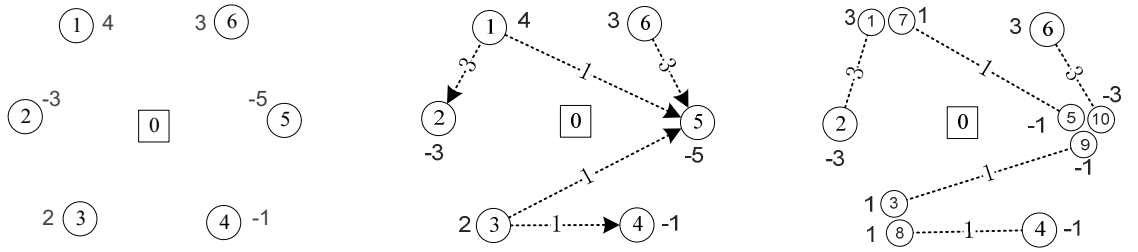
4.4.1 Création d'une instance auxiliaire

Comme mentionné précédemment, l'utilisation d'une instance auxiliaire a pour objectif de diviser d'abord les demandes afin de pouvoir adapter les heuristiques classiques à la résolution du PPRV-PM. L'utilisation d'une instance auxiliaire est une étape incontournable dans notre schéma de résolution. Dans cette création, un processus de division unitaire de demande serait coûteux par la suite, dans la phase de résolution. En effet, la taille de l'instance auxiliaire risque d'être bien supérieure à celle de l'instance initiale. Par conséquent, cela risque d'augmenter fortement le temps de calcul et de produire des solutions plus éloignées de l'optimum. Nous proposons une stratégie de division plus sophistiquée qui nous permet de réduire la taille de l'instance auxiliaire en utilisant un processus de couplage.

Le principe du couplage consiste à affecter des objets entre les sommets de collecte et les sommets de livraison afin de créer un ensemble de relations « offre-demande ». Ces relations doivent être vues comme des contributions équilibrées (avec la somme nulle) dans le système global. Le résultat du couplage est utilisé ensuite pour la division des demandes. On peut alors avoir plusieurs stratégies de division de demande à partir d'un résultat de couplage. Par exemple, on peut fixer un seuil pour limiter la quantité d'objets affectée dans les couples, etc. La division donne un ensemble de couples offre-demande spécifiques sans connexion entre eux. Un couple offre-demande est représenté sous la forme d'un triplet qui contient un sommet logique « fournisseur » (avec la demande positive), un sommet logique « consommateur » (avec la demande négative) et une quantité impliquée par chacun. Les

sommets logiques sont donc liés deux à deux par les couples dans une instance auxiliaire. Notons que l'heuristique proposée tient compte des couples offre-demande dans la phase de résolution.

La Figure 4.6 donne un exemple de la création d'une instance auxiliaire. La Figure 4.6 (a) est une instance du PPRV-PM avec 7 sommets (0 est le dépôt, les sites de service sont numérotés de 1 à 6). La figure (b) donne le résultat de couplage avec 5 relations offre-demande, dans laquelle les sommets (physiques) 1, 3 et 5 font partie de plusieurs couples. La Figure 4.6 (c) illustre une instance auxiliaire possible en divisant ces 3 sommets physiques. L'instance auxiliaire obtenue contient 11 sommets (dont le dépôt 0 et les 9 sommets logiques) avec 5 couples offre-demande. L'algorithme de couplage sera décrit dans la section 4.4.1.1 et le détail lié aux stratégies de la division sera discuté dans la section 4.4.1.2.



(a) instance initiale (b) un résultat de couplage (c) une instance auxiliaire

Figure 4.6 : Création d'une instance auxiliaire

4.4.1.1 Algorithme de création des couples « offre-demande »

Le processus de couplage repose sur un problème d'affectation, qui est défini sur un graphe biparti $G = (V, E)$ tel que $V = V^+ \cup V^-$et $E = \{(i,j) | \forall i \in V^+, \forall j \in V^-\}$. Chaque sommet i est associé à une demande D_i, où $D_i > 0$ si $i \in V^+$et $D_i < 0$ sinon. On considère que la somme des demandes est nulle. Chaque arc (i,j) est associé à un coût d'utilisation T_{ij} par unité d'objet. Le coût T_{ij} correspond ici au temps de parcours entre deux sommets. L'objectif est de trouver une affectation des objets des sommets de V^+ (sommets source) aux sommets de V^-(sommets puits), tout en minimisant le coût total d'affectation. Le programme linéaire (PL 4.1) donne une formulation linéaire pour modéliser un tel problème d'affectation. Il utilise les variables à deux indices f_{ij} pour la quantité d'objets affectés du sommet i au sommet j où $i \in V^+$ et $j \in V^-$. Les contraintes (4.1.1) assurent que le nombre d'objets affectés d'un sommet source vers les sommets puits est égal à la demande. Les contraintes (4.1.2) sont la version des contraintes (4.1.1) correspondant à chaque sommet puits.

$$\text{(PL 4.1)} \begin{cases} \text{Minimiser} \sum_{(i,j)\in E} T_{ij} f_{ij} \\ \text{s. c.} \\ \sum_{j\in V^-} f_{ij} = D_i \qquad \forall i \in V^+ \quad (4.1.1) \\ \sum_{j\in V^+} f_{ji} = |D_i| \qquad \forall i \in V^- \quad (4.1.2) \\ f_{ij} \in \mathbb{N} \end{cases}$$

Il existe plusieurs méthodes exactes pour résoudre un tel problème d'affectation en temps polynômial. Nous avons utilisé la méthode décrite dans Hillier et Liberman [Hil01]. On applique d'abord l'algorithme du coin supérieur gauche pour construire une affectation initiale. Elle est ensuite améliorée par l'algorithme du simplexe de transport (*Transportation*

simplex method). Les deux algorithmes sont décrits respectivement par l'Algorithme 4.1 et l'Algorithme 4.2.

Dans l'Algorithme 4.1, on utilise une matrice de $|V^+|$ lignes et $|V^-|$ colonnes pour stocker le résultat des affectations. Dans cette matrice, les sommets sources sont donc indexés sur les lignes et les sommets puits sont sur les colonnes. Une case (i, j) dans la matrice représente une variable f_{ij}, qui donne la quantité d'objets affectés de i à j. Le nom de cet algorithme vient du fait que l'affectation commence par le coin supérieur gauche de la matrice (*i.e.* $= j = 0$) et se termine en bas à droite, *i.e.* la case $(|V^+| - 1, |V^-| - 1)$. L'algorithme donne une trajectoire d'affectation de la case (0,0) vers la case $(|V^+| - 1, |V^-| - 1)$ dans la matrice. Les variables dans cette trajectoire sont dites *variables de base*. Le nombre de variables de base est égal à $(|V^+| + |V^-| - 1)$. L'affectation construite ici est dite solution *BFS* (*Basic Feasible Solution*) au problème d'affectation. Une *BFS* est donc une matrice d'affectation et un ensemble de variables de base. La solution obtenue sert ensuite dans l'algorithme de simplexe de transport (Algorithme 4.2) comme paramètre d'entrée.

Algorithme 4.1 : Algorithme du coin supérieur gauche

```
    Paramètres d'entrée :
        V = V⁺ ∪ V⁻ : ensemble des sommets
        D : vecteur de la demande
    Paramètre de sortie :
        F : matrice d'affectation
        B : matrice pour indiquer les variables de base
1   Début
2       i, j ¬ 0 ;
3       B, F ¬ 0 ;                          // matrices avec |V⁻| colonnes et |V⁺| lignes
4       Tant que i < |V⁺ | et j < |V⁻ | faire
5       |   F_ij ¬ min{D_i , - D_j};
6       |   B_ij ¬ 1 ;
7       |   D_i ¬ D_i − F_ij ;
8       |   D_j ¬ D_j − F_ij ;
9       |   Si D_i est nul alors
10      |   |   i++ ;
11      |   Sinon Si D_j est nul alors
12      |   |   j++ ;
13      |   Fin Si
14      Fin Tant que

15      Tant que i < |V⁺ | faire
16      |   B_i(|V⁻|−1) ¬ 1 ;
17      |   i++ ;
18      Fin Tant que
19  Fin
```

L'algorithme du simplexe de transport (l'Algorithme 4.2) est une version simplifiée du simplexe, adaptée aux problèmes de transport [Hil01]. Il se repose sur un schéma d'exécution itérative : étant donné une solution *BFS*, l'algorithme du simplexe de transport cherche à remplacer à chaque itération une variable de base par une *variable hors base* lorsque celle-ci apporte la réduction maximale du coût total d'affectation. L'algorithme se termine quand il n'existe plus de réduction possible en parcourant tous les variables hors base.

Lors de l'évaluation d'une variable hors base dans une itération, la réduction liée à l'inclusion (en base) d'une *variable hors base* f_{ij} est calculé par l'équation suivante : $T_{ij} - U_i - X_j$ où U_i correspond au coût (relatif) d'envoi d'un objet du sommet source i et X_j représente le coût (relatif) de réception d'un objet sur le sommet puits j dans la solution courante. Cette équation permet de mesurer la réduction du coût en supposant d'établir une affectation de i à j.

Si $T_{ij} - U_i - X_j < 0$, alors l'affectation d'objets de i vers j est plus intéressante que celle d'actuelle liée à ces deux sommets. Les deux vecteurs U et X sont calculés en parcourant toutes les variables de base f_{ij} avec les équations $T_{ij} - U_i - X_j = 0$ et $U_0 = 0$. Notons que les variables U_i sont équivalents aux « variables dual » dans un algorithme simplexe.

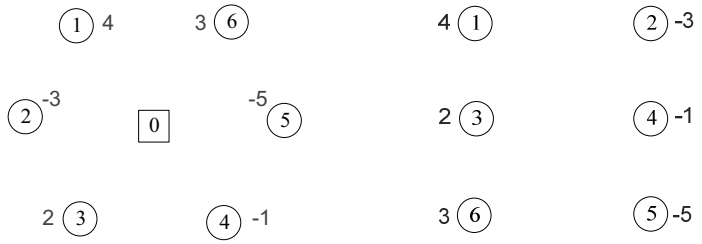
Une fois la plus petite valeur de réduction (c^*) est identifiée en parcourant toutes les variables hors base, le changement de base aura lieu quand il est profitable, *i.e.* $c^* < 0$. Le changement de base consiste à remplacer une variable de base par la variable hors base correspondant à c^*. Pour déterminer la variable sortant de la base, on construit dans un premier temps un arbre en présentant les arcs liés aux variables de base sur le graphe biparti G. En supposant que $f_{i^*j^*}$ est la variable hors base liée à c^*, l'introduction de $f_{i^*j^*}$ va donc former un circuit sur l'arbre. Nous nous intéressons à la recherche de l'arc (i', j') dans le sens contraire de (i^*, j^*) sur le circuit, avec la plus petite quantité affectée (noté f'). Après avoir trouvé l'arc (i', j'), on effectue la mise à jour des variables d'affectation dans le circuit en diminuant la quantité f' pour celles qui sont dans le sens contraire (de l'arc (i^*, j^*)) et en augmentant pour celles qui ont le même sens. La variable $f_{i'j'}$ est la variable sortant de la base. Le détail d'un tel changement sera décrit dans l'exemple suivant (avec Figure 4.7 et Figure 4.8). L'algorithme s'arrête lorsque $c^* \geq 0$.

Algorithme 4.2 : Algorithme du simplexe de transport

Paramètres d'entrée :
$V = V^+ \cup V^-$: ensemble des sommets
C : matrice de coût
F : matrice d'affectation
B : matrice pour indiquer les variables de base
Paramètre de sortie :
F : matrice d'affectation

```
1  Début
2    stop ¬ faux ;
3    Tant que non stop faire
4      | Calculer des deux vecteurs U et X ;
5      | c* ¬ 0 ;                                   // coût d'augmentation
6      | Pour chaque couple (i, j) où B_ij est nul faire
7      |   | Si T_ij – U_i – X_j < c* alors
8      |   |   | c* ¬ T_ij – U_i – X_j;
9      |   |   | i* ¬ i ;
10     |   |   | j* ¬ j ;
11     |   | Fin Si
12     | Fin Pour
       |
13     | Si c* < 0 alors
14     |   | Identifier le chemin de j* à i* ;      // chemin critique : cycle en ajoutant l'arc (i*, j*)
15     |   | (i', j') ¬ avec F_i'j' la plus grande réduction pour tous les arcs (i, j) sur le chemin critique;
16     |   | B_i'j' ¬ 0;                            // variable sort de la base
17     |   | Pour chaque arc (i, j) dans le chemin critique faire
18     |   |   | Si (i, j) est la même sens que (i*, j*) alors
19     |   |   |   | F_ij ¬ F_ij + F_i'j' ;
20     |   |   | Sinon
21     |   |   |   | F_ij ¬ F_ij – F_i'j' ;
22     |   |   | Fin Si
23     |   | Fin Pour
24     |   | F_i*j* ¬ F_i'j' ;
25     |   | B_i*j* ¬ 1 ;                           // variable entre dans la base
26     | Sinon
27     |   | stop ¬ vrai ;
28     | Fin Si
29   Fin Tant que
30 Fin
```

Voici un exemple de création de couplage « offre-demande » en appliquant l'algorithme proposé : en considérant que la Figure 4.7 (a) est une instance initiale du PPRV-PM, la figure (b) est le graphe biparti correspondant à l'instance initiale avec $V^+ = \{1, 3, 6\}$ et $V^- = \{2, 4, 5\}$. La figure (c) est le tableau des distances T associé au graphe biparti.



(a) instance initiale (b) graphe biparti

	②	④	⑤
①	1	2	1.7
③	1	1	1.7
⑥	1.7	1.7	1

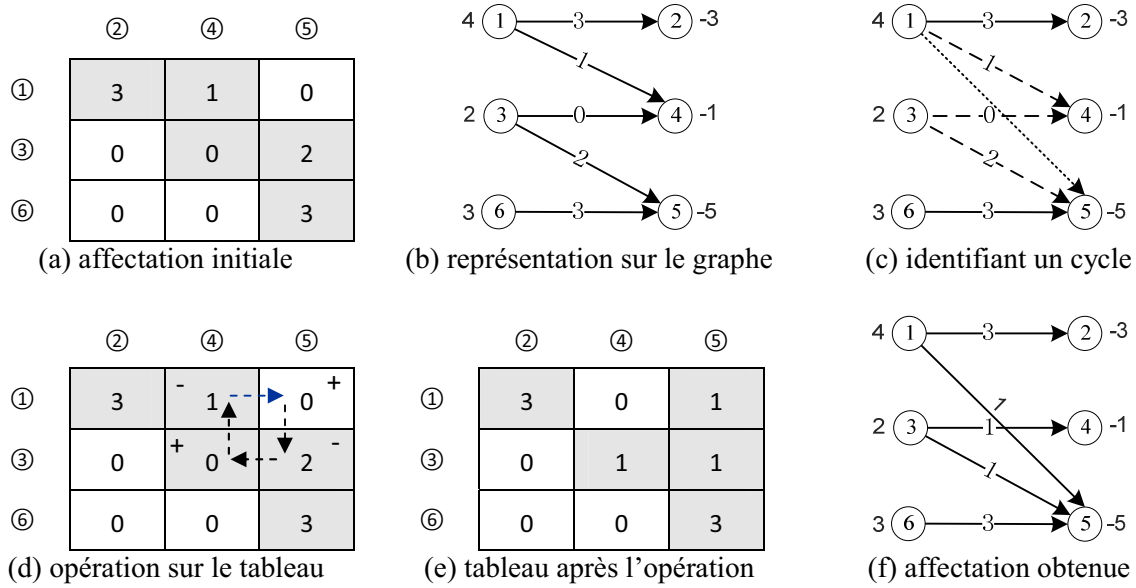
(c) tableau des distances T

Figure 4.7 : Données pour l'affectation

La Figure 4.8 montre l'application de l'algorithme du simplexe de transport : les deux premières figures décrivent l'affectation initiale obtenue en appliquant l'algorithme du coin supérieur gauche sur le graphe biparti. La Figure 4.8 (a) est la matrice d'affectation et les variables de base (cases en gris) et la Figure 4.8 (b) donne le graphe correspondant à l'affectation initiale. Les figures (c), (d), (e) et (f) de la Figure 4.8 illustrent les itérations en appliquant l'algorithme du simplexe de transport à partir de l'affectation initiale. Chaque itération commence par calculer les deux vecteurs U et X en parcourant toutes les variables de base f_{ij} avec $T_{ij} - U_i - X_j = 0$ et $U_0 = 0$. Comme par exemple, à la première itération, nous avons les équations suivantes :

$$1 - 0 - X_0 = 0 \qquad 2 - 0 - X_1 = 0$$
$$1 - U_1 - X_1 = 0 \qquad 1.7 - U_1 - X_2 = 0$$
$$1 - U_2 - X_2 = 0$$

La résolution de ces équations est triviale, elle donne $U = \{0, -1, -1.7\}$ et $X = \{1, 2, 2.7\}$. Puis, on parcoure toutes les variables hors base afin d'identifier celle qui a la plus petite valeur de réduction : $c^* = \min\{c_{02}, c_{10}, c_{20}, c_{21}\} = \min\{-1, 1, 2.4, 1.4\} = -1$, *i.e.* $(i^*, j^*) = (0, 2) = (1,5)$. La Figure 4.8 (c) montre le circuit obtenu en ajoutant l'arc (i^*, j^*) sur l'arbre correspondant aux variables de base, dans lequel on cherche le chemin (sur les arcs d'arbre) pour remonter de j^* à i^* , *i.e. chemin critique*. Le chemin critique illustré sur la Figure 4.8 (c) est (5,3,4, 1). Pour déterminer la variable sortant de la base, on cherche la plus grande réduction sur le chemin critique. Elle est égale à la plus grande charge des arcs dans le sens opposé au chemin : $F_{i'j'} = \min\{F_{12}, F_{01}\} = \min\{F_{35}, F_{14}\} = F_{14} = 1$. La variable f_{14} est la variable sortant de la base, et f_{15} est la variable entrante. Une fois ces deux variables déterminées, on fait la mise à jour de la charge du chemin critique. La Figure 4.8 (d) montre une représentation de la mise à jour du chemin critique sur le tableau de simplexe : la réduction de la quantité $F_{i'j'}$ pour les arcs de sens opposé au chemin et l'addition sinon. La Figure 4.8 (e) donne le tableau à la fin de cette itération et la Figure 4.8 (f) illustre le graphe correspondant. Pour cet exemple, on ne trouve plus de c^* négatif à l'itération suivante. L'affectation sur la Figure 4.8 (f) est l'affectation optimale.



(a) affectation initiale (b) représentation sur le graphe (c) identifiant un cycle

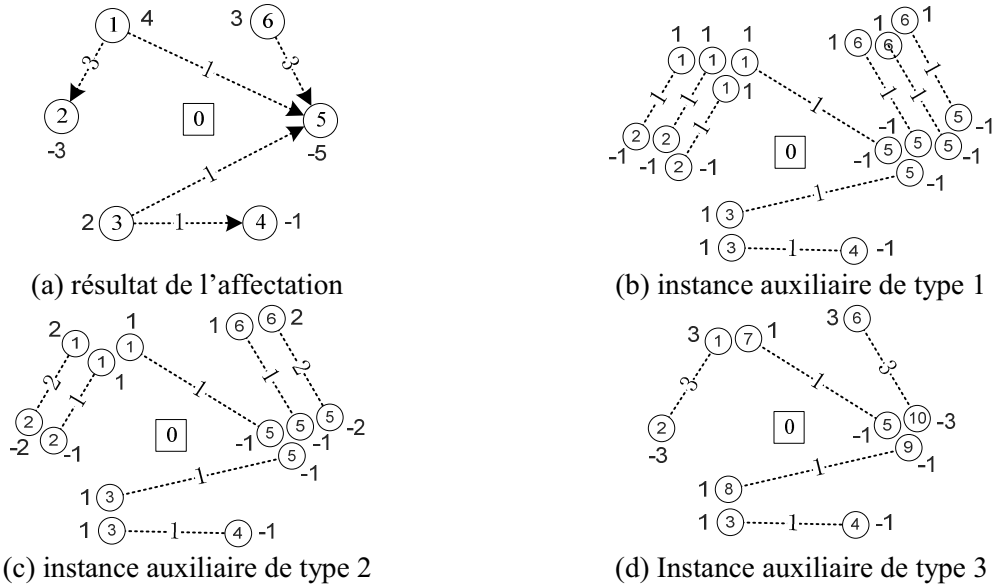
(d) opération sur le tableau (e) tableau après l'opération (f) affectation obtenue

Figure 4.8 : Application de l'algorithme du simplexe de transport

4.4.1.2 Stratégies de la division de demande

Le résultat d'affectation contient un ensemble de relations qui sert ensuite à la création des couples offre-demande. Comme mentionné précédemment, un couple offre-demande est représenté sous la forme d'un triplet qui contient un sommet de collecte, un sommet de livraison et une quantité d'objets couplés. Tous les couples doivent être indépendants les uns des autres. La création des couples accompagne la division des *sommets physiques* en *sommets logiques*. Nous rappelons que les sommets dans l'instance initiale (respectivement l'instance auxiliaire) sont désignés par *sommets physiques* (*sommets logiques*). Notons que dans une instance auxiliaire, la distance entre deux *sommets logiques* est égale à 0 s'ils correspondent à un même *sommet physique* et à la distance entre les deux *sommets physiques* sinon.

Il existe plusieurs stratégies pour créer des couples offre-demande à partir du résultat d'affectation, en utilisant les différentes valeurs du ***seuil d'affectation*** qui limite le nombre d'objets affectés dans un couple. Les 3 cas suivants nous semblent les plus significatifs : (1) unitaire ; (2) une valeur prédéfinie ; (3) la capacité de l'agent. La Figure 4.9 illustre la différence entre ces 3 cas. On suppose que la Figure 4.9 (a) est le résultat de l'affectation sur l'instance initiale. La Figure 4.9 (b) montre l'instance auxiliaire obtenue en appliquant la division unitaire. Elle contient ici $\sum_{i\in V^+} D_i$ (ou $\sum_{i\in V^-} D_i$) couples où V^+ (V^-) est l'ensemble des sommets physiques avec la demande positive (négative). Il peut y avoir une grande quantité de couples (on en a 9 sur cet exemple). La Figure (c) donne l'instance auxiliaire avec la division par la valeur du seuil fixée à 2. Une telle division permet d'offrir un équilibre entre le nombre de couples et la perte de qualité liée à l'agrégation lors de la résolution. Rappelons qu'une fois la division de demande effectuée, les sommets logiques ne sont plus divisibles dans la phase de résolution. Une telle stratégie permet d'accélérer la résolution en autorisant une partie de perte au niveau de solution. La Figure 4.9 (d) montre le cas où la capacité de l'agent sert comme seuil de division. Dans ce cas, on minimise le nombre de couples (on en a 5 sur cet exemple). Cependant la perte de qualité liée à l'agrégation risque d'être sensible.



(a) résultat de l'affectation

(b) instance auxiliaire de type 1

(c) instance auxiliaire de type 2

(d) Instance auxiliaire de type 3

Figure 4.9 : Création des instances auxiliaires

L'utilisation d'un seuil d'affectation offre une possibilité de « contrôle » du degré d'agrégation et donc de la complexité de résolution de l'instance auxiliaire. Outre la division avec le *seuil d'affectation*, on peut éventuellement aussi avoir une division basée sur le nombre de couples souhaités. Dans ce cas, la division de sommet se fait itérativement et l'on s'arrête lorsque le nombre souhaité est atteint. Nous utilisons la division avec la valeur de seuil prédéfinie.

Remarque 4.3

Une instance auxiliaire ressemble à une instance du PPRV (avec une structure de type M-M, voir le chapitre 3). On fait l'hypothèse que la demande d'un *sommet logique* ne peut plus être divisée. L'accès sur un *sommet logique* respecte la contrainte mono-accès (ce n'est pas le cas pour une instance originale du PPRV-PM). Chaque sommet logique est visité une et une seule fois. De plus, la présence des couples offre-demande dans l'instance auxiliaire induisent une différence par rapport à l'instance du PPRV. Un tel schéma de résolution réduit l'espace des solutions du PPRV-PM. Notons que l'utilisation d'un faible seuil d'affectation permet de réduire l'effet de réduction au niveau de l'espace des solutions.

Remarque 4.4

Un couple offre-demande possède la même structure qu'une requête de type 1-1 (voir la section 1.1.3.1). Ils sont représentés tous les deux sous la forme d'un triplet qui contient un sommet origine, un sommet destination et une quantité d'objets. Cependant, la contrainte de précédence imposée sur une requête ne s'applique pas à un couple offre-demande. Plus précisément, on n'impose pas de relation de précédence pour les deux sommets d'un couple. Le sommet destination d'une couple peut être visité avant ou après son sommet origine selon la charge de l'agent. Le couple sert donc à indiquer que la contribution globale des sommets logiques est nulle. Autrement dit, supprimer le couple n'affecte pas la balance générale des demandes sur le problème.

Remarque 4.5

L'utilisation des couples offre-demande a pour objectif de réduire la taille des instances auxiliaires. Cette taille se réfère à la complexité de résolution, qui correspond en général au nombre d'itérations/mouvements. Par exemple, pour un opérateur de recherche locale avec des mouvements sur sommets/arcs, la complexité est liée au nombre de sommets/arcs. Dans l'heuristique proposée, l'utilisation des opérateurs avec des mouvements sur sommets/arcs risquerait d'augmenter fortement les temps de calcul. Nous proposons ici des opérateurs fonctionnant sur les couples offre-demande : dans la partie de construction d'une solution, on insère un couple (deux sommets logiques) à chaque itération ; dans la partie d'amélioration locale, l'exploration des solutions voisines repose sur la suppression puis la réinsertion d'un certain nombre de couples dans la solution courante. De plus, les couples permettent aussi d'équilibrer la quantité d'objets dans chaque tournée d'agent.

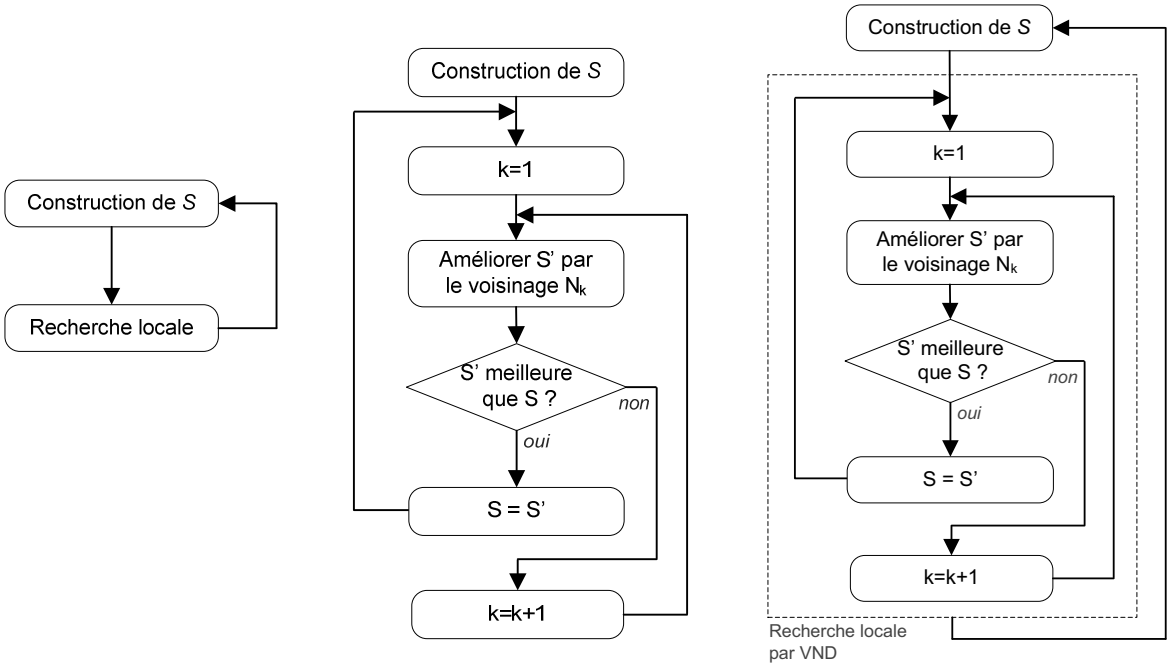
4.4.2 Heuristique de résolution

4.4.2.1 Schéma de l'heuristique proposée

L'heuristique de résolution repose sur le schéma hybride GRASP/VND, qui est une hybridation entre GRASP et VND, dont on rappelle brièvement le principe :

- GRASP (*Greedy Randomized Adaptive Search Procedure*) a été introduit par Feo et Resende [Feo89]. C'est une méta-heuristique à démarrages multiples, dont chaque itération contient deux phases : une phase de construction et une phase de recherche locale. La phase de construction génère une solution initiale de manière randomisée. La solution générée est ensuite améliorée par une recherche locale dans la seconde phase. Ces deux phases induisent une stratégie de type diversification/intensification pour la recherche d'optimum local. La phase de construction correspond à la diversification et la phase de recherche locale à l'intensification ;
- VND (*Variable Neighborhood Descent*) est considérée comme la recherche locale de VNS. Elle utilise un ensemble de structures de voisinage prédéfinies (noté N) et souvent triées selon leur complexité. Le principe de VND consiste à utiliser une structure plus complexe lorsque l'optimum local de la structure actuelle est atteint. On peut se référer à la section 3.2.1 pour davantage de détails sur la VND.

Le principe d'une telle hybridation consiste à effectuer la recherche locale de GRASP par la VND. On garde la bonne diversification offerte par GRASP et on bénéficie de l'excellent comportement de la VND en tant que recherche locale. Le principe du schéma GRASP/VND est montré dans la Figure 4.10.



(a) : Schéma de GRASP (b) : Schéma de VND (c) : Schéma de GRASP/VND

Figure 4.10 : Schéma hybridation entre GRASP et VNS

L'Algorithme 4.3 donne le schéma GRASP/VND. À chaque itération, on commence par la construction d'une solution initiale S en utilisant la procédure **Construire_solution** (voir la section 4.4.2.2). Chaque itération utilise une valeur de grain différente. La solution construite est ensuite améliorée par **Recherche_locale** (voir la section 4.4.2.3). On applique une procédure de post-amélioration (qui utilise le voisinage le plus complexe) sur la meilleure solution trouvée (au bout de $It_{\max}$ itérations), avant de la projeter sur l'instance initiale.

Algorithme 4.3 : Schéma de GRASP/VND

Paramètres d'entrée :
G : l'instance auxiliaire
$Grain$: graine du générateur aléatoire
$It_{\max}$: nombre d'itérations
Paramètre de sortie :
S_p : solution du PPRV-PM

Début
  Mettre $Grain$ comme valeur de grain aléatoire;
  $S^* \leftarrow \emptyset$;
  $Liste\_grains \leftarrow \emptyset$;
  Pour $i \leftarrow 1$ à $It_{\max}$ **faire**
    Faire
      $gr \leftarrow$ générer aléatoirement un chiffre ;
    Tant que $gr \notin Liste\_grains$;
    $Liste\_grains \leftarrow Liste_{grains} \cup \{ gr \}$;
    $S \leftarrow$ Construire_solution (G, gr) ; *// voir l'Algorithme 4.4*
    $S \leftarrow$ Recherche_locale (G, S) ; *// voir l'Algorithme 4.6*
    Si S est meilleure que S^* **alors**
      $S^* \leftarrow S$;
    Fin Si
  Fin Pour

  $S^* \leftarrow$ Post_amélioration(G, S^*) ;
  $S_p \leftarrow$ Projection_solution(G, S^*) ; *// voir l'Algorithme 4.7*
Fin

4.4.2.2 Procédure de construction d'une solution initiale

La procédure de construction utilisée est basée sur une heuristique gloutonne randomisée. L'Algorithme 4.4 donne le schéma d'une telle heuristique : on commence par la distribution des couples aux tournées. Les couples distribués à la tournée numérotée k sont stockés dans le vecteur C_k. On passe ensuite dans la phase construction : une solution construite est composée d'un ensemble de K tournées, $i.e.\ S = \{S_1, S_2, \dots, S_K\}$. Les tournées sont construites les unes après les autres (s'il en existe plusieurs). Lors de la construction de la tournée S_k, on insère d'abord la moitié des couples de C_k de manière gloutonne. La seconde partie de C_k est ensuite insérée de manière aléatoire. L'insertion gloutonne fait appel à la procédure **Evaluer_cout_insertion**, pour évaluer l'augmentation induite par l'insertion du couple dans la tournée actuelle. La procédure se termine lorsque tous les couples sont insérés.

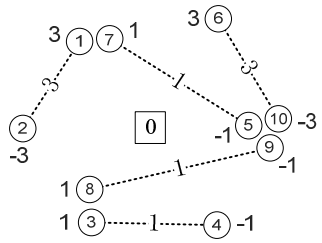
Algorithme 4.4 : Procédure de construction (Construire_solution)

```
    Paramètres d'entrée :
        G : instance auxiliaire
        Grain : graine du générateur aléatoire
    Paramètre de sortie :
        S : solution auxiliaire
1   Début
2     Mettre Grain comme valeur de grain aléatoire;
3     K ¬ Nb_agents(G) ;
4     R ¬ Couples(G) ;
5     C ¬ ∅ ;
6     S ¬ ∅ ;
7     k ¬ 1 ;
8     Pour i = 1 à |R| faire
9     |     C_k ¬ C_k ∪ {R_i} ;
10    |     k ¬ k + 1 ;
11    |     Si k > K alors
12    |     |     k ¬ 1 ;
13    |     Fin Si
14    Fin Pour

15    Pour k ¬ 1 à K faire
16    |     S_k ¬ nouvelle_tournée() ;
17    |     Pour i=1 à ⌊|R|/K⌋ /2 faire
18    |     |     c* ¬ +∞ ;
19    |     |     Pour i=1 à |C_k| faire
20    |     |     |     c ¬ Evaluer_cout_insertion(S_k, C_k[i]) ;
21    |     |     |     Si c < c* alors
22    |     |     |     |     r* ¬ C_k[i] ;
23    |     |     |     |     c* ¬ c ;
24    |     |     |     Fin Si
25    |     |     Fin Pour
26    |     |     Insérer_couple (S_k, r*) ;
27    |     |     C_k ¬ C_k \ {r*} ;
28    |     Fin Pour
29    |
30    |     Tant que C_k non vide faire
31    |     |     r ¬ sélectionner aléatoirement un couple dans C_k ;
32    |     |     Insérer_couple (S_k, r) ;
33    |     |     C_k ¬ C_k \ {r} ;
34    |     Fin Tant que
35    Fin Tant que
36  Fin
```

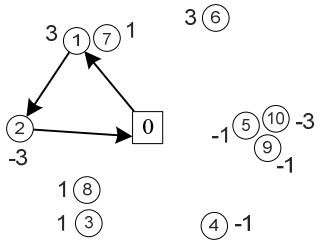
La Figure 4.11 donne un exemple de la construction d'une solution initiale avec un agent. La Figure 4.11 (a) montre le graphe auxiliaire et l'ensemble des couples associés. Elle

correspond à l'instance initiale présentée dans la Figure 4.7 (a). On suppose que le nombre d'agents est fixé à 1. La procédure de construction commence par l'insertion des 2 premiers couples de manière gloutonne. La Figure 4.11 (b) et la Figure 4.11 (c) illustrent respectivement l'insertion des couples R_1 et R_2. Les 3 couples restant vont être insérés de manière aléatoire. On suppose qu'ils sont choisis dans l'ordre R_4, R_5, R_3. Les Figure 4.11 (d) à (f) montrent l'insertion de ces couples. La dernière donne la solution auxiliaire obtenue. La Figure 4.11 (g) est la projection de cette solution sur l'instance initiale.

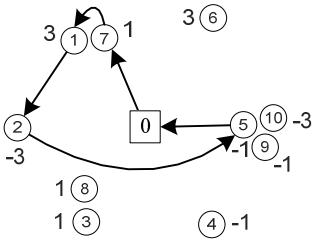


Les couples offre-demande :
$R_1 : (1, 2, 3)$
$R_2 : (7, 5, 1)$
$R_3 : (3, 4, 1)$
$R_4 : (8, 9, 1)$
$R_5 : (6, 10, 3)$

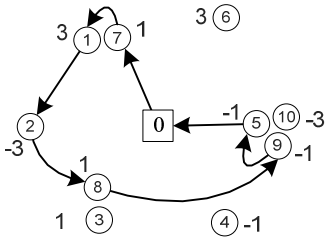
(a) instance auxiliaire



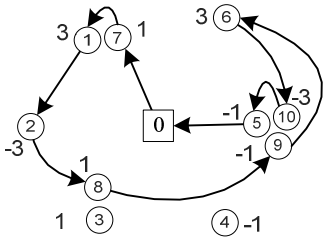
(b) Insertion du couple R_1



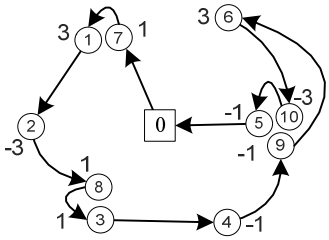
(c) Insertion du couple R_2



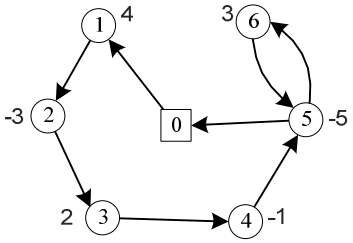
(d) Insertion du couple R_4



(e) Insertion du couple R_5



(f) Insertion du couple R_3



(g) Projection sur l'instance initiale

Figure 4.11 : Construction d'une solution auxiliaire

4.4.2.3 Opérateurs d'amélioration locale et Recherche locale

Dans l'heuristique proposée, la recherche locale utilise les opérateurs de type « *Remove-And-Reinsert* » (RAR). L'idée de ce type d'opérateurs vient du travail de *Ropke* et *Pisinger* [Rop06]. Ils utilisent les opérateurs RAR fonctionnant sur les requêtes de type 1-1 dans la recherche locale pour le VRPB. Le principe de ce type d'opérateurs consiste à retirer un certain nombre de requêtes dans la solution courante, puis à les ré-insérer. Une telle stratégie permet de réorganiser la position des requêtes dans la solution courante, afin de l'améliorer. Les opérateurs RAR que nous avons retenus ont été modifiés pour s'adapter aux couples offre-demande. Dans cette section, nous présentons d'abord ces opérateurs avant de décrire le schéma de la recherche locale.

(a) Les opérateurs d'amélioration locale

Un opérateur de type RAR est caractérisé essentiellement par le critère de sélection des couples et le critère d'acceptation. Le critère de sélection est utilisé pour identifier l'ensemble

de coupes à retirer. Le critère d'acceptation évalue une solution voisine obtenue. Nous avons considéré plusieurs choix pour le critère de sélection : (1) sélection par l'ordre de visite dans les tournées; (2) sélection par la combinaison d'un certain nombre de couples; (3) sélection par cluster (lié aux sommets physiques) ; (4) sélection par la combinaison d'un certain nombre de sommets physiques. Pour le critère d'acceptation, nous avons considéré *all-accept*, *best-accept* et *first-accept*. Nous utilisons *first-accept* comme critère d'acceptation dans les opérateurs utilisés. C'est-à-dire, on valide le premier mouvement des couples (retirer et réinsérer) qui améliore la solution courante.

L'Algorithme 4.5 illustre le schéma général d'un opérateur du type RAR. Le critère de sélectionne définit la taille de l'espace des sous-ensembles de couples lié à l'opération. La ré-insertion des couples retirés de la solution courante se fait de façon gloutonne : le couple avec le plus petit coût d'augmentation va être sélectionné et inséré dans une tournée. Notons que la tournée d'insertion pour un couple n'est pas nécessairement la tournée initiale. Les opérateurs RAR utilisés sont donc des opérateurs de type inter-tournés.

Algorithme 4.5 : Schéma d'un opérateur du type RAR (Opérateur_RAR)

Paramètres d'entrée :
G : instance auxiliaire
S : solution auxiliaire
Paramètre de sortie :
S : solution auxiliaire

```
1  Début
2     R ¬ Couples (G) ;
3     Ω ¬ espace des sous-ensembles de R selon le critère de sélection ; // critère de sélection
4     Pour tout sous-ensemble ω dans Ω faire
5     |    S' ¬ S ;
6     |    Retirer les couples de ω dans S' ;
7     |    Ré-insérer les couples retirés dans S' ;
8     |    Si S' est meilleure que S alors                    // critère d'acceptation
9     |    |    S ¬ S' ;
10    |    Fin Si
11    Fin Pour
12 Fin
```

Selon les différents choix pour le critère de sélection, nous nous intéressons plus particulièrement aux 6 opérateurs RAR suivants :

- *k-cpl-autour* : utilisé dans la recherche locale, il commence par créer un ordre de traitement des couples dans chaque tournée de la solution courante, puis parcourt l'ordre de chaque tournée en sélectionnant les k couples autour de la position (couple) actuelle. Les couples retirés sont ensuite ré-insérés de façon gloutonne (comme présenté dans la méthode de construction). Le paramètre k est fixé ici à 3, *i.e. 3-cpl-autour* ;

- *k-cpl-opt* : on utilise ici le critère de sélection (2) mentionné auparavant. À chaque itération, on sélectionne une combinaison de k couples. Afin de réduire le temps de calcul, on trie l'ensemble des couples R dans l'ordre décroissant selon leur coût d'insertion dans la solution actuelle. Au lieu de parcourir toutes les combinaisons de k couples pour tous les éléments de R, on ne prend ici que les N premiers couples où $N \leq |R|$. Le nombre d'itérations est égal à N^k. La valeur de N est fixée par $\lfloor |R| * 0.8 \rfloor$. Le paramètre k est égal à 2 pour *2-cpl-opt* et à 3 pour *3-cpl-opt* ;

- *k-cluster-opt* : on définit par « cluster de niveau 1 » un ensemble de couples liés à un sommet physique. En prenant l'instance sur la Figure 4.11 (a), les couples R_1 (1, 2, 3) et R_2 (7, 5, 1) définissent le « cluster de niveau 1 » pour le sommet physique numéro 1. De même principe, le « cluster de niveau 2 » pour le sommet physique 1 inclut aussi les couples liés directement aux sommets physiques 2 et 5. Le paramètre k représente ici le niveau du cluster à sélectionner pour un sommet physique. Le nombre de couples sélectionnés n'est pas fixé dans cet opérateur car il dépend de la division effectuée sur un sommet physique. Le nombre d'itérations est défini par la taille de l'instance initiale. Nous utilisons *2-cluster-opt* dans la recherche locale ;

- *k-som-opt* : on utilise aussi la définition « cluster de niveau 1 » dans cet opérateur. Au lieu de faire la propagation décrite dans *k-cluster-opt*, on parcourt ici toutes les combinaisons de k sommets physiques. Les couples sélectionnés à chaque itération consistent donc en k « cluster de niveau 1 ». Le nombre d'itérations est égale à M^k où M est le nombre des sommets physiques. Le paramètre k est égal à 2 pour *2-som-opt* et à 3 pour *3-som-opt*. Notons que le *3-som-opt* est utilisé dans la procédure **Post_amélioration** (décrit dans l'Algorithme 4.3).

(b) La recherche locale

La recherche locale repose sur le schéma de VND. Elle utilise les 4 opérateurs inter-tournés suivants : *3-cpl-autour*, *2-cpl-opt*, *2-cluster-opt* et *k-som-opt*. Ces opérateurs sont renommés respectivement de **Opérateur_RAR_1** à **Opérateur_RAR_4**. Notons que l'ordre de ces 4 opérateurs est basé sur les tests effectués dans la section 4.5.2. L'Algorithme 4.6 donne le schéma de la recherche locale : à chaque itération, lorsque la solution ne peut plus être améliorée par l'opérateur actuel, alors on passe à l'opérateur suivant. Les itérations se terminent lors que la solution actuelle ne peut plus être améliorée par les 4 opérateurs.

Algorithme 4.6 : Schéma de la recherche locale (Recherche_locale)

```
    Paramètres d'entrée :
        G : instance auxiliaire
        S : solution auxiliaire
    Paramètre de sortie :
        S : solution auxiliaire
1   Début
2       stop ¬ false ;
3       S' ¬ S ;
4       k ¬ 1 ;

5       Tant que non stop faire
6       |   S' ¬ Opérateur_RAR_k(G, S') ;        // voir l'Algorithme 4.5
7       |   Si S' est meilleure que S alors
8       |   |   S ¬ S' ;
9       |   Sinon Si k < 4 alors
10      |   |   k ¬ k + 1 ;
11      |   |   S' ¬ S ;
12      |   Sinon
13      |   |   stop ¬ vrai ;
14      |   Fin Si
15      Tant que non stop ;
16  Fin
```

4.4.2.4 Projection d'une solution auxiliaire sur l'instance initiale

La solution auxiliaire obtenue est projetée sur l'instance initiale à l'aide de la procédure **Projection_solution**(). La projection consiste simplement à regrouper les sommets logiques dans des sommets physiques. L'Algorithme 4.7 donne le schéma de la procédure de projection, dans lequel la fonction **Sommet_physique**() retourne le sommet physique correspondant à un sommet logique.

Algorithme 4.7 : Projection d'une solution sur l'instance initiale (Projection_solution)

Paramètres d'entrée :
G : instance auxiliaire
S : solution auxiliaire
Paramètre de sortie :
S_p : solution pour l'instance initiale

```
1   Début
2     S_p ¬ ∅ ;
3     dépôt ¬ sommet_dépôt(G) ;
4     Pour k = 1 à |S| faire
5     |   T' ¬ { dépôt } ;
6     |   Pour i ¬ 1 à |S_k| faire
7     |   |   Si Sommet_physique(G, S_k[i]) ≠ dernier sommet de T' alors
8     |   |   |   T' ¬ T' ∪ { Sommet_physique(G, S_k[i])} ;
9     |   |   Fin Si
10    |   Fin Pour
11    |   S_p ¬ S_p ∪ T' ;
12    Fin Pour
13  Fin
```

4.5 Expérimentations

L'approche heuristique proposée est implémentée en C++ (GNU gcc 4.1.2) et expérimentée sur un PC doté deux processeurs AMD opteron 2.3 GHz sous Linux (CentOs 5.4 en 64 bits). Les instances utilisées dans cette partie sont les mêmes que celles présentées dans les chapitres précédents. Les tests sont réalisés sur les petites instances (*i.e.* au maximum 60 sommets). Les résultats expérimentaux du PPRV-PM sont présentés en deux parties : le cas mono agent et le cas multi-agents.

4.5.1 Taille des instances auxiliaires

La création des instances auxiliaires consiste à appliquer l'algorithme d'affectation (présenté dans la section 4.4.1.1) puis à diviser le résultat obtenu en un ensemble de couples offre-demande. Une instance auxiliaire est composée d'un graphe auxiliaire et d'un ensemble de couples offre-demande. Le Tableau 4.2 montre la taille moyenne des instances auxiliaires obtenues en utilisant les différents seuils d'affectation. Ses colonnes sont définies comme suit:

- Groupe : groupe des 10 instances initiales de la même taille ;
- seuil = {10, 5, 3, 2, 1} : taille moyenne (nombre moyen de sommets et nombre moyen de couples, entre parenthèses) des instances auxiliaires correspond à chaque seuil d'affectation.

Notons que dans les instances utilisées, la demande d'un sommet est définie dans l'intervalle $[-10, 10]$. Ainsi, la quantité d'objets affectée dans un couple est inférieure ou égale à 10. De ce fait, lorsque le seuil d'affectation est fixé à 10, on obtient une instance auxiliaire de taille minimum. Les valeurs du seuil d'affectation testées sont {10, 5, 3, 2,

1}. Dans le Tableau 4.2, on constate que la taille moyenne des instances double lorsque le seuil est égal à 5. Elle est multipliée par 5 lorsque le seuil est à 1 (*i.e.* division unitaire).

Instances initiales	Taille moyenne des instances auxiliaires				
Groupe	seuil = 10	seuil = 5	seuil = 3	seuil = 2	seuil = 1
N20q10	35.6 (17.3)	39.4 (19.2)	49.0 (24.0)	61.4 (30.2)	101.4 (50.2)
N30q10	53.4 (26.2)	59.2 (29.1)	73.0 (36.0)	90.2 (44.6)	146.4 (72.7)
N40q10	70.2 (34.6)	76.4 (37.7)	92.0 (45.5)	113.8 (56.4)	184.2 (91.6)
N50q10	91.6 (45.3)	100.4 (49.7)	122.2 (60.6)	150.2 (74.6)	242.0 (120.5)
N60q10	108.8 (53.9)	121.2 (60.1)	146.0 (72.5)	183.0 (91.0)	296.0 (147.5)

Tableau 4.2 : Taille moyenne des instances auxiliaires

4.5.2 Opérateurs d'amélioration et la recherche locale

Le Tableau 4.3 montre le résultat de la comparaison des 6 opérateurs d'amélioration. Les tests sont réalisés en appliquant un seul opérateur sur une solution initiale. Nous utilisons ici 5 groupes d'instances, chaque groupe contient 10 instances de même taille. Le nombre d'agents est fixé à 1 et le seuil d'affectation est fixé ici à 3. Les colonnes du Tableau 4.3 sont données comme suit :

- Groupe : groupe des 10 instances initiales ;
- GAP : pourcentage du taux d'amélioration moyen entre les solutions initiales et les solutions améliorées par l'opérateur ;
- Temps : temps d'exécution moyen en seconde pour chaque opérateur.

	3-autour-opt		2-cluster-opt		2-cpl-opt		2-som-opt		3-cpl-opt		3-som-opt	
Groupe	GAP	Temps	GAP	Temps	GAP	Temps	GAP	Temps	GAP	Temps	GAP	Temps
N20q10	-16.83	<0.01	-18.92	<0.01	-17.30	0.02	-19.46	0.01	-19.54	0.31	-24.99	0.30
N30q10	-20.31	0.01	-23.35	0.01	-20.57	0.07	-23.94	0.05	-24.61	1.78	-28.51	2.07
N40q10	-18.89	0.02	-18.84	0.02	-19.16	0.17	-21.32	0.13	-22.25	5.18	-24.83	6.61
N50q10	-18.71	0.04	-19.20	0.05	-18.96	0.45	-21.80	0.33	-22.94	19.01	-27.32	22.25
N60q10	-17.33	0.06	-17.18	0.86	-16.03	0.88	-19.54	0.63	-20.06	43.14	-24.50	53.26
	-18.41		-19.50		-18.40		-21.21		-21.88		-26.03	

Tableau 4.3 : Comparaison des opérateurs d'amélioration (**seuil = 3**)

Dans le Tableau 4.3, la dernière ligne donne l'amélioration moyenne pour chaque opérateur. Ces opérateurs peuvent donc être classés comme suit : *2-cpl-opt* < *3-autour-opt* < *2-cluster-opt* < *2-som-opt* < *3-cpl-opt* < *3-som-opt*. Dans la procédure de recherche locale (VND), nous n'utilisons que les 4 premiers opérateurs. Les deux derniers opérateurs consomment en effet trop de temps malgré les gains qu'ils permettent d'obtenir. L'ordre d'appel des 4 opérateurs dans VND tient compte des résultats obtenus dans cette partie. L'opérateur *3-som-opt* donne le meilleur taux d'amélioration et il est utilisé dans la procédure de post-amélioration.

Le Tableau 4.4 et le Tableau 4.5 montrent respectivement l'influence du seuil d'affectation sur les opérateurs *2-som-opt* et *3-som-opt*. Les seuils d'affectation testés sont {10, 5, 3, 2, 1}. Les colonnes « Groupe », « Temps » ont les mêmes définitions que dans le Tableau 4.3. Les autres colonnes sont définies comme suit :

- GAP^1 : taux d'amélioration moyen (pourcentage) entre les solutions initiales et les solutions améliorées par l'opérateur ;
- GAP^2 : écart moyen (pourcentage) entre les solutions du seuil et les solutions du seuil 3.

	seuil = 10			seuil = 5			seuil = 2			seuil = 1		
Groupe	GAP^1	GAP^2	Temps	GAP^1	GAP^2	Temps	GAP^1	GAP^2	Temps	GAP^1	GAP^2	Temps
N20q10	-21.06	-0.90	0.01	-18.22	-0.13	0.01	-25.48	-0.29	0.03	-28.68	3.85	0.12
N30q10	-20.82	0.53	0.02	-18.36	0.33	0.03	-22.74	2.62	0.10	-27.34	-0.09	0.46
N40q10	-17.55	0.77	0.06	-19.17	1.58	0.08	-24.46	2.96	0.24	-22.35	3.01	1.08
N50q10	-17.29	1.37	0.15	-18.87	0.95	0.18	-24.54	1.09	0.60	-29.16	2.50	2.78
N60q10	-20.13	0.12	0.27	-20.72	0.98	0.38	-24.54	0.87	1.2	-31.29	1.20	5.73
	-19.37	0.38		-19.07	0.74		-24.35	1.45		-27.76	2.09	

Tableau 4.4 : Influence du seuil d'affectation sur l'opérateur *2-som-opt*

	seuil = 10			seuil = 5			seuil = 2			seuil = 1		
Groupe	GAP^1	GAP^2	Temps	GAP^1	GAP^2	Temps	GAP^1	GAP^2	Temps	GAP^1	GAP^2	Temps
N20q10	-26.14	-0.66	0.13	-22.51	1.26	0.17	-30.59	-0.51	0.58	-35.32	0.97	2.8
N30q10	-25.40	0.62	0.85	-23.55	-0.06	1.11	-28.44	0.98	3.94	-31.06	0.97	17.34
N40q10	-20.84	1.35	3.13	-24.02	-0.12	4.10	-30.51	-0.91	12.85	-28.01	-0.08	59.37
N50q10	-22.63	1.97	9.83	-24.76	0.68	12.46	-28.13	3.59	39.88	-34.61	1.67	196.72
N60q10	-23.82	1.68	22.46	-24.99	1.70	31.28	-28.69	1.51	105.11	-35.57	1.08	480.91
	-23.77	0.99		-23.97	0.69		-29.27	0.93		-32.91	0.92	

Tableau 4.5 : Influence du seuil d'affectation sur l'opérateur *3-som-opt*

Dans le Tableau 4.4 et le Tableau 4.5, on constate que le gain (GAP^1) augmente pour les deux opérateurs lorsque le seuil d'affectation diminue. Ceci a pour conséquence que la qualité des solutions initiales baisse en fonction de l'augmentation de la taille des instances auxiliaires. La colonne GAP^2 montre que le résultat avec seuil = 3 domine les autres seuils en terme de qualité. Par ailleurs, le temps de calcul est proportionnel à la taille des instances auxiliaires.

4.5.3 Résultats avec un agent

Les résultats du PPRV-PM (avec un agent) sont comparés avec les résultats du PPRV Sans Charge Initiale (PPRV-SCI) présentés dans la section 2.5.3. Une solution du PPRV-SCI est une borne supérieure pour le PPRV-PM. Notons que la solution optimale du PPRV-PM peut être obtenue en résolvant le modèle linéaire adapté sur le graphe auxiliaire avec la division unitaire. Cependant le temps de résolution d'un tel modèle est très élevé : environ 64 heures pour une instance à 40 sommets. Or la taille moyenne des graphes auxiliaires avec la division unitaire dépasse souvent 100 sommets (voir la dernière colonne du Tableau 4.2). Par conséquent, nous avons comparé les résultats avec la borne supérieure et nous nous intéressons à la réduction du coût apportée par la présence du passage multiple.

Le Tableau 4.6 montre la comparaison sur les petites instances avec une capacité $Q = \{10, 20\}$ des agents. Le seuil de relaxation est fixé à 3 et le nombre d'itérations de GRASP est fixé à 300. Les colonnes sont classées en 3 groupes : identifiant de l'instance, résultat avec la capacité $Q = 10$ et résultat avec la capacité $Q = 20$. Elles sont définies comme suit :

- {Groupe, Id} : information liée à une instance initiale;
- Réf. : solution de référence (solution du PPRV sans charge initiale) ;
- Best : la meilleure solution obtenue au bout de 100 itérations ;
- %GAP : écart en pourcentage entre la solution de référence et la meilleure trouvée ;
- Itr. : numéro d'itération pour la meilleure solution ;
- Temps : temps de calcul en secondes.

		$Q = 10$					$Q = 20$				
Groupe	Id	Réf.	Best	%GAP	Itr.	Temps	Réf.	Best	%GAP	Itr.	Temps
N20	A	4963	**4759**	-4.11	1	37.09	3816	3816	0.00	1	99.89
	B	4976	**4790**	-3.74	5	23.68	4224	4224	0.00	4	53.79
	C	6333	**6159**	-2.75	190	26.30	4492	4492	0.00	18	37.47
	D	6334	**6059**	-4.34	42	32.30	4767	**4736**	-0.65	24	74.94
	E	6415	**6318**	-1.51	87	40.58	4673	4673	0.00	74	153.62
	F	4805	4805	0.00	43	15.14	4262	4262	0.00	1	49.54
	G	5119	5119	0.00	1	27.19	4399	4399	0.00	1	51.60
	H	5734	**5553**	-3.16	2	36.52	4372	4372	0.00	4	61.11
	I	5130	**4717**	-8.05	4	33.82	4138	4138	0.00	18	61.65
	J	4430	4430	0.00	3	37.29	3815	3815	0.00	1	79.18
N30	A	6403	**6256**	-2.30	54	151.86	5067	**4937**	-2.57	1	209.06
	B	6646	**6611**	-0.53	23	134.36	5109	5109	0.00	3	217.64
	C	6486	**6348**	-2.13	34	127.65	4901	4901	0.00	5	186.11
	D	6652	**6380**	-4.09	33	143.52	5467	**5344**	-2.25	31	192.20
	E	6070	**6068**	-0.03	1	117.98	5011	5011	0.00	249	314.23
	F	5737	5737	0.00	289	92.71	4583	4583	0.00	5	124.96
	G	9371	**9075**	-3.16	31	170.40	6672	**6550**	-1.83	6	233.77
	H	6638	**6343**	-4.44	114	127.27	4689	4689	0.00	24	182.15
	I	5821	**5596**	-3.87	59	100.20	4487	4487	0.00	1	230.16
	J	6271	**5924**	-5.53	56	94.76	4936	**4706**	-4.66	41	251.17
N40	A	7297	**7209**	-1.47	167	336.12	**5481**	5513	0.58	343	823.63
	B	6647	**6193**	-6.83	21	220.63	5462	5462	0.00	82	388.66
	C	**7528**	7567	0.52	29	307.12	5799	**5773**	-0.45	266	499.25
	D	8300	**7935**	-4.48	43	321.94	6253	**6071**	-2.91	203	653.05
	E	7142	**6843**	-4.19	31	341.98	**5778**	5858	1.38	5	1033.46
	F	7591	**7312**	-3.36	221	247.04	5491	5491	0.00	1	451.37
	G	7757	**7512**	-3.20	245	239.59	5588	5588	0.00	91	404.81
	H	6794	**6683**	-0.25	38	265.87	5306	5306	0.00	188	435.55
	I	7223	**7091**	-1.97	234	372.15	5348	5348	0.00	48	698.39
	J	6741	**6656**	-1.26	181	218.55	5386	**5359**	-0.50	167	408.93

Tableau 4.6 : Résultat sur les petites instances avec un agent (seuil $= 3$)

Dans le Tableau 4.6, pour les instances avec capacité $Q = 10$ ($Q = 20$), le taux de réduction moyen du coût vaut -2.766% (-0.065%) pour le groupe à 20 sommets et -2.648% (-0.189%) pour le groupe à 40 sommets. Nous constatons que la réduction est beaucoup plus faible pour les instances avec capacité $Q = 20$. Le bénéfice lié à la contrainte de passage multiple diminue logiquement lorsque la capacité augmente.

Le Tableau 4.7 montre la comparaison des résultats pour différentes valeurs de seuil d'affectation. Le nombre d'itérations de GRASP est fixé à 300. Les colonnes sont définies comme suit :

- {Groupe, Id} : information liée à une instance ;
- seuil = {10, 5, 3, 2, 1} : résultat correspondant à chaque seuil d'affectation ;
- Best : meilleure solution obtenue;
- Temps : temps de calcul en secondes.

Instance		seuil = 10		seuil = 5		seuil = 3		seuil = 2		seuil = 1	
Groupe	Id	Best	Temps	Best	Temps	Best	Temps	Best	Temps	Best	Temps
N40q10	A	7247	88.50	7281	107.72	7209	336.12	7190	634.05	**7164**	4344.04
	B	6312	90.88	**6193**	117.22	6193	220.63	**6193**	535.12	6234	3312.45
	C	7606	132.85	7606	165.17	**7567**	307.12	**7567**	785.71	7651	5128.71
	D	7939	142.78	7960	156.31	7935	321.94	7928	1016.64	**7928**	5414.26
	E	**6843**	135.22	**6843**	245.69	**6843**	341.98	**6843**	779.55	6951	4467.01
	F	7404	95.07	7389	123.88	**7312**	247.04	7336	552.32	7365	3815.96
	G	7516	83.14	7519	108.94	7512	239.59	7509	481.59	7532	2912.82
	H	**6683**	97.82	**6683**	109.97	**6683**	265.87	6777	514.15	6751	3644.58
	I	7099	161.13	7088	201.40	7091	372.15	**7081**	855.83	7107	5187.72
	J	6658	107.87	**6529**	157.47	6656	218.55	6656	552.90	6588	4404.58
N50q10	A	**6881**	317.48	6885	303.08	6885	831.86	6899	1557.20	6914	9814.27
	B	9284	290.27	9284	376.14	9274	884.00	9365	2313.10	**9185**	13302.57
	C	9088	349.66	9083	434.54	**8956**	986.32	9083	2289.61	9355	12254.18
	D	10227	274.16	10062	431.12	**10045**	972.35	10152	2063.56	10170	12453.00
	E	9340	305.89	**9281**	468.45	9370	1159.29	9351	2250.35	9588	15370.75
	F	8101	252.31	**7946**	371.38	8090	776.36	8075	2026.72	8148	17365.49
	G	**7072**	296.58	**7072**	395.13	7074	710.85	7147	1886.11	7327	10958.19
	H	8780	306.09	8809	359.82	**8706**	785.27	8800	1655.12	8870	9706.52
	I	8295	311.56	8245	391.80	8265	950.92	**8190**	1903.29	8285	13283.20
	J	**8449**	318.32	**8449**	459.61	8601	1046.20	8660	2014.77	8597	17403.75
N60q10	A	8493	597.43	8413	778.45	**8371**	1764.85	8373	2957.27	**8371**	20987.78
	B	8652	528.17	**8583**	688.35	8652	1565.46	8834	4392.53	9041	31170.44
	C	9349	404.66	**9332**	553.36	9376	1288.97	9376	3481.66	9493	16203.94
	D	10849	790.03	10852	1069.76	**10843**	2467.05	11017	5491.68	11130	35088.83
	E	**9470**	528.77	**9470**	884.36	9490	1814.20	9572	4114.20	9612	29386.04
	F	8506	412.38	8457	719.95	8476	2023.50	**8419**	3810.40	8589	36975.44
	G	9004	439.73	**8979**	639.81	9061	1625.45	9038	3956.46	9140	25898.77
	H	**8297**	501.59	8599	938.76	8588	1539.22	8806	4683.86	8976	26180.51
	I	9514	472.57	9509	741.01	9409	2028.82	**9387**	3318.15	9567	26927.82
	J	8680	677.02	8561	945.72	**8468**	2002.86	8541	4700.01	8712	29362.12

Tableau 4.7 : Résultats sur les différents seuils d'affectation ($K = 1$)

On remarque que les temps de calcul croissent très fortement lorsque le seuil diminue, c'est-à-dire lorsque l'agrégation diminue. Dans le même temps, ceci ne s'accompagne pas d'une amélioration des valeurs obtenues. De ce fait, un seuil avec une valeur intermédiaire (3 ou 5 par exemple) semble un bon compromis entre les temps de calcul et la qualité moyenne des solutions. Le Tableau 4.8 synthétise les résultats présentés dans le Tableau 4.7. Les colonnes du Tableau 4.8 sont définies comme suit:

- Groupe : groupe de 10 instances initiales de la même taille ;
- GAP : écart moyen en pourcentage entre les solutions du seuil et les solutions du seuil 3 pour chaque groupe d'instances;
- Temps : temps de calcul moyen en seconde sur les instances du groupe.

La dernière ligne donne la moyenne des écarts sur les 3 groupes d'instances. Elle montre que le seuil 5 donne les résultats légèrement meilleurs que ceux du seuil 3. On constate que la réduction du seuil n'offre pas forcément la meilleure qualité. Par contre, elle accompagne une augmentation du temps de calcul non négligeable : pour un groupe d'instances initiales de 40 sommets (de 60 sommets), le temps de calcul augmente environ 37 fois (52 fois).

	seuil = 10		seuil = 5		seuil = 2		seuil = 1	
Groupe	GAP	Temps	GAP	Temps	GAP	Temps	GAP	Temps
N40q10	0.45	113.53	0.10	149.38	0.12	670.79	0.39	4263.21
N50q10	0.26	302.23	-0.19	399.11	0.52	1995.98	1.39	13191.12
N60q10	0.10	535.24	0.03	795.95	0.68	4090.62	2.10	27818.17
	0.270		-0.023		0.440		1.293	

Tableau 4.8 : Comparaison du résultat sur les différents seuils ($K = 1$)

Ceci confirme donc les remarques faites précédemment.

4.5.4 Résultats avec multi-agents

Dans cette partie, les tests sont effectués sur les petites instances et on considère 2 ou 3 agents. Les résultats obtenus sont présentés respectivement dans le Tableau 4.9 et le Tableau 4.10. Le nombre d'itérations de GRASP est fixé à 300. Les colonnes de ces deux tableaux sont définies comme suit :

- {Groupe, Id} : information liée à une instance ;
- seuil = {10, 5, 3, 2, 1} : résultat lié à chaque seuil d'affectation ;
- Durée : durée du redéploiement de la meilleure solution trouvée ;
- Coût : durée totale des K agents de la meilleure solution trouvée ;
- Temps : temps de calcul en secondes.

Instance		seuil = 10			seuil = 5			seuil = 3			seuil = 2			seuil = 1		
Groupe	Id	Durée	Coût	Temps	Durée	Coût	Temps	Durée	Coût	Temps	Durée	Coût	Temps	Durée	Coût	Temps
N20q10	A	**2706**	5410	12.80	**2706**	5410	16.70	**2706**	5410	35.13	**2706**	5410	86.20	**2706**	5410	164.05
	B	**2765**	5456	11.42	**2765**	5456	15.59	**2765**	5456	28.20	**2765**	5456	53.86	**2765**	5456	176.12
	C	**3228**	6355	14.16	**3228**	6355	13.90	**3228**	6355	35.29	**3228**	6355	105.50	**3228**	6355	357.81
	D	**3216**	6280	12.92	**3216**	6280	23.09	**3216**	6280	44.60	**3216**	6280	121.05	**3216**	6280	574.52
	E	**3368**	6708	11.05	**3368**	6708	28.91	**3368**	6708	43.91	**3368**	6708	131.81	**3368**	6708	533.24
	F	**2572**	4983	7.66	**2572**	4983	15.10	**2572**	4983	17.25	**2572**	4983	45.03	**2572**	4983	124.66
	G	3047	5796	22.03	3047	5796	22.14	3047	5796	27.75	**2912**	5820	112.58	**2912**	5820	237.62
	H	3241	6382	14.03	**3141**	6274	13.96	**3141**	6274	68.00	**3141**	6274	72.09	**3141**	6274	265.35
	I	2750	5410	9.62	2750	5410	17.23	**2650**	5271	43.47	**2650**	5271	122.21	2660	5312	420.08
	J	**2753**	5315	12.80	**2753**	5315	15.71	**2753**	5315	35.94	**2753**	5315	99.79	**2753**	5315	236.26
N30q10	A	3433	6863	71.05	3477	6943	78.12	3477	6943	183.60	**3421**	6830	460.46	3433	6863	1135.30
	B	**3596**	7046	67.17	3727	7411	75.24	3727	7411	149.84	3784	7558	643.64	3729	7437	782.08
	C	**3413**	6802	57.63	3460	6846	95.48	3460	6846	190.16	3440	6860	564.69	3448	6890	1801.95
	D	3502	6556	67.36	3471	6805	119.96	3471	6805	222.41	**3465**	6519	510.67	**3465**	6519	1834.37
	E	3545	7070	51.61	3525	7039	85.73	3525	7039	177.80	**3522**	7023	322.83	3525	7039	1120.95
	F	3193	6246	47.30	3193	6236	60.59	3193	6236	174.81	3193	6236	316.19	**3152**	6269	854.49
	G	**4807**	9476	69.58	4857	9538	122.01	4857	9538	207.53	4845	9678	694.20	4842	9679	2294.30
	H	3523	6975	71.28	3582	7141	96.34	3582	7141	198.30	**3495**	6981	404.21	3518	7029	1422.08
	I	3114	5991	50.46	2997	5933	53.91	2997	5933	152.91	**2923**	5823	313.95	2951	5879	1559.21
	J	3469	6846	36.46	**3440**	6817	73.44	**3440**	6817	154.66	3478	6945	289.05	3460	6900	976.51
N40q10	A	3748	7493	134.55	3745	7488	144.47	3745	7488	560.97	3745	7396	924.04	**3734**	7455	4147.02
	B	3292	6525	126.57	**3235**	6430	164.48	**3235**	6430	350.49	**3235**	6430	785.22	**3235**	6430	2854.67
	C	**3910**	7803	185.89	3943	7852	260.47	3943	7852	457.75	3943	7847	1087.63	3978	7920	5015.98
	D	4236	8467	190.75	4204	8377	245.83	4204	8377	369.18	**4177**	8333	1394.84	4179	8351	4265.44
	E	**3596**	7132	165.54	3623	7159	371.65	3623	7159	538.39	3643	7252	1255.81	**3596**	7132	3540.40
	F	3978	7878	153.40	3953	7818	207.14	3953	7818	458.47	**3898**	7699	882.41	**3898**	7758	3421.33
	G	**3851**	7676	118.41	**3851**	7676	173.73	**3851**	7676	448.46	3890	7726	807.92	**3851**	7670	3256.60
	H	3555	7080	137.45	3555	7080	145.73	3555	7080	417.08	3612	7145	665.78	**3520**	7029	2784.41
	I	3970	7936	138.72	3985	7815	210.25	3985	7815	578.39	3912	7800	1192.06	**3862**	7711	3956.01
	J	3577	7139	158.51	**3562**	7121	221.52	**3562**	7121	334.99	3578	6995	908.81	3569	7112	3113.12

Tableau 4.9 : Résultats avec deux agents ($K = 2$)

Instance		seuil = 10			seuil = 5			seuil = 3			seuil = 2			seuil = 1		
Groupe	Id	Durée	Coût	Temps	Durée	Coût	Temps	Durée	Coût	Temps	Durée	Coût	Temps	Durée	Coût	Temps
N20q10	A	**2134**	6163	17.79	**2134**	6163	28.66	**2134**	6163	35.88	**2134**	6007	81.30	**2134**	6022	211.40
	B	**2115**	6027	15.46	**2115**	6027	13.46	**2115**	6027	23.19	**2115**	6027	43.16	**2115**	6027	135.19
	C	2511	6974	13.43	2511	6974	13.52	**2508**	7449	35.61	**2508**	7449	108.09	**2508**	7449	386.72
	D	**2414**	6927	14.40	**2414**	6927	26.64	**2414**	6927	83.13	**2414**	6927	123.67	**2414**	6927	579.90
	E	2484	7297	12.14	**2414**	7083	23.37	**2414**	6822	48.41	**2414**	6930	110.84	**2414**	6822	467.00
	F	**2041**	6101	6.08	**2041**	6101	11.93	**2041**	6101	14.26	**2041**	6101	32.20	**2041**	6101	114.17
	G	**2091**	5955	16.95	**2091**	5955	16.79	**2091**	5955	27.77	**2091**	5961	75.39	**2091**	5961	157.92
	H	**2513**	7423	11.98	**2513**	7423	18.38	**2513**	7423	52.97	**2513**	7423	78.31	**2513**	7423	227.93
	I	**1945**	5576	11.94	**1945**	5576	18.55	**1945**	5576	42.02	**1945**	5576	134.31	**1945**	5576	223.27
	J	**2177**	6271	8.55	**2177**	6271	11.24	**2177**	6290	38.41	**2177**	6290	76.18	**2177**	6290	170.77
N30q10	A	**2481**	7267	75.40	**2481**	7267	80.49	**2481**	7267	188.03	**2481**	7267	499.68	**2481**	7267	1218.22
	B	**2564**	7304	60.65	**2564**	7304	64.98	**2564**	7304	144.79	**2564**	7304	216.41	**2564**	7304	703.73
	C	**2505**	7431	59.97	**2505**	7431	77.62	**2505**	7431	158.66	**2505**	7385	463.64	**2505**	7385	1540.27
	D	2487	7394	59.44	2487	7377	103.93	**2438**	7144	164.81	2455	7265	514.95	**2438**	7144	1419.33
	E	2534	7527	48.90	2569	7381	72.63	**2534**	7346	194.02	**2534**	7346	363.18	**2534**	7346	1090.81
	F	**2385**	7036	43.54	**2385**	6979	64.32	**2385**	6979	149.30	**2385**	6758	259.30	**2385**	6798	698.17
	G	**3408**	9968	61.08	**3360**	10052	110.48	3374	10089	226.36	3374	10029	544.70	3366	9986	2034.78
	H	2463	7291	65.21	2463	7291	82.57	**2456**	7346	176.08	**2456**	7346	380.34	**2456**	7346	1178.17
	I	2238	6337	30.52	**2215**	6292	53.80	**2215**	6292	136.61	2238	6337	334.42	**2215**	6292	1074.18
	J	2543	7606	52.79	2543	7606	81.64	2543	7606	146.29	**2537**	7577	342.54	**2537**	7577	1053.20
N40q10	A	2740	7874	113.81	2691	8011	118.90	2658	7940	416.76	**2651**	7929	711.30	**2651**	7779	3161.53
	B	2615	7772	109.61	**2550**	7508	146.46	**2550**	7508	279.34	**2550**	7508	653.72	**2550**	7508	2470.26
	C	2784	8276	157.00	2794	8254	208.13	2784	8168	420.07	**2769**	8149	1092.42	2813	8299	3888.08
	D	3034	8977	168.51	3035	9075	192.76	3034	8977	335.33	2997	8792	1163.34	**2991**	8911	3318.14
	E	**2665**	7943	159.63	**2665**	7943	263.42	2674	7938	389.06	2674	7895	830.23	**2665**	7943	3119.14
	F	2821	8379	127.61	2823	8325	170.49	**2775**	8308	383.90	**2775**	8308	729.64	2789	8241	3107.55
	G	**2855**	8368	99.59	**2855**	8368	137.94	**2855**	8368	293.88	**2855**	8324	645.88	2859	8328	1912.15
	H	2698	8078	114.82	2700	7998	132.39	2689	8041	329.65	2700	7925	621.40	**2673**	7855	2293.84
	I	2878	8553	177.73	2821	8445	218.14	**2807**	8281	385.87	2888	8585	1046.75	**2807**	8281	3003.91
	J	**2547**	7522	137.45	**2547**	7449	206.39	**2547**	7522	298.08	**2547**	7449	813.11	**2547**	7449	3019.15

Tableau 4.10 : Résultats avec trois agents ($K = 3$)

Le Tableau 4.11 et le Tableau 4.12 comparent respectivement les résultats présentés dans le Tableau 4.9 et le Tableau 4.10. Les colonnes de ces deux tableaux de comparaison possèdent la même définition que celles du Tableau 4.8. La dernière ligne de chaque tableau donne la moyenne des écarts sur les 3 groupes d'instances testées. On constate que la qualité des résultats est proportionnelle à la réduction du seuil d'affectation. Au niveau du temps de calcul, les résultats avec 3 agents consomment légèrement moins de temps que 2 agents.

	seuil = 10		seuil = 5		seuil = 2		seuil = 1	
Groupe	GAP	Temps	GAP	Temps	GAP	Temps	GAP	Temps
N20q10	0.70	12.80	0.38	18.23	-0.44	95.01	-0.41	308.97
N30q10	-0.16	58.99	0.12	86.08	-0.38	451.99	-0.49	1378.12
N40q10	0.34	150.98	0.17	214.53	0.14	990.45	-0.44	3635.50
	0.293		0.223		-0.227		-0.447	

Tableau 4.11 : Comparaison des résultats pour les différents seuils ($K = 2$)

	seuil = 10		seuil = 5		seuil = 2		seuil = 1	
Groupe	GAP	Temps	GAP	Temps	GAP	Temps	GAP	Temps
N20q10	0.34	12.87	0.01	18.25	0.00	86.34	0.00	267.43
N30q10	0.43	55.75	0.33	79.25	0.15	391.92	-0.05	1201.09
N40q10	0.98	136.58	0.39	179.50	-0.16	830.78	-0.09	2929.38
	0.583		0.243		0.003		-0.047	

Tableau 4.12 : Comparaison des résultats pour les différents seuils ($K = 3$)

Dans l'ensemble, on observe que les seuils de valeur 1 et 2 sont ceux qui permettent d'obtenir les meilleurs résultats, au prix d'un temps de calcul conséquent. Au vu du temps prohibitif consommé lorsque le seuil est unitaire, il est sans doute préférable dans un contexte industriel d'utiliser la valeur 2, voire 3. Dans un contexte fortement réactif (obtention d'une réponse en quelques secondes), on constate que même un seuil égal à la capacité des véhicules permet d'obtenir des solutions de qualité raisonnable.

4.6 Conclusion

Dans ce chapitre, nous avons abordé un problème de type PPRV avec la contrainte de division de demande. Nous nous concentrons ici sur le traitement de la version sans transfert, *i.e.* PPRV-PM. Une approche heuristique à deux phases est proposée pour la résolution du PPRV-PM. Elle repose sur un schéma hybride GRASP/VND et utilise une stratégie de type « *divide-first, route-second* » en deux phases. Dans un premier temps on divise les demandes en sous-demandes. Ceci modifie les caractéristiques de l'instance initiale et le graphe auxiliaire produit sert de support à la résolution pour les étapes suivantes. La phase 2 vise à construire une solution, c'est-à-dire construire un ensemble de tournées, sur ce graphe auxiliaire.

La phase 1 de l'approche consiste à construire un graphe auxiliaire. Elle utilise un modèle d'affection pour créer un couplage offre-demande entre les sommets de collecte et les sommets de livraison tout en minimisant le coût total d'affectation. La résolution de ce modèle utilise l'algorithme du simplexe dédié au problème de transport. Elle permet de calculer rapidement l'affectation optimale. Un graphe auxiliaire contenant un ensemble de couples offre-demande est obtenue en effectuant une division sur le résultat de couplage.

La phase 2 consiste à appliquer une heuristique sur le graphe auxiliaire. Dans cette heuristique, la procédure de construction et la recherche locale utilisent des opérateurs de type *Remove-And-Reinsert* (RAR) reposant sur les couples offre-demande. La solution obtenue à la fin de la phase 2 sera ensuite projetée sur le graphe initial. La recherche locale est une VND avec 4

opérateurs. Chaque itération de GRASP fait donc appel consécutivement à la phase 1 puis à la phase 2. On retient la meilleure solution parmi celles obtenues à la fin de chaque itération.

Les tests dans la partie d'expérimentation sont effectués sur les petites instances (de 20 à 60 sommets). Le résultat avec un agent montre que la présence de la contrainte de division de demande permet un taux de réduction intéressant lorsque la capacité des agents est petite. Nous avons également comparé les résultats pour différents seuils d'affectation. On constate que la méthode donne le meilleur taux de réduction lorsque la valeur du seuil est égale à 3. Les résultats expérimentaux montrent aussi le bénéfice du passage multiple lié à la division de la demande lorsqu'on compare par rapport au PPRV sans charge initiale (PPRV-SCI) qui impose un passage unique sur chaque sommet.

Au niveau de l'heuristique proposé, on remarque que l'affection des sommets dans les couples offre-demande n'est jamais remise en question. Il serait intéressant de pouvoir remonter à l'étape de division des demandes afin d'ajuster un certain nombre de couples avant de repasser dans la phase de résolution. Les informations fournies par la solution courante donnent en effet des indications sur les couples dont la modification permettrait d'améliorer la qualité de la solution. Par ailleurs, une comparaison avec une heuristique classique, munie des opérateurs fonctionnant sur les sommets/arcs, dans la phase de résolution permettrait aussi de juger de la pertinence de l'approche actuelle.

Chapitre 5

PPRV avec transfert d'objets : définition et modélisation

Sommaire

5.1 Introduction 145
5.2 Le Problème des Flots Entiers Couplés (PFEC) 147
5.2.1 Définition du Problème des Flots Entiers Couplés 147
5.2.2 Traitement du PFEC Acyclique dans un cadre statique 148
5.3 Modélisation du PPRV-T comme PFEC-A posé sur un réseau dynamique 156
5.3.1 Reformulation du PPRV-T en PFEC-A posé sur un réseau dynamique 156
5.3.2 Application du schéma Insertion-PFEC-A au PFEC-A-Dyn 158
5.4 Expérimentations 159
5.5 Conclusion 160

Ce chapitre aborde le Problème de la Planification du Redéploiement de Véhicules partagés avec Transfert d'objets (PPRV-T) dans lequel les objets peuvent être échangés entre les agents lors de leur passage sur les différents sites de service. Comme mentionné dans le chapitre précédent, le PPRV-T peut être considéré comme une extension du PPRV avec la contrainte de division de demande autorisant l'opération de transfert entre agents. Il présente de nombreuses similarités avec le *preemptive Swapping Problem* (*preemptive* SP). Ce dernier est lui aussi un PDP de type M-M avec le transfert d'objets. Afin de pouvoir exposer notre approche pour le PPRV-T, nous avons introduit ici un problème auxiliaire, dit Problème de Flots Entiers Couplés (PFEC). Le PPRV-T est modélisé ainsi comme un PFEC Acyclique posé sur un réseau « dynamique ».

Dans ce chapitre, la section 5.1 présente la contrainte de transfert d'objets. Le problème des flots entiers couplés (PFEC) sera étudié dans la section 5.2. Nous proposons une modélisation du PPRV-T en un PFEC Acyclique sur un réseau dynamique dans la section suivante.

5.1 Introduction

Le Problème de la Planification du Redéploiement de Véhicules partagés avec Transfert d'objets (PPRV-T) est défini dans le même contexte que le PPRV-PM (présenté dans la section 4.2) : soit un graphe complet $G = (V, E)$ où $V = \{0, .., n\}$ est l'ensemble des sommets et $E = \{(i, j) | \forall i, j \in V, i \neq j\}$ est l'ensemble des arcs. L'ensemble V est composé du dépôt central (numéroté 0) et des sites de service. Chaque site i est associé à une demande D_i qui exprime la présence sur le sommet i d'un déficit/excédent en objets dont le type est indifférencié. On dispose un coût C_{ij} sur chaque arc (i, j). Un ensemble de K agents identiques de capacité Q est disponible pour transporter les objets. L'objectif du PPRV-T consiste à déterminer la trajectoire de chaque agent en satisfaisant toutes les contraintes (demandes, capacités, logique de transport) avec un coût de redéploiement minimum.

Dans le PPRV-T, la contrainte de « voyage direct » d'un objet entre son sommet d'origine et son sommet de destination (l'un des sommets de livraison) est relâchée. Autrement dit, un objet collecté peut nécessiter plusieurs « correspondances » avant d'arriver à sa destination. On impose que chaque correspondance ait lieu sur un sommet (de collecte ou de livraison) : l'objet est déposé temporairement sur le sommet et il sera chargé plus tard par un agent (le même ou un autre) pour poursuivre son voyage. On autorise ainsi le transfert d'objets.

Il existe essentiellement deux types de transferts selon le nombre d'agents impliqués dans l'opération : dans le premier cas on cherche à réduire le coût de transport en coordonnant plusieurs agents, chaque agent prenant en charge une partie du déplacement. Le transfert entre les agents se fait sur les points de correspondance. Dans le deuxième cas, un agent dépose temporairement des objets sur un site de correspondance, puis effectue un circuit avant de revenir sur ce site pour récupérer sa charge. Notons que dans le premier cas, on doit aussi tenir compte du temps d'attente nécessaire à la synchronisation des échanges entre les deux agents dans le cas où l'agent devant charger les objets est arrivé avant celui qui doit les déposer.

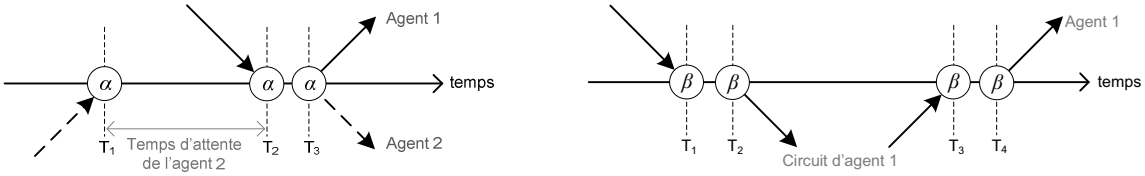
Dans le PPRV-T, nous tenons compte des deux types de transferts, ce qui rend ce problème très difficile à modéliser. Une modélisation sur un graphe classique ne permet pas de lever toutes les ambiguïtés liées à la synchronisation temporelle sur un sommet de correspondance/rendez-vous. La Figure 5.1 illustre deux situations typiques : sur la Figure 5.1 (a), on considère que le sommet α est le point de transfert entre les deux agents. Est-ce que les deux agents sont ponctuels pour le rendez-vous sur ce sommet ? Rien ne permet de lever l'ambiguïté. Sur la Figure 5.1 (b), lorsque l'agent dépose temporairement sa charge sur le sommet β, comment tenir compte de l'évolution dans le temps de sa demande ?

(a) transfert entre deux agents (b) transfert avec un agent

Figure 5.1 : Limitations de modélisation classique

Dans la littérature, le *preemptive* SP autorise lui-aussi la contrainte de transfert. En fait, le *preemptive* SP peut être considéré comme une version simplifiée du PPRV-T. Car il n'utilise qu'un seul agent de capacité unitaire. Évidement, seul le deuxième type de transfert ne peut exister dans ce problème. Avec les simplifications présentes, le *preemptive* SP peut être modélisé sur un graphe classique.

Afin de pouvoir modéliser le PPRV-T, et surtout les contraintes de rendez-vous, nous proposons ici d'utiliser un graphe auxiliaire dont ses nœuds seront des copies de sommets (du graphe initial) indexées dans le temps. Les deux illustrations (a) et (b) de la Figure 5.2 représentent respectivement les situations (a) et (b) de la Figure 5.1 dans un graphe auxiliaire. Sur la Figure 5.2 (a), l'agent 2 doit recevoir des objets de l'agent 1. Il arrive sur le sommet à la date T_1 et il attente jusqu'à l'arrivée de l'agent 2 à la date T_2 pour effectuer le transfert. La période $[T_2, T_3]$ correspond à la période nécessaire au transfert des objets avant que les agents soient disponibles pour d'autres opérations. Sur la Figure 5.2 (b), le sommet β stocke dans l'intervalle de temps $[T_2, T_3]$ une quantité d'objets laissée par l'agent.



(a) transfert entre deux agents (a) transfert avec un agent

Figure 5.2 : Représentation des deux exemples

Un tel graphe auxiliaire est dit réseau dynamique, car la taille du graphe n'est pas statique et dépend de la trajectoires des agents. Dans ce chapitre, nous proposons une modélisation du PPRV-T comme un problème des flots entiers couplés (PFEC) posé sur un réseau dynamique. Nous commençons par une étude sur le PFEC dans la section suivante.

5.2 Problème des Flots Entiers Couplés (PFEC)

Afin de pouvoir exposer notre approche pour le PPRV-T, nous avons besoin de présenter ici le problème auxiliaire, dit Problème des Flots Entiers Couplés (PFEC) et plus particulièrement le PFEC Acyclique. Nous étudions aussi un schéma basé sur la décomposition de Benders pour ce problème auxiliaire. L'idée d'un tel schéma sera ultérieurement reprise par l'approche pour le PPRV-T (dans la section 5.3).

5.2.1 Définition du Problème des Flots Entiers Couplés

On considère un réseau $G = (V, E)$ où $V = \{s, p\} \cup \{1, .., n\}$ est l'ensemble des sommets et E l'ensemble des arcs. L'ensemble V inclut un sommet source s, un sommet puits p et un groupe de nœuds (numérotés de 1 à n). Chaque arc e est muni d'un coût C_e et d'une capacité inférieure sur le flot d'objets $Inf(e)$. L'ensemble A contient les 4 types d'arcs suivants :

- l'arc de retour (p, s) qui coïncide avec l'arc support e_{sup};
- un groupe d'arcs *émetteurs* (noté E_{Em}) reliant la source s aux sommets de $V\backslash\{s, p\}$;
- un groupe d'arcs *récepteurs* (noté E_{Rec}) reliant $V\backslash\{s, p\}$ au puits p ;
- un groupe (noté E_{Tr}) des arcs de *transport* (ou arcs de couverture) correspondent à ceux qui ne sont ni arc support, ni arcs émetteurs, ni arcs récepteurs.

Parmi ces 4 types d'arcs, la capacité inférieure est nulle ($Inf(e) = 0$) pour l'arc de retour et l'arcs de transport, $e \in \{e_{sup}\} \cup E_{Tr}$. La valeur $Inf(e)$ correspond à la quantité imposée pour le flot d'objet sur les arcs émetteurs et sur les arcs récepteurs. On suppose que la somme des valeurs imposées sur les arcs émetteurs est égale à la somme des valeurs imposées sur les arcs récepteurs.

Il existe deux flots entiers sur le réseau G : le flot d'objets f et le flot de transporteurs F. Sur un arc de transport e, la capacité supérieure du flot d'objets f_e est limitée par la valeur $Q * F_e$, où Q est la capacité unitaire des agents pour le flot F. D'autre part, la quantité totale du flot F est limitée par K (le nombre d'agents disponible).

L'objectif du Problème des Flots Entiers Couplés (PFEC) consiste à chercher sur le réseau G deux flots minimisant le coût $C * F$, tels que :

- (c5.1) pour tout arc e de transport ($e \in E_{tr}$), $Q * F_e \geq f(e)$;

- (c5.2) pour tout arc e dans E, $f(e) \geq inf(e)$;
- (c5.3) $F(e_{sup}) \leq K$;
- (c5.4) F et f sont entiers.

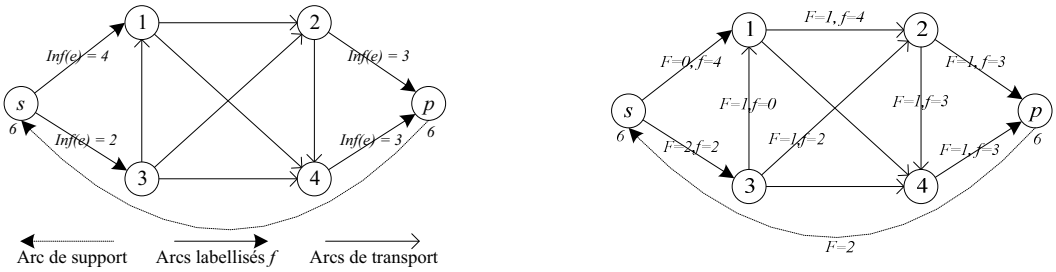
La contrainte (c5.1) est la contrainte de couplage entre les flots posée par la capacité des transporteurs. La contrainte (c5.2) impose la capacité inférieure sur le flot d'objets pour l'ensemble des arcs. La limite sur la quantité totale de flot F est définie par la contrainte (c5.3). La contrainte (c5.4) définit le type des flots.

Le PFEC est un problème assez général qui peut être adapté dans de nombreux contextes. Il peut être particularisé en PFEC Acyclique (PFEC-A) en supposant :

- le réseau G est acyclique si on retire l'arc support e_{sup} ;
- les arcs émetteurs et récepteurs ont une intersection vide avec les arcs de transfert, et sont les arcs labellisés « f ».

Clairement, la sémantique du PFEC-A peut être associée à l'existence d'une flotte homogène de véhicules/agents « transporteurs », représentée par le flot F, qui doit assurer le convoyage d'un ensemble homogène d'objets « transportés », représenté par le flot f. On devine donc une certaine forme de proximité entre le PPRV-T et ce problème, les aspects liés à la gestion du temps et des mécanismes de synchronisation étant cependant absents du PFEC-A.

La Figure 5.3 donne un exemple du PFEC-A. La partie (a) illustre un réseau G et la partie (b) montre une solution sur ce réseau. Notons que la contrainte (c5.1) de couplage entre les deux flots ne s'applique ni aux arcs labellisés « f » ni à l'arc support e_{sup}.



(a) un réseau G (b) une solution sur le réseau

Figure 5.3 : Illustration du PFEC Acyclique

Le PFEC-A peut aussi être étendu sous forme de modèle de multiflot, de manière à intégrer le fait que le flot F « transporteur » peut dériver de l'existence de différents types de véhicules de transport (flotte hétérogène), ou que le flot d'objets f peut dériver de l'existence de plusieurs type d'objets transportés (plusieurs classes d'objets, les objets étant indifférenciés dans leur classe respective). Les modulations rendues ainsi possibles se traduisent essentiellement au niveau de la contrainte de couplage (c5.1).

5.2.2 Traitement du PFEC Acyclique dans un cadre statique

Nous nous intéressons plus particulièrement au traitement du PFEC Acyclique et nous étudions 2 approches : l'une basée sur un schéma de transformation locale reposant sur une décomposition de Benders, l'autre sur un schéma glouton par insertions successives de flots, qui reprend aussi d'une certaine façon la notion de dualité telle qu'elle est utilisée dans les décompositions de Benders.

5.2.2.1 Modèle mathématique du PFEC-A

Nous proposons une formulation du PFEC-A, qui est basée sur les modèles PFEC présentés dans [Quil06a, Quil06b]. Les notions utilisées sont les suivantes :

Données d'entrée :

- $V = \{s, p\} \cup \{1, .., n\}$: ensemble de sommets du réseau G ;
- $E = E_{Tr} \cup E_{Em} \cup E_{Rec} \cup \{e_{sup}\}$: ensemble des arcs, dont :
  - E_{Tr} : arcs de transport ;
  - e_{sup} : arc de support ;
  - E_{Em} : arcs émetteurs ;
  - E_{Rec} : arcs récepteurs.
- $C_{ij} \in \mathbb{N}$: coût associé sur l'arc (i,j) où $(i,j) \in E\backslash\{e_{sup}\}$;
- $Q \in \mathbb{N}$: capacité de transporteur ;
- $K \in \mathbb{N}$: nombre total de transporteurs ;
- $Inf_{ij} \in \mathbb{N}$: capacité inférieure de l'arc (i,j) où $(i,j) \in E\backslash\{e_{sup}\}$;
- $N \in \mathbb{N}$: quantité totale du flot d'objets à acheminer de la source s au puits p.

Variables :

- $F_{ij} \in \mathbb{N}$: flot de transporteurs passant sur l'arc (i,j) où $(i,j) \in E$;
- $f_{ij} \in \mathbb{N}$: flot d'objets sur l'arc (i,j) où $(i,j) \in E\backslash\{e_{sup}\}$.

Le PFEC-A peut alors se formulé comme suit :

$$
(\text{PL5.1}) \begin{cases} \text{Minimiser} \displaystyle\sum_{(i,j)\in E\backslash\{e_{sup}\}} C_{ij}F_{ij} & & \\ \text{s.c.} & & \\ \quad Q * F_{ij} - f_{ij} \geq 0 & \forall (i,j) \in E_{Tr} & (5.1-1) \\ \quad f_{ij} \geq Inf_{ij} & \forall (i,j) \in E\backslash\{e_{sup}\} & (5.1-2) \\ \quad F_{e_{sup}} \leq K & & (5.1-3) \\ \displaystyle\sum_{j:(i,j)\in E} F_{ij} - \sum_{j:(j,i)\in E} F_{ji} = 0 & \forall i \in V & (5.1-4) \\ \displaystyle\sum_{j:(i,j)\in E} f_{ij} - \sum_{i:(j,i)\in E} f_{ji} = \begin{cases} N & i = s \\ -N & i = p \\ 0 & \forall i \in V\backslash\{s,p\} \end{cases} & & (5.1-5) \\ \quad F_{ij} \in \mathbb{N} & \forall (i,j) \in E & \\ \quad f_{ij} \in \mathbb{N} & \forall (i,j) \in E\backslash\{e_{sup}\} & \end{cases}
$$

La fonction objectif vise à minimiser le coût total du flot de transporteurs $\boldsymbol{F}$. Les contraintes $(\mathbf{5.1-1})$ à $(\mathbf{5.1-3})$ correspondent respectivement aux (c5.1) à (c5.3) du PFEC. Les contraintes $(\mathbf{5.1-4})$ et $(\mathbf{5.1-5})$ correspondent respectivement à la conservation du flot de transporteurs et du flot d'objets en chaque sommet.

Le PFEC-A est un problème *NP-difficile*, car lorsque le nombre de transporteurs $K = 1$, l'objectif du PFEC-A revient à l'identification d'un cycle hamiltonien du flot *F* avec le coût minimal. Cela réduit donc le PFEC-A en *Travelling Salesman Problem* (TSP) qui est *NP-difficile*.

5.2.2.2 Application de la décomposition de Benders au PFEC-A

a) Décomposition de Benders sur le PFEC-A

Le principe de la décomposition de Benders est déjà présenté dans la section 1.3.2.1. Elle consiste à partitionner les variables du problème initial en deux vecteurs (dans notre problème f et F), puis à fixer les valeurs du vecteur f afin d'obtenir un problème uniquement avec les variables F (le sous-problème résultant en F est noté SP). Dans la résolution de SP, si les valeurs de f sont réalisables, alors la solution de SP offre un point extrême dans l'espace des solutions du problème initial ; sinon, on obtient un rayon extrême à l'aide du dual de SP qui permet de séparer la zone non réalisable courante (correspondant aux valeurs de f) dans l'espace des solutions. SP permet donc d'obtenir, en fonction de sa réalisabilité, des couples d'optimalité ou de réalisabilité. On obtient le problème maître (noté PM) en regroupant tous les points extrêmes et tous les rayons extrêmes. La résolution de PM est équivalente à la résolution du problème initial [Ben62, Las70].

Nous allons appliquer le principe de décomposition de Benders au problème PFEC-A. On commence par construire un flot f en routant les objets de la source s en direction du puits p. Avec le flot f fixé, on cherche à résoudre le problème en F, noté PFEC-A (f). Le sous-problème PFEC-A(f) est modélisé comme (PL 5.2) en utilisant les notations présentées dans la section 5.2.2.1 :

$$(\text{PL } 5.2)\begin{cases} \text{Minimiser} \displaystyle\sum_{(i,j)\in E\setminus\{e_{sup}\}} C_{ij}F_{ij} & & \\ \text{s. c.} & & \\ Q * F_{ij} \geq f_{ij} & \forall (i,j) \in E_{Tr} & (5.2-1) \\ F_{e_{sup}} \leq K & & (5.2-2) \\ \displaystyle\sum_{j:(i,j)\in E} F_{ij} - \sum_{j:(j,i)\in E} F_{ji} = 0 & \forall i \in V & (5.2-3) \\ F_{ij} \in \mathbb{N} & \forall (i,j) \in E & (5.2-4) \end{cases}$$

Si on relâche la contrainte $(5.2-4)$ en $(5.2-5)$:

$$F_{ij} \geq 0 \quad \forall (i,j) \in E \quad (5.2-5)$$

Nous obtenons alors le problème dual du PFEC-A(f), noté PFEC-A-Dual(f) avec les notations utilisées dans la formulation du PFEC-A et les variables duales suivantes :

- $\lambda_{ij} \geq 0$: variables indexées sur les arcs de E_{Tr} ;
- $\upsilon_{e_{sup}} \geq 0$: variable indexée sur l'arc e_{sup} ;
- $\mu_i \geq 0$: variables indexées sur les sommets de V.

Le modèle mathématique correspondant à PFEC-A-Dual(f) est formulé comme suit (c'est le sous-problème en dans la décomposition Benders) :

$$(\text{PL } 5.3)\begin{cases} \text{Maximiser} \displaystyle\sum_{(i,j)\in E_{Tr}} f_{\text{ij}} * \lambda_{ij} - \text{K} * \upsilon_{e_{sup}} & & \\ \text{s. c.} & & \\ Q\lambda_{ij} + \mu_i - \mu_j \leq C_{ij} & \forall (i,j) \in E\setminus\{e_{sup}\} & (5.3-1) \\ \upsilon_{e_{sup}} \geq 0 & & (5.3-2) \\ \lambda_{ij} \geq 0 & \forall (i,j) \in E_{Tr} & (5.3-3) \\ \mu_i \geq 0 & \forall i \in V & (5.3-4) \end{cases}$$

On note U^p l'ensemble des points extrêmes et U^r l'ensemble des rayons extrêmes. La résolution de PFEC-A-Dual(f) fournit un point ou un rayon extrême : si le sous-problème est réalisable alors sa solution $(\lambda'_{ij}, \upsilon'_{e_{sup}}, \mu'_i)$ est un point extrême dans U^p ; si PFEC-A-Dual(f) n'est pas réalisable, alors la dernière colonne dans le tableau de simplexe $(\lambda''_{ij}, \upsilon''_{e_{sup}}, \mu''_i)$ correspond à un rayon extrême dans U^r. Le problème maître (PM) est formulé comme suit :

$$(\text{PL } 5.4)\begin{cases} \text{Minimiser } z \\ \text{s. c.} \\ z \geq \sum_{(i,j)\in E_{Tr}} f_{\text{ij}} * \lambda^l_{ij} - \text{K} * \upsilon^l_{e_{sup}} & \forall l = 1,..,|U^p| \quad (5.4-1) \\ \sum_{(i,j)\in E_{Tr}} f_{\text{ij}} * \lambda^l_{ij} - \text{K} * \upsilon^l_{e_{sup}} \leq 0 & \forall l \in 1,\dots,|U^r| \quad (5.4-2) \\ f_{ij} \in \mathbb{N} & \forall (i,j) \in E_{Tr} \end{cases}$$

Les contraintes $(5.4 - 1)$ correspondent aux points extrêmes générés. Elle servent à améliorer la solution précédente et sont aussi appelées « contraintes d'optimalité ». Les contraintes $(5.4 - 2)$ correspondent aux rayons extrêmes générés. Elles servent à couper les solutions non-réalisables et sont aussi appelées « contraintes de réalisabilité ».

b) Schéma algorithmique de Benders au PFEC-A

Comme pour la génération de colonnes, la solution du problème maître (PM) est optimale lorsque tous les points extrêmes et les rayons extrêmes sont pris en compte. Il est souvent impossible de générer la totalité des points et des rayons extrêmes. Nous nous intéressons alors plus particulièrement à la résolution de la version restreinte du problème maître (PMR) dans lequel on ne considère qu'un sous-ensemble de U^p et de U^r. Ils sont enrichis itérativement en résolvant le sous-problème (SP) à chaque itération. Notons que la résolution du couple PMR/SP fournit une borne inférieure/supérieure au problème initial (qui est un problème de minimisation) à chaque itération de l'algorithme. Le principe d'un processus de Benders consiste donc à résoudre alternativement le SP et le PMR dans chaque itération, puis mettre à jour les bornes selon la solution obtenue. Les itérations s'arrêtent lorsque la différence entre les deux bornes est inférieure à une valeur ε, arbitrairement faible. La solution trouvée est dite ε-optimale. La Figure 5.4 illustre un exemple sur l'évolution des deux bornes en fonction du nombre d'itérations. On note que la borne inférieure est une fonction monotone croissante en fonction du nombre d'itération. Ce n'est pas le cas de la borne supérieure et l'on doit considérer le minimum des valeurs sur les itérations pour la définir.

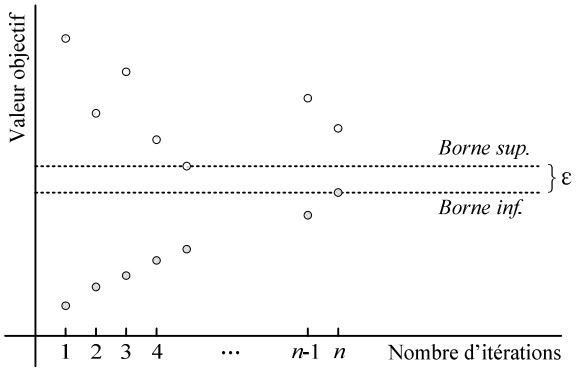


Figure 5.4 : Evolution des deux bornes en fonction des itérations

Dans le cas où le sous problème n'a pas de solution réalisable, on peut extraire une coupe $P = X \cup Y$, telle que (notée C_{coupe}) :

- $s \in X$ et $p \in Y$;
- Aucun arc allant de Y vers X n'existe, excepté l'arc de support e_{sup} ;
- $K \leq \sum_{(i,j):i\in X, j\in Y} f_{ij}$.

En supposant U^c l'ensemble des coupes, alors la contrainte de type rayons extrêmes $(5.4-2)$ peut être remplacée par $(5.4-3)$:

$$K \leq \sum_{(i,j):i\in X, j\in Y} f_{ij} \quad \forall\, (X,Y) \in U^C \quad (5.4-3)$$

L'Algorithme 5.1 montre le processus de Benders, qui fonctionne comme un processus par transformation locale. La solution recherchée est fournie par le couple (f, F), tel que F est une solution de PFEC-A(f) obtenue à la fin des itérations.

Algorithme 5.1 : Schéma de résolution de type Benders

```
1   Paramètres d'entrée :
2       G = (V, E) : graphe orienté acyclique
3
4   Paramètres de sortie :
5       {F, f} : couple de flots sur G
6
7   Début
8     ε ← un petit nombre ;
      UB ← +∞ ;
9     LB ← −∞ ;
10    U^p ← ∅ ;
11    U^r ← ∅ ;
12    Stop ← faux ;
13    Initialiser le flot f ;
14
15    Tant que UB − LB > ε et non Stop faire
16    |     Résoudre PFEC-A(f) ;
17    |     Si Echec alors
18    |     |     Extraire une coupe P = X ∪ Y, telle que C_coupe ;
19    |     |     U^c ← U^c ∪ {P} ;
20    |     |     Recalculer f en résolvant PMR ;
21    |     |     Si Echec alors
22    |     |     |     Stop ← vrai ;
23    |     |     Fin Si
24    |     Sinon   //contrainte de type  point extrême
25    |     |     Calculer la solution primale F et la solution duale (λ*, v*, μ*) ;
26    |     |     UB ← valeur optimale de PFEC-A(f) ;
27    |     |     U^p ← U^p ∪ {(λ*, v*, μ*)} ;
28    |     |     Recalculer f en résolvant PMR ;
29    |     |     Si Echec alors
30    |     |     |     Stop ← vrai ;
      |     |     Sinon
      |     |     |     LB ← valeur optimale de P_r ;
      |     |     Fin Si
      |     Fin Si
      Fin Tant que
      Fin
```

L'Algorithme 5.1 est exact en ce sens qu'il produit la solution optimale, mais sa convergence est lente en pratique, ce qui est un défaut inhérent à la décomposition de Benders classique. Le fait que les deux flots sont imposés entiers complique la mise en œuvre de cet algorithme.

On peut envisager ici une approche heuristique pour recalculer f à l'aide d'un algorithme de coût minimum et un graphe auxiliaire G'. Soit F et f la solution courante sur un réseau G, et Q la capacité unitaire du flot de transporteur. On définit C' le vecteur coût sur les arcs du graphe G' obtenu en dédoublant les arcs de G :

- Si e est un arc de G, tel que $Q * F_e = f_e$ alors $C'_e = C_e$;
- Si e est un arc de G, tel que $Q * F_e > f_e$ alors $C'_e = 0$;
- Si e est l'inverse d'un arc e' de G, tel que $Q * (F_{e'} - 1) < f_{e'}$, alors $C'_e = 0$;
- Si e est l'inverse d'un arc e' de G, tel que $Q * (F_{e'} - 1) \geq f_{e'}$, alors $C'_e = -\alpha(C_{e'} - \varepsilon)$, où α set un paramètre entre 0 et 1, et ε est un bruit uniforme situé entre 0 et $C_{e'}$.

On cherche alors dans le graphe G' un cycle γ (de coût négatif, c'est à dire $\sum_{e\in\gamma} C'_e < 0$) de telle sorte que l'ajout de δf_γ au flot f maintienne f dans les contraintes liées aux U^c et U^p et qu'il diminue le coût $\sum_{e\in G'} C'_e f_e$, où δ est le coefficient d'augmentation du flot f dans le cycle, et f_γ est un vecteur indexé sur les arcs de G' avec les valeurs : 1 si l'arc $e \in \gamma$ est parcouru dans le sens du cycle ; -1 si l'arc $e \in \gamma$ est parcouru dans le sens inverse du cycle ; 0 sinon. Le processus s'arrête si cette recherche échoue. Sinon on recommence la recherche d'un cycle, jusqu'à ce qu'un certain nombre d'itérations est atteint. On conserve alors le meilleur couple (F, f) obtenu.

5.2.2.3 Schéma d'insertion pour le PFEC-A

Nous nous intéressons ici à un algorithme d'insertion qui permet de construire un couple de flots (F, f) sur un graphe acyclique de façon approchée. Dans un tel schéma, on construit le flot d'objets f par choix successifs des plus courts chemins de s à p le long desquels sont routées les valeurs imposées sur les arcs émetteurs et sur les arcs récepteurs. On construit le flot de transporteurs F en même temps que f car dans le processus de sélection d'un plus court chemin, la longueur d'un chemin est évaluée par le coût d'augmentation du flot F. Le détail de ce schéma est décrit dans l'Algorithme 5.2.

Algorithme 5.2 : Schéma d'insertion pour le PFEC-A (Insertion-PFEC-A)

Paramètres d'entrée :
$G = (V, E)$: graphe orienté acyclique
C : vecteur de coût indexé sur les arcs dans $E \backslash \{e_{sup}\}$

Paramètres de sortie :
$\{F, f\}$: couple de flots sur G

1 **Début**
2 $F \longleftarrow 0$; $f \longleftarrow 0$;
3 Fixer la valeur δ, de telle sorte que chaque valeur Inf_e à router, $e \in E_{Em} \cup E_{Rec}$, soit un multiple de δ ;
4 $V^+ \longleftarrow$ ensemble des sommets de destination pour les arcs émetteurs E_{Em}
5 $V^- \longleftarrow$ ensemble des sommets d'origine pour les arcs récepteurs E_{Rec}
6 $Stop \longleftarrow$ faux ;
7 **Tant que** non $Stop$ **faire**
8 | Définir pour l'ensemble des arcs $e \in E \backslash \{e_{sup}\}$, un coût auxiliaire C'_e qui mesure l'augmentation du coût CF, où F est la solution optimale de PFECA(f), induit par une augmentation de δ de la valeur de f_e ;
9 | Choisir un sommet α dans V^+ ;
10 | Choisir un ensemble de sommets $U \subseteq V^-$;
11 | Calculer un plus court chemin γ de s à p en tenant compte du coût auxiliaire C', passant par le sommet α et les sommets de U ;
12 | Router δ unités de flots f le long de γ ;

```
        |
13      |   Tant que ∃ un arc e de γ et e ∈ E_tr, tel que  Q * F_e < f_e faire
14      |   |   Choisir un tel arc e' = (i, j) ;
15      |   |   Chercher un plus court chemin Γ de j vers i dans le graphe réduit G_r(F, f, e') défini par :
        |   |   {Les sommets de G_r sont ceux de G ;
        |   |    Les arcs de G_r sont les arcs de G, avec les arcs e" munis de coûts :
        |   |     - Nuls si Q * F_e" < f_e" ;
        |   |     - − C_e" si e" est dans le sens inverse de la boucle {e'} ∪ Γ , tel que Q * (F_e" − 1) ≥ f_e" ;
        |   |     -  C_e" sinon} ;
16      |   |   Router une quantité unitaire de flot F le long de Γ, et compléter en augmentant le -
        |   |     flot F_e' d'une unité ;
17      |   Fin Tant que
        |
18      |   Mettre à jour V+ en retirant le sommet α si ∃ l'arc e = (s, α) tel que  f_e = Inf_e ;
19      |   Mettre à jour V− en retirant un sommet i si ∃ l'arc e = (i, p) tel que  f_e = Inf_e ;
20      |   Stop ⟵ vrai si V+ (ou V−) est vide.
21      Fin Tant que
22  Fin
```

Notons que la spécification de façon complète de cet algorithme requiert :

- de spécifier les coûts auxiliaires C' sur le graphe G;
- de spécifier le choix du sommet émetteur α et l'ensemble de récepteurs U ;
- de spécifier les algorithmes utilisés pour la recherche des chemins γ et Γ.

(a) Coûts auxiliaires C' sur le graphe G

Le coût auxiliaire C' est dit aussi coût résiduel. Il est calculé en tenant compte de la situation actuelle du flot F sur chaque arc :

- $C'_e = 0$ si $Q * F_e > f_e$;
- $C'_e = C_e + \varepsilon$ si $Q * F_e = f_e$, où ε est un nombre aléatoire selon la loi uniforme sur l'intervalle $[-C_e, C_e]$.

(b) Choix du sommet émetteur α et l'ensemble de récepteurs U

On construit au départ le graphe biparti auxiliaire G_b qui contient les sommets émetteurs et les sommets récepteurs de G. On associe à chaque sommet de G_b la valeur de flots émise/reçue. Un sommet émetteur est adjacent à un sommet récepteur s'il est possible de les relier par un chemin dans le graphe G : pour tout sommet émetteur $\in G_b$, on note N_i^+ pour l'ensemble de ses voisins ; pour tout sommet récepteur $j \in G_b$, on note N_j^- l'ensemble de ses voisins.

Etant donné un couple de flots (F, f) sur G qui correspond à une itération dans l'algorithme de construction ci-dessus (Algorithme 5.2), on peut déduire la valeur L_i^+ de flot f restant à émettre depuis chaque sommet émetteur i et la valeur L_j^- restant à recevoir pour tout sommet récepteur j.

Pour chaque sommet émetteur i, on pose la valeur résiduelle $R_i = \sum_{j \in N_i^+} L_j^-$. On choisit alors α de telle sorte que $R_i - L_i^+$ soit le plus petit possible. On pose ainsi $U = \{j \in N_v^+$ tels que L_j^- non nul$\}$.

(c) Algorithmes pour la recherche des chemins γ et Γ

On suppose ici que le graphe sur lequel on applique la recherche des chemins est un graphe orienté dans lequel chaque arc e est associé à un coût L_e. Plus précisément, on utilise le graphe G_r pour la recherche du chemin Γ et le graphe G avec les coûts auxiliaires C' dans le cas de γ. On suppose que l'on part du sommet α et que l'on cherche à atteindre un ensemble de sommets U, en parcourant des plus courts chemins.

Nous utilisons pour ce faire un algorithme de *Dijkstra* adapté. Il consiste à construire un arbre A permettant de conserver l'ensemble des plus courts chemins. Dans un tel arbre, chaque nœud n contient 3 attributs supplémentaires :

- $\mathrm{N}(n) \in V$: identifiant du sommet associé, où V est l'ensemble de sommets du graphe associé ;
- $\Pi(n) \in \mathbb{Z}^+$: distance pour aller du sommet associé à celui de la racine dans le graphe associé ;
- $\mathrm{M}(n) \in \{0, 1\}$: indicateur d'exploration pendant l'exécution de l'algorithme, =1 si le sommet est déjà exploré, =0 sinon.

Chaque feuille de l'arbre représente le plus court chemin entre la racine et le sommet associé. On obtient le parcours d'un chemin en remontant d'une feuille jusqu'à la racine. Le détail de ce schéma est décrit comme suit :

Algorithme 5.3 : Algorithme de Dijkstra adapté

Paramètres d'entrée :
$G' = (V', E')$: graphe orienté sur lequel on applique la recherche des chemins
L : vecteur de coût indexé sur les arcs E'
α : sommet de départ

Paramètres de sortie :
A : l'ensemble des chemins sous forme d'arbre

Début
  $A \longleftarrow \emptyset$;
  $n \longleftarrow$ Créer_noeud_racine(A, α) ;
  $\Pi(n) \longleftarrow 0$;
  $\mathrm{M}(n) \longleftarrow 0$;
  Nœud_père(A, n) ⟵ Nul ;
  $stop \longleftarrow$ faux ;
  Tant que non $stop$ **faire**
    Chercher le nœud n dans l'arbre A, tel que $\mathrm{M}(n) = 0$ et $\Pi(n)$ soit le plus petit possible ;
    $l \longleftarrow \mathrm{N}(n)$;
    Pour tout nœud n' de l'arbre A, tel que $\mathrm{M}(n') = 0$ et l'arc $(l, \mathrm{N}(n')) \in E'$ **faire**
      $i \longleftarrow \mathrm{N}(n')$;
      Si $\Pi(n) + L_{li} < \Pi(n')$ **alors**
        Nœud_père(A, n') ⟵ n ;
        $\Pi(n') \longleftarrow \Pi(n) + L_{li}$;
      Fin Si
    Fin Pour
    $S \longleftarrow$ Créer l'ensemble des nœuds fils pour le nœud n, tels que l'identifiant $\mathrm{N}(S_i)$ n'existe pas dans A et l'arc $(l, \mathrm{N}(S_i)) \in E'$.
    Pour tout nœud n' dans S **faire**
      Nœud_père(A, n') ⟵ n ;
      $\mathrm{M}(n') = 0$;
      $\Pi(n') \longleftarrow \Pi(n) + L_{li}$;
    Fin Pour
    $\mathrm{M}(n) \longleftarrow 1$;
    $stop \longleftarrow$ vrai si $\nexists\ \mathrm{M}(n') = 0, \forall\ n'$ dans l'arbre A;
  Fin Tant que
Fin

Remarque 5.1

Notons que l'Algorithme 5.3 est optimal quand les coûts sont positifs. Il donc n'est pas optimal pour la recherche de Γ, qui induit la présence de coût négatifs. Cet algorithme devra être ultérieurement adapté de façon spécifique pour la recherche des chemins γ et Γ dans le contexte d'un réseau dynamique dans la section 5.3.2.

5.3 Modélisation du PPRV-T comme PFEC-A posé sur un réseau dynamique

Le PFEC-A a clairement un rapport avec le PPRV-T. Cependant il ne prend pas en compte les contraintes temporelles comme le temps d'attent, ou la durée du processus de redéploiement. Le but de cette section consiste à modéliser le PPRV-T en un problème de multiflot (PFEC-A), en utilisant un réseau qui permet d'établir une correspondance entre ces deux problèmes. Ce réseau sera dynamique, au sens où ses nœuds seront des copies des sommets du V (*i.e.* l'ensemble de sommets du graphe G), indexés sur le temps. Nous allons appliquer ainsi le schéma Insertion-PFEC-A au modèle PFEC-A posé sur le réseau dynamique.

5.3.1 Reformulation du PPRV-T en PFEC-A posé sur un réseau dynamique

5.3.1.1 Création d'un réseau dynamique

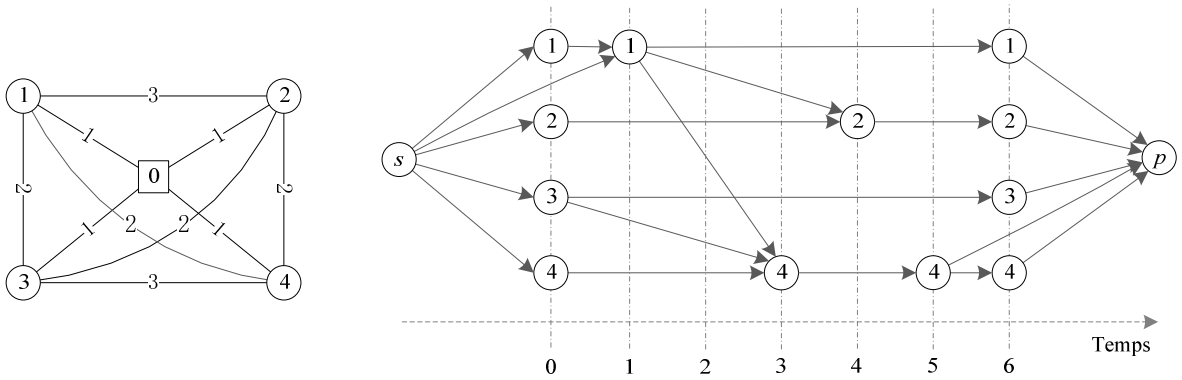
Soit un graphe $G = (V, E)$ où $V = \{0, .., n\}$ est l'ensemble des sommets (avec le dépôt numéroté 0 et les sites numérotés de 1 à n) et $E = \{(i, j) | \forall i, j \in V, i \neq j\}$ est l'ensemble des arcs. Chaque arc (i, j) est associé à un coût de parcours C_{ij} qui correspond au temps nécessaire pour se rendre de i à j. On note un espace de temps $\Delta = [0, T]$ où T est la durée maximale du redéploiement. Un réseau dynamique $G_\Delta = (V_\Delta, E_\Delta)$ est construit à partir de G et Δ. Son ensemble des sommets V_Δ est formé de :

- Deux sommets virtuels : la source s et le puits p ;
- L'ensemble des couples actifs entre v et t (noté $V_\Delta^{v,t}$), où $v \in V$ et $t \in \Delta$.

L'ensemble d'arcs E_Δ consiste en :

- Un arc retour (p, s), muni d'un coût $L_\Delta^{ps} = 1$;
- Les arcs qui connectent la source s aux sommets $V_\Delta^{v,0}$, ainsi que les sommets $V_\Delta^{v,T}$ au puits p, où $v \in V$. Ces arcs sont respectivement notés émetteurs et récepteurs ;
- Les arcs qui connectent la source s aux sommets $V_\Delta^{v,t}$, où $v \in V$ et $t \in \Delta$, tels que $t \geq C_{0v}$; ainsi que des sommets $V_\Delta^{v,t}$ au puits t, où $v \in V$ et $t \in \Delta$, tels que $T - t \geq C_{v0}$; le coût L_Δ^e d'un tel arc e est égal à C_{0v} dans le premier cas et à C_{v0} dans le deuxième cas ;
- Les arc labellisés « Attente », qui connectent tout sommet $V_\Delta^{v,t}$ aux sommets $V_\Delta^{v,t'}$, tel que $t' > t$; un tel arc e est associé à un cout d'attente $L_\Delta^e = t' - t$;
- Les arcs labellisés « Transport », qui connectent tout sommet $V_\Delta^{v,t}$ à tout sommet $V_\Delta^{v',t'}$ tel que $t' \geq t + C_{v,v'}$; un tel arc e est muni d'un coût $L_\Delta^e = C_{v,v'}$: ces arcs constituent les arcs de transport E_{Tr} ;

Un tel réseau dynamique est infini dans le cas où l'espace de temps est continue. Il peut cependant avoir une taille importante même si l'espace de temps est discret. La Figure 5.5 montre un exemple de réseau dynamique. Un graphe G est illustré sur la figure à gauche. Il contient 5 sommets et 10 arcs. Le coût C_e sur chaque arc e est indiqué. Sur la figure à droite, le réseau dynamique G_Δ est construit à partir du graphe G et d'un espace de temps $\Delta = [0, T]$ où T est égal à 6. Sur ce réseau dynamique, l'arc ($V_\Delta^{4,3}$, $V_\Delta^{4,5}$) est labellisé « Attente » par exemple ; les arcs ($V_\Delta^{1,1}$, $V_\Delta^{4,3}$), ($V_\Delta^{1,1}$, $V_\Delta^{2,4}$) et ($V_\Delta^{3,0}$, $V_\Delta^{4,3}$) sont labellisés « Transport ». Notons qu'un réseau dynamique est acyclique en retirant l'arc retour (p, s).



(a) un graphe G (b) un réseau dynamique

Figure 5.5 : Réseau dynamique

5.3.1.2 Reformulation du PPRV-T

L'objectif de cette partie est de montrer comment on peut représenter une solution du PPRV-T par un multiflot sur réseau dynamique G_Δ, afin de ramener le PPRV-T à la recherche des deux flots F et f sur G_Δ. Ils représentent respectivement le flot « transporteurs » et « objets ». Le PPRV-T est ainsi reformulé comme PFEC Acyclique Dynamique (noté PFEC-A-Dyn), tel que :

- F est entier, positif ou nul, et mesure une quantité de transporteurs ;
- f est positif ou nul, les objets à transporter peuvent être mesurés en entier ou en fractionnaire ;
- Sur tout arc e labellisé *transport*, $Q * F_e \geq f_e$, où Q est la capacité de transporteurs ;
- Sur tout arc émetteur $e = (s, V_\Delta^{v,0})$, $f_e = D_v$ où D_v est la demande du sommet v ;
- Sur tout arc récepteur $e = (V_\Delta^{v,T}, p)$, $f_e = -D_v$;
- F est nul sur les arcs émetteurs ou récepteurs ;
- $F_{(p,s)} \leq K$, où K est le nombre total de transporteurs ;
- Le coût total du flot $L_\Delta * F = \sum_{e \in G_\Delta} L_\Delta^e * F_e$ est minimal.

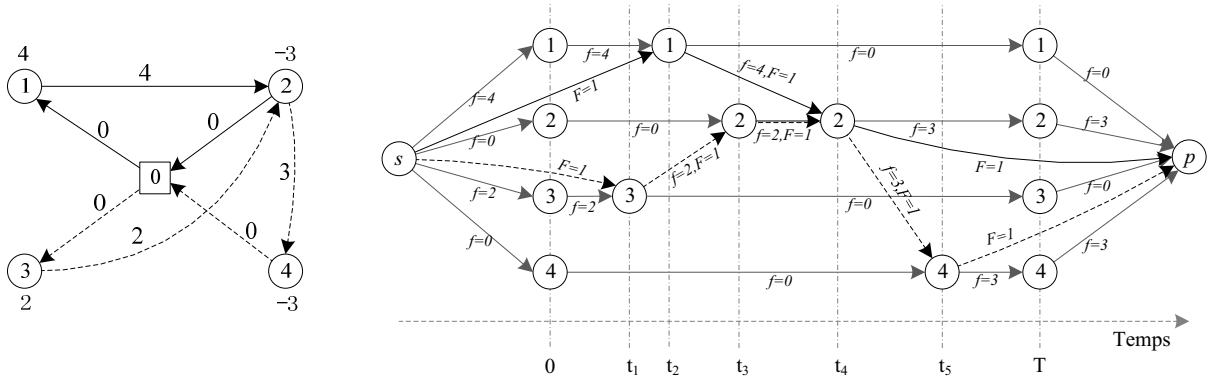
Le PFEC-A-Dyn est une version du PFEC-A, qui présente la particularité d'être posé sur un réseau dynamique. Une solution du PPRV-T peut être interprétée comme un couple de flots (F, f) dans le PFEC-A-Dyn. Réciproquement, étant donné un couple de flots (F, f) qui satisfait les contraintes du PFEC-A-Dyn, on peut reconstruire les parcours des transporteurs/agents et des objets en procédant comme suit :

- On décompose F en une somme de $n = F_{(p,s)}$ flot-circuits, dont chacun représente le parcours d'un agent, noté $F^{\Delta 1}, .., F^{\Delta n}$;

- On décompose le flot f en une somme $f^{\Delta 1}+..+f^{\Delta n}$ de flots positifs ou nuls et tel que :
  - Pour chaque arc e de E_{Tr}, $Q * F_e^{\Delta i} \geq f_e^{\Delta i},\ \forall i = 1,..,n$;

  Chacun des flots $f^{\Delta i}$ représente les objets véhiculés par l'agent transporteur i. Notons qu'une telle décomposition est toujours réalisable.

La Figure 5.6 donne un exemple sur la représentation d'une solution de PPRV-T par le PFEC-A-Dyn. La figure à gauche illustre une solution de PPRV-T avec deux agents transporteurs. La ligne continue (pointillée) représente le parcours de l'agent 1 (l'agent 2), et le nombre sur un sommet (arc) représente la demande du sommet (charge de l'agent). La figure à droite montre la représentation d'une telle solution en PFEC-A-Dyn.



(a) une solution de PPRV-T (b) la représentation en PFEC-A-Dyn

Figure 5.6 : Représentation d'une solution de PPRV-T en PFEC-A-Dyn

Remarque 5.2

Un arc labellisé *transport* ne porte pas la même définition que celui dans le PFEC-A. Dans le PFEC-A-Dyn, un arc $e = (V_\Delta^{v,t}, V_\Delta^{v',t'})$ est labellisé *transport* si $v \neq v', V_\Delta^{v,t} \neq s$ et $V_\Delta^{v',t'} \neq p$.

Remarque 5.3

Dans le PFEC-A-Dyn, on fait abstraction des coûts liés à l'échange d'objets entre les agents transporteurs et notamment des problèmes de fiabilité qui peuvent dériver d'incertitudes sur les durées des parcours et donc sur la difficulté qu'il peut y avoir à finaliser les processus de rendez-vous. Si l'on veut prendre en compte ces coûts, alors il faut définir aussi des coûts sur les sommets qui pénalisent la différence entre la quantité d'objets entrant et sortant d'un même sommet. Ceci n'est pas forcément simple car il s'agira d'une approximation.

5.3.2 Application du schéma Insertion-PFEC-A au PFEC-A-Dyn

Le schéma Insertion-PFEC-A présenté dans la section 5.2.2.3 peut être appliqué directement au PFEC-A-Dyn dès lors que l'on spécifie comment s'effectue la recherche des chemins γ et Γ. Il s'agit donc de transposer l'algorithme Dijkstra adapté (voir Algorithme 5.3) dans le contexte du graphe dynamique G_Δ.

Dans une telle adaptation, il faut tout d'abord expliquer comment on génère l'ensemble des nouveaux voisins S pour le nœud actuel n dans l'arbre A. Dans le cas du réseau dynamique G_Δ, l'identifiant du nœud actuel $\mathrm{N}(n)$ correspond ici à un sommet $V_\Delta^{v,t}$ de G_Δ. Le sommet $V_\Delta^{v,t}$ est un couple actif entre le sommet v (du graphe G) et l'instant t, alors les identifiants pour les nœuds de S sont les couples actifs de la forme $V_\Delta^{u,(t+C_{vu})}$, qui ne sont pas encore présentés dans l'arbre A.

Le nombre de nœuds fils à créer dans S risque d'être assez important à chaque étape. Dans le cas où l'on effectue la recherche du chemin γ pour le routage du flot d'objets f, on peut filtrer l'ensemble S en exploitant le fait que la matrice des coûts C vérifie l'inégalité triangulaire et en appliquant la règle suivante :

- Règle 1 : par analogie avec le représentation du graphe, un arc $(V_\Delta^{v,t}, V_\Delta^{v,t'})$ sera dit *horizontal* ; un arc $e = (V_\Delta^{v,t}, V_\Delta^{u,t+C_{vu}})$ sera dit *oblique* ; un tel arc oblique e sera dit *saturé* si l'on a $f_e = Q * F_e$. L'identifiant du nœud actuel n étant connecté à son père par un arc de type horizontal, oblique saturé ou oblique non saturé, on exigera que les arcs servant à créer les nœuds de S soient d'un type différent à cet arc.

Dans le cas où l'on effectue la recherche du chemin Γ pour le routage du flot F, la règle de filtrage appliquée pendant la création de l'ensemble S est comme suit :

- Règle 2 : un arc sera dit *nouveau* s'il dérive de S ; on fera en sorte que, à l'intérieur du chemin induit par l'arbre A et reliant la racine de A au nœud actuel, il y ait forcément au moins 1 arc oblique dans le bon sens (sens du chemin Γ) et de coût nul entre 2 arcs nouveaux, ou au moins un arc pris à rebrousse-poil.

5.4 Expérimentations

Nous présentons ici les résultats préliminaires du PPRV-T obtenus par l'application du schéma Insertion-PFEC-A sur PFEC-A-Dyn (heuristique d'insertion). Les tests ont été réalisés sur un PC doté deux processeurs AMD opteron 2.3 GHz sous Linux (CentOs 5.4 en 64 bits). Les algorithmes sont programmés en C++ (avec le compilateur GNU gcc 4.1.2). Les instances utilisées sont les mêmes que celles présentées dans les chapitres précédents. Les tests sont effectués sur les petites instances.

Le Tableau 5.1 montre les résultats du groupe d'instances N20q10 avec le nombre d'agents $K = \{1, 2, 3\}$. Les colonnes sont définies comme suit :

- {Groupe, Id} : information liée à une instance ;
- Coût : coût total des K agents de la solution construite ;
- Temps : temps de calcul en secondes pour l'heuristique d'insertion.

Instance		$K = 1$		$K = 2$		$K = 3$	
Groupe	Id	Coût	Temps	Coût	Temps	Coût	Temps
N20q10	A	8993	0.047	8993	0.094	10257	0.047
	B	12686	0.062	11225	0.062	9266	0.037
	C	9080	0.110	9487	0.077	9625	0.126
	D	13677	0.079	12589	0.232	13620	0.156
	E	9010	0.094	9838	0.109	10810	0.620
	F	10053	0.031	11297	0.062	10284	0.063
	G	12179	0.032	12592	0.047	12950	0.031
	H	14783	0.078	14343	0.031	12926	0.035
	I	12595	0.094	11208	0.078	10790	0.094
	J	9065	0.125	8909	0.045	9086	0.046

Tableau 5.1 : Résultats obtenus par l'heuristique d'insertion

Notons que le PPRV-T est un problème nouveau. À notre connaissance, il n'existe pas de benchmark dans la littérature. Par contre, lorsque le nombre d'agents est égal à 1, le résultat du PPRV-PM peut être considéré comme borne supérieure pour le PPRV-T car ils cherchent tous les deux à minimiser le coût total dans ce cas. En comparant le résultat sur le Tableau 5.1 à cette borne supérieure (la colonne « Best » dans le Tableau 4.6), on constate que les résultats obtenus restent encore perfectibles. Plus précisément, l'écart moyen entre les résultats du groupe N20q10 (avec 1 agent) et la borne supérieure est égale à 115.46%.

5.5 Conclusion

Dans ce chapitre, nous nous sommes intéressés au PPRV-T, dans lequel les objets transportés peuvent être échangés entre les agents lors du passage sur les sommets. Nous commençons par l'introduction du Problème des Flots Entiers Couplés Acyclique (PFEC-A). Le PPRV-T est modélisé ensuite comme un PFEC-A posé sur un réseau dynamique. Le déplacement des agents est représenté par le flot d'agents, et les objets transportés sont représentés par le flot d'objets. Ces deux flots sont couplés par la capacité des agents. Le réseau dynamique est un graphe auxiliaire dans lequel l'ensemble des nœuds consiste en une source, un puits et les copies des sommets du graphe initial indexées sur le temps (*i.e.* couple sommet-temps).

Nous avons proposé une heuristique d'insertion basée sur cette modélisation. À chaque itération, on cherche à acheminer une quantité d'objets de la source au puits, puis à ajuster le flot d'agents en satisfaisant la contrainte de couplage entre eux avec un coût de transport minimum. Les itérations se terminent lorsque tous les objets sont acheminés.

Cette heuristique reste à ce jour une méthode de construction. Nous avons présenté quelques résultats préliminaires sur les petites instances. En absence d'une procédure d'amélioration locale, la qualité des résultats reste perfectible. Concernant le futur travail, l'ajout d'une recherche locale à cette heuristique (par manipulation des couples sommet-temps) serait très intéressant. De même, la possibilité de passer de l'espace naturel des solutions de type VRP à l'espace du graphe dynamique permettrait de combiner la performance des méthodes développées dans les chapitres précédents avec la capacité d'expression de l'approche présentée dans ce chapitre. En outre, définir des recherches locales dans cet espace permettrait de définir une approche à espace multiples.

Conclusion

Dans le cadre de cette thèse, nous nous sommes intéressés aux problèmes d'optimisation combinatoire pour les outils gestionnaires aux systèmes de véhicules partagés. Le travail réalisé concerne la modélisation et la résolution de problèmes du redéploiement de convois en tenant compte de différentes situations. Ces problèmes sont proches des problèmes de collecte et de livraison ainsi que des problèmes de flots. Après avoir réalisé une étude théorique des différents problèmes et des méthodes de résolutions, nous nous sommes intéressés à trois problèmes particuliers. Nous proposons pour chaque problème une méthode de résolution.

Dans un premier temps, nous avons abordé le Problème de la Planification du Redéploiement de Véhicules partagés (PPRV). Le PPRV est un PDP de type *Many-to-Many*. Plus précisément, c'est un 1-PDP. Il est considéré comme le problème de base dans le cadre de cette thèse. Nous avons proposé un modèle linéaire, avec lequel on peut résoudre de petites instances du PPRV en utilisant un solveur linéaire comme Gurobi. À notre connaissance, il n'existe pas de *benchmark* pour le 1-PDP avec multi-agents dans la littérature. Le résultat de ce modèle est utilisé comme référence pour les méthodes approchées.

Nous avons proposé ensuite une approche méta-heuristique de type « *route-first, cluster-second* » pour la résolution du PPRV. L'idée de ce type d'approche consiste à construire d'abord une tournée géante, puis à la découper en plusieurs tournées. Nous avons implémenté un schéma ILS/VND, dans lequel le VND est utilisé comme une recherche locale dans ILS. La procédure de perturbation dans ILS se sert d'un voisinage basé sur un modèle d'affectation. La résolution d'un tel modèle utilise un solveur linéaire. La procédure SPLIT est utilisée pour la transformation d'une tournée géante en tournées. Les recherches locales utilisent une adaptation des opérateurs classiques de type *k-opt*. Les opérateurs utilisés pour l'amélioration des tournées géantes relaxent la contrainte de capacité des véhicules. L'expérimentation montre que cette approche offre les résultats satisfaisants : pour le cas avec mono-agent, elle a été comparée à trois méta-heuristiques de la littérature ; pour le cas avec multi-agents, nous l'avons comparé avec le résultat optimal donné par le modèle linéaire.

Dans un second temps, nous avons traité le PPRV avec le Passage Multiple (PPRV-PM). Ce problème est destiné au cas où la capacité d'agent est succeptible plus petite que les demandes. Cette extension consiste à autoriser le splitting d'une demande en autant de demandes que souhaitées, chaque partie de demande pouvant être traitée par un agent différent. Nous proposons une approche de type « *divide-first, route-second* » pour résoudre le PPRV-PM. Le principe de cette approche consiste à pré-calculer le splitting de la demande dans une instance auxiliaire, puis à appliquer un schéma heuristique pour la résolution. La solution obtenue est projetée sur l'instance initiale. L'heuristique de résolution repose sur le schéma hybride GRASP/VND. La recherche locale utilise les opérateurs de type «*Remove-and-Reinsert*» (RAR) basés sur les couples offre-demande. Comme il n'existe pas de *benchmark* pour ce problème dans la littérature, le résultat est comparé avec celui du PPRV sans Charge Initiale, qui donne la borne supérieure au PPRV-PM. On constate que la présence

du passage multiple apporte une réduction du coût plus intéressante sur les instances de petite capacité que celle de capacité moyenne.

Dans un troisième temps, nous nous intéressons au PPRV avec Transfert d'objets (PPRV-T). Dans ce problème, les objets peuvent être échangés entre les agents au niveau des sommets. On tient compte aussi du temps d'attente nécessaire à la synchronisation des échanges. Dans cette partie, nous avons introduit d'abord un problème auxiliaire : le Problème des Flots Entiers Couplés Acyclique (PFEC-A). Le PPRV-T est modélisé ici comme un PFEC-A sur un graphe auxiliaire, dit réseau dynamique. L'ensemble des nœuds dans un réseau dynamique contient une source, un puits et les copies des sommets du graphe initial indexées sur le temps (*i.e.* couple entre un sommet et le temps). Notons que le sens dynamique vient du fait que le nombre des couples n'est pas fixé. Dans une telle modélisation, on utilise deux flots : l'un pour représenter le déplacement des agents et l'autre pour les objets transportés. Ces deux flots sont couplés par la contrainte de capacité. Nous avons proposé ainsi une heuristique basée sur cette modélisation. L'idée principale de cette heuristique vient la décomposition de Benders : on cherche à acheminer le flot d'objets sur le réseau dynamique, puis ajuste l'autre afin de satisfaisant la contrainte de couplage, tout en minimisant le coût total du transport.

Enfin, nous avons abordé plusieurs problèmes liés au redéploiement de véhicule partagé. Parmi ces problèmes, l'un est un complément du problème existant : PPRV (*i.e.* le 1-PDP avec multi-agents) ; les autres sont totalement nouveaux : PPRV-PM (*i.e.* 1-PDP avec la contrainte de division de demande) et PPRV-PM (*i.e.* 1-PDP avec la contrainte de transfert). Au niveau des heuristiques proposées, en basant sur les méthodes existantes, nous avons implémenté des idées originales comme la procédure de perturbation par un modèle linéaire, le schéma « *division-first, route-second* », les opérateurs basés sur les couples, ou encore les idées qui n'ont pas été détaillés dans ce manuscrit, comme la relaxation de contrainte de mono-accès pour la résolution du 1-PDP avec mono-agent.

Dans cette thèse, nous nous sommes intéressés essentiellement à des problèmes d'optimisation combinatoire liés à la gestion des systèmes de véhicules partagés, et aussi aux méthodes de résolution associées. À ce jour, travaux effectués sont évalués sur des instances que nous avons voulues proches de la réalité mais qui sont toutefois des instances académiques.

Bibliographie

[Aar03] E.H.L. Aarts, J. K. Lenstra. *Local Search in Combinatorial Optimization*. Princeton University Press, 2003.

[Aar97] E.H.L. Aarts, J. K. Lenstra. *Local Search in Combinatorial Optimization*. Wiley, Chichester, 1997.

[Aga89] Y. Agarwal, K. Mathur, H.M. Salkin. *A set-partitioning-based exact algorithm for the vehicle routing problem*. Networks, vol. 19, pp. 731–749, 1989.

[Ahu98] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Wiley, 1998.

[Alt91] K. Altinkemer, B. Gavish. *Parallel savings based heuristic for the delivery problem*. Operations Research, vol. 39, pp. 456–469, 1991.

[Ame05] A. Amekudzi, M. Meyer, *Consideration of Environmental Factors in Transportation Systems Planning,* NCHRP Report, 2005.

[Ani11] S. Anily, M. Gendreau, G. Laporte. *The preemptive swapping problem on a tree.* Networks, vol. 58, pp. 83-94, 2011.

[Ani92] S. Anily, R. Hassin. *The swapping problem*, Networks, vol. 22, pp. 419–433, 1992.

[Ani94] S. Anily, G. Mosheiov. *The traveling salesman problem with delivery and backhauls*. Operations Research Letters, vol. 16, pp.11–18, 1994.

[Ani99] S. Anily, J. Bramel. *Approximation algorithms for the capacitated traveling salesman problem with pickups and deliveries*. Naval Research Logistics, vol. 46, pp. 654–670, 1999.

[Ang02] E. Angelelli, R. Mansini. *The vehicle routing problem with time windows and simultaneous pickup and delivery*. In: A. Klose, M.G. Speranza, L.N. VanWassenhove, editors, *Quantitative Approaches to Distribution Logistics and Supply Chain Management*. Springer, Berlin-Heidelberg, pp 249–267, 2002.

[Arc06] C. Archetti, A. Hertz, M.G. Speranza. *A tabu search algorithm for the split delivery vehicle routing problem*. Transport Science, vol. 40, pp. 64–73, 2006.

[Arc08] C. Archetti, M.G. Speranza. *The split delivery vehicle routing problem: A survey. In Golden,* B., Raghavan, R., Wasil, E., editors, *The Vehicle Routing Problem: Latest Advances and New Challenges*. Springer, New York, pp. 103–122, 2008.

[Arc08a] C. Archetti, M.W.P. Savelsbergh, M.G. Speranza. *To split or not to split: That is the question*. Transportation Research, part E, vol. 44, pp.114–123, 2008.

[Arc11] C. Archetti, M. G. Speranza. *Vehicle routing problems with split deliveries*. International transactions in operational research, vol.19, pp. 3–22, 2011.

[Aro89] J.E. Aron,son. *A survey on dynamic network flows*. Annals of Operation Research, vol. 20, pp. 1–66, 1989.

[Arr75] K.J. Arrow. *Choix Collectifs et Préférences Individuelles*. Calmann-Levy, 1975.

[Ash98] K. Ashok. *Estimation and prediction of time dependent origin-destination flows*. PhD Thesis (Dir. M.BEN-AKIVA), MIT, 1998.

[Bal03] R. Baldacci, E. Hadjiconstantinou, A. Mingozzi. *An exact algorithm for the traveling Salesman problem with deliveries and collections*. Networks, vol. 42, pp. 26–41, 2003.

[Bal64] M.L. Balinski, R.E. Quandt. *On an Integer Program for a Delivery Problem*. Operations Research, vol. 12, pp. 300–304, 1964.

[Bal95] M. Ball, T. Magnanti, C. Monna, G. Nemhauser. *Network routing*. Handbook in Operation Research, Vol. 8, North Holland Amsterdam, 1995.

[Bar07] C. Barnhart, G. Laporte. *Transportation on Demand, chapitre in "Handbook in OR & MS"*. vol.14, pp. 429–466, 2007.

[Bea83] J.E. Beasley. *Route-first cluster-second methods for vehicle routing*. Omega, vol. 11, pp. 403–408, 1983.

[Bea94] J.C. Bean. *Genetic algorithms and random keys for the sequencing and optimization*. ORSA Journal on Computing, vol. 6, pp. 154–160, 1994.

[Ben01] A.W. Ben, N. Michel, E. Gourdin, B. Liau. *Routing strategies for IP networks.* Telektronik, vol. 2 (3), pp. 145–158, 2001.

[Bend01] F. Bendali, J. Mailfert, A. Quilliot. *Flots entiers et multiflots fractionnaires couplés par une contrainte de capacité*. Investigacion Operativa 9, 2001.

[Ben03] F. Bendali, J. Mailfert et A. Quilliot. *Méthodes de décomposition et d'agrégation pour le traitement de problèmes de multiflots*. Pesquisa Operacional, vol. 23 (3), pp. 475–498, 2003.

[Ben62] J.F. Benders. *Partionning procedures for solving mixed variable programming problems*. Numerische Mathematik, vol. 4, pp. 238–252, 1962.

[Ber04] J. Berger, M. Barkaoui. *A new hybrid genetic algorithm for the capacitated vehicle routing problem*. Journal of the Operational Research Society, vol. 54, pp. 1254–1262, 2004.

[Ber07] G. Berbeglia, J. Cordeau, I. Gribkovskaia, G. Laporte. *Static pickup and delivery problems: a classification scheme and survey*. TOP, vol. 15, pp. 45–47, 2007.

[Ber10] G. Berbeglia, J. Cordeau, G. Laporte: *Dynamic pickup and delivery problems*. European Journal of Operational Research, vol. 202(1), pp. 8-15, 2010.

[Ber57] C. Berge. *Théorie des Jeux à n personnes*. Gauthier-Villars, Paris, Memorial Sciences Maths, 1957.

[Ber96] D. Bertsimas and D. Simchi-levi. *A new generation of vehicle routing research: robust algorithms, addressing uncertainty*. Operations Research, vol. 44 (2), pp. 286-304, 1996.

[Bia00] L. Bianchi. *Notes on Dynamic Vehicle Routing - The state of the art*. Technical report INDSIA-05-01, 2000.

[Bia94] L. Bianco, A. Mingozzi, S. Ricciardelli, M. Spadoni. *Exact and heuristic procedures for the traveling salesman problem with precedence constraints, based on dynamic programming*. INFOR, vol. 32, pp. 19–31, 1994.

[Blu04] C. Blum. *Theoretical and Practical Aspects of Ant Colony Optimization*. Dissertations in Artificial Intelligence. Berlin: Akademische Verlagsgesellschaft Aka GmbH, 2004.

[Blu05] C. Blum. *Ant colony optimization: Introduction and recent trends*. Physics of Life Reviews, vol. 2 (4), pp. 353–373, 2005.

[Bod83] L.D. Bodin, B. L. Golden, A. A. Assad and M. O. Ball. *Routing and Scheduling of Vehicles and Crews: The State of Art*. Computers & Operations Research, vol. 10 (2), pp. 63–211, 1983.

[Bon63] O.N. Bondareva, *Applications of linear programming methods to the theory of cooperative games*. Problemy Kibernetica, vol. 10, pp. 119–139, 1963.

[Bor97] R. Borndörfer, M. Grötschel, F. Klostermeier, and C. Küttner. *Telebus Berlin: Vehicle scheduling in a dial-a-ride system*, (ZIB Report 97-23), 1997.

[Bra01] O. Braysy, M. Gendreau. *Route Construction and Local Search Algorithms for the Vehicle Routing Problem with Time Windows*. Internal Report STF42 A01024, SINTEF Applied Mathematics, Department of Optimisation, Norway, 2001.

[Bra06] J. Brandao. *A new tabu search algorithm for the vehicle routing problem with backhauls*. European Journal of Operational Research, vol. 173 (2), pp. 540–555, 2006.

[Bru87] H. Brundtland. *Report of the World Commission on Environment and Development*, General Assembly Resolution, No. 42/187, 1987.

[Bul98] B. Bullnheimer, R.F. Hartl, C. Strauss. *Applying the ant system to the vehicle routing problem*. In S. Voss, S. Martello, I.H. Osman, and C. Roucairol, editors, *Meta-Heuristics: Advances and Trends in Local Search Paradigms for Optimization*, Kluwer Academic Publishers, Boston, pp. 109–120, 1998.

[Bul99] B. Bullnheimer, R.F. Hartl, C. Strauss. *An improved ant system for the vehicle routing problem*. Annals of Operations Research, vol. 89, pp. 319–328, 1999.

[Car01] C. Carreto, B. Baker. *A GRASP interactive approach to the vehicle routing problem with backhauls*. In P. Hansen and C.C. Ribeiro, editors, *Essays and surveys on metaheuristics*. Kluwer Academic Publishers, Boston, 2001.

[Car02] M. Caramia, G.F. Italiano, G. Oriolo, A. Pacifici, A. Perugia. *Routing a fleet of vehicles for dynamic combined pickup and deliveries services*. In: Chamoni P, Leisten R, Martin A, Minnemann J, Stadtler H, editors, *Operations Research Proceedings 2001*. Springer, Berlin, Germany, pp. 3-8, 2002.

[Cas88] D. Casco, B. Golden, E. Wasil. *Vehicle routing with backhauls: Models, algorithms, and case studies*. In: B. Golden, A. Assad, editors, *Vehicle Routing: Methods and Studies*. North-Holland, Amsterdam, pp. 127–147, 1988.

[Cha02] P. Chardaire, A. Lisser. Simplex and interior point specialized algorithms for solving non oriented multicommodity flow problems. Operations Research, vol. 50 (2), pp. 260–276, 2002.

[Cha05] I. Charon, O. Hudry. *Méthodes arborescentes par séparation et evaluation*. PASCHOS, V.Th., editors, *Optimisation combinatoire : concepts fondamentaux*, chapitre 3, Hermes, Paris, pp. 63-91, 2005.

[Cha06] A. Chabrier. *Vehicle Routing Problem with elementary shortest path based column generation,* Computers & Operations Research, Part Special Issue: Constraint Programming, vol. 33 (10), pp. 2972–2990, 2006.

[Cha09] S. Chakravartya, A. Chikkatur, H. de Coninck, S. Pacala and R. Socolow. *Sharing global CO2 emission reductions among one billion high emitters*, PNAS, vol. 106, pp. 11884–11888, 2009.

[Cha99] P. Chardaire, J.L. Lutton, A. Suttter. *Upper and lower bounds for the two level simple plant location problem*. Annals of Operation Research, vol. 86, pp. 117–140, 1999.

[Char96] P. Chardaire. *Multihour design of computer backbone networks*. Telecommunication Systems, vol. 6, pp. 347–365, 1996.

[Chi94] J. Chifflet, P, Mahey, V. Reynier. *Proximal decomposition for multicommodity network problems*. Telecommunication Systems, vol. 3, pp. 1–10, 1994.

[Cho94] S. Chopra, M. Rao. *The Steiner tree problem I: formulations, composition and extension of facets*. Math Programming, vol. 64, pp. 209–229, 1994.

[Chr69] N. Christofides, S. Eilon. *An Algorithm for the Vehicle Dispatching Problem*. Operational Research Quarterly, vol. 20, pp. 309–318, 1969.

[Chr79] N. Christofides, A. Mingozzi, P. Toth. *The vehicle routing problem*. In Combinatorial Optimization, John Wiley, vol. 11, pp. 315–338, 1979.

[Chr81] N. Christofides, A. Mingozzi, and P. Toth. *State space relaxation procedures for the computation of bounds to routing Problems*. Networks, vol. 11, pp. 145-164, 1981.

[Chr85] N. Christofides. *Vehicle Routing*, in The Traveling Salesman Problem, Lawler EL, Lenstra JK, Rinnooy Kan AHG, Shmoys DB (eds) J. Wiley & Sons, pp. 431–448, 1985.

[Chv73] V. Chvatal. *Edmonds polytopes and a hierarchy of combinatorial problems. Discrete mathematics*. vol. 4(4), pp. 305–337, 1973.

[Cla64] G. Clarke and J.W. Wright. *Scheduling of vehicles from a central depot to a number of delivery points*. Operations Research, vol. 12, pp. 568-581, 1964.

[Col06] L. Coslovich, R. Pesenti, W. Ukovich. *A two-phase insertion technique of unexpected customers for a dynamic dial-a-ride problem*. European Journal of Operational Research, vol. 175, pp. 1605–1615, 2006.

[Col91] A. Colorni, M. Dorigo, and V. Maniezzo. *Distributed optimization by ant colonies*. In F. Varela and P. Bourgine, editors, *Proceedings of the European Conference on Artificial Life*, Elsevier, Amsterdam, pp. 134–142, 1991.

[Coo63] L. Cooper. *Location allocation problems*. Operation Research, vol. 11, pp. 331–343, 1963.

[Cor02] J.F. Cordeau, G. Desaulniers, J. Desrosiers, M.M. Solomon, F. Soumis. *VRP with time windows*. In: P. Toth, D. Vigo, editors, *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, PA, pp. 175–193, 2002.

[Cor04] J.F. Cordeau, G. Laporte. *Tabu search heuristics for the vehicle routing problem*. In: Rego, C., Alidaee, B., editors, *Metaheuristic Optimization via Memory and Evolution: Tabu Search and Scatter Search*. Kluwer Academic, Boston, pp. 145–163, 2004.

[Cor05] J.F. Cordeau, M. Gendreau, A. Hertz, G. Laporte, J.S. Sormany. *New heuristics for the vehicle routing problem*. In: A. Langevin, D. Riopel, Editors. *Logistics Systems: Design and Optimization*. Springer-Verlag, New York, pp. 279–297, 2005.

[Cor06] J.F. Cordeau. *A branch-and-cut algorithm for the dial-a-ride problem*. Operation Research, vol. 54, pp. 573–586, 2006.

[Cor07] J. Cordeau, G. Laporte, M. Savelsbergh, D. Vigo. *Chapter 6 Vehicle Routing*. In: Cynthia Barnhart and Gilbert Laporte, editors, *Handbooks in Operations Research and Management Science*, Elsevier, vol. 14, pp. 367–428, 2007.

[Cort10] C. Cortes, M. Matamala, C. Contardo. *The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method*. European Journal of Operational Research, vol. 200 (3), pp. 711–724, 2010.

[Cull81] F. Cullen, J. Jarvis, D. Ratliff. *Set partitioning based heuristics for interactive routing*. Networks, vol. 11, pp. 125–143, 1981.

[Dah94] G. Dahl, M. Stoer, *A 166olyhedral approach to multicommodity survivable network design*. Numerisch Mathematik, vol. 68, pp. 149–167, 1994.

[Dan54] G.B. Dantzig, D.R. Fulkerson, S.M. Johnson. *Solutions of a large-scale traveling-salesman problem.* Operations Research, vol. 2, pp. 363–410, 1954.

[Dan59] G.B. Dantzig and J.H. Ramser. *The Truck Dispatching Problem*. Management Science, vol. 6, pp. 80–91, 1959.

[Dan60] G.B. Dantzig, P. Wolfe. *Decomposition Principle for Linear Programs*. Operations Research, vol. 8, pp. 101–111, 1960.

[Das89] M.S. Daskin, M.D. Panayatopoulos, *A lagrangean relaxation approach to assigning aircraft to routes in hub and spoke networks*. Transportation Science, vol. 23 (2), pp. 91–99, 1989.

[Dav91] L. Davis. *The Genetic Algorithm HandBook*. Ed New York, 1991.

[Del06] M. Dellamico, G. Righini, M. Salani. *A branch-and-price approach to the vehicle routing problem with simultaneous distribution and collection*. Transportation Science, vol. 40 (2), pp. 235–247, 2006.

[Des02] G. Desaulniers, J. Desrosiers, A. Erdmann, M. Solomon, F. Soumis. *VRP with pickup and delivery*. In: Toth, P., Vigo, D. editors, *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, PA, pp. 225–242, 2002.

[Des05a] G. Desaulniers, J. Desrosiers, and M.M. Solomon (Eds.). *Column Generation, Springer*. 2005.

[Des05b] G. Desaulniers, J. Desrosiers, M. Solomon. *Column Generation*, No. 5. In: GERAD 25th Anniversary, Springer, 2005.

[Des08] S. Parragh , K. Doerner, R. Hartl. *A survey on pickup and delivery problems*. Journal für Betriebswirtschaft, Springer Berlin / Heidelberg, vol. 58 (2), pp.81–117, 2008.

[Des86] J. Desrosiers, Y. Dumas, F. Soumis. *A dynamic programming solution of the large-scale single-vehicle dial-a-ride problem with time windows*. American Journal of Mathematical and Management Sciences, vol. 6, pp. 301–325, 1986.

[Des88] M. Desrochers, F. Soumis. *A generalized permanent labeling algorithm for the shortest path problem with time windows*. INFOR, vol. 26, pp. 191–212, 1988.

[Des89] M. Desrochers, T.W. Verhoog. *A matching based savings algorithm for the vehicle routing problem*. Les Cahiers du GERAD G–89–04, HEC Montréal, 1989.

[Des91] M. Desrochers, G. Laporte. *Improvements and extensions to the Miller–Tucker–Zemlin subtour elimination constraints*, Operations Research Letters, vol. 10, pp. 27–36, 1991.

[Des95] J. Desrosiers, Y. Dumas, M. Solomon, and F. Soumis. *Chapter Time Constrained Routing and Scheduling*. Handbooks in Operations Research and Management Science, Elsevier Science, Amsterdam, vol. 8, pp. 35–140, 1995.

[Det02] J. Dethloff. *Relation between vehicle routing problems: An insertion heuristic for the vehicle routing problem with simultaneous delivery and pick-up applied to the vehicle routing problem with backhauls*. Journal of the Operational Research Society, vol. 53, pp. 115–118, 2002.

[Dor02] M. Dorigo, T. Stützle. *The ant colony optimization metaheuristic: Algorithms, applications and advances*. In F. Glover and G. Kochenberger, editors, *Handbook of Metaheuristics*, *International Series in Operations Research & Management Science*, Kluwer Academic Publishers, Norwell, MA, vol. 5, pp. 251–285, 2002.

[Dor04] M. Dorigo, T. Stützle. *Ant Colony Optimization*. MIT Press, Boston, MA, 2004.

[Dor05] M. Dorigo, C. Blum. *Ant colony optimization theory: A survey*. Theoretical Computer Science, vol. 344 (2–3), pp. 243–278, 2005.

[Dor99] M. Dorigo, G.D. Caro, L.M. Gambardella. *Ant algorithms for discrete optimization*. Artificial Life, vol. 5, pp.137–172, 1999.

[Dou06] A. Douaud. *Recommandations pour un développement durable des biocarburants en France*, rapport du groupe de travail sur les biocarburants, CIVEPE, 2006.

[Dre03] Z. Drezner. *A new genetic algorithm for the quadratic assignment problem*. INFORMS Journal on Computing, vol. 15, pp. 320–330, 2003.

[Dre95] S. Drezner. *Facility location: a survey of applications and methods*. N.Y Springer-Verlag Eds, 1995.

[Dro89] M. Dror, P. Trudeau. *Savings by split delivery routing*. Transportation Science, vol. 23, pp. 141–145, 1989.

[Dro90] M. Dror, P. Trudeau. *Split delivery routing*. Naval Research Logistics, vol. 37, pp. 383–402, 1990.

[Dro98] M. Dror, D. Fortin, C. Roucairol. *Redistribution of self-service electric cars: A case of pickup and delivery*. Technical Report W.P. 3543, INRIA-Rocquencourt, Rocquencourt, 1998.

[Duh10] C. Duhamel, P. Lacomme, C. Prins, C. Prodhon. *A GRASP×ELS approach for the capacitated location-routing problem*, Computers and Operations Research, vol. 37 (11), pp. 1912–1923, 2010.

[Duh12] C. Duhamel, P. Lacomme, C. Prodhon. *A hybrid evolutionary local search with depth first search split procedure for the heterogeonous vehicle routing problems*, Engineering Applications of Artificiial Intellignence, vol. 25 (2), pp. 345–358, 2012

[Dum91] Y. Dumas, J. Desrosiers, F. Soumis. *The pickup and delivery problem with time windows*. European Journal of Operational Research, vol. 54, pp. 7–22, 1991.

[Eco91] A. Economides, J. Silvester, *Multiobjective routing in integrated service networks: a game theory approach*. Proc IEEE INFOCOM, vol. 91, pp. 1220–1227, 1991.

[Eil71] S. Eilon, C.D.T. Watson-Gandy, and N. Christofides. *Distribution Management: Mathematical Modelling and Practical Analysis*. Griffin, London, 1971.

[Eks09] B. Eksioglu, A. V. Vural, A. Reisman. *The vehicle routing problem: A taxonomic review*. Computers & Industrial Engineering, In Press, Corrected Proof, Available online 25 May 2009.

[Erg03] Ö. Ergun, J.B. Orlin, A. Steele-Feldman. *Creating very large scale neighborhoods out of smaller ones by compounding moves: A study on the vehicle routing problem*. MIT Sloan working Paper 4393-02, Massachusetts Institute of Technology, Cambridge, MA, 2003.

[Fdi04a] S. Fdida et G. Hebuterne, éditeurs. *Méthodes exactes d'analyse de performance des réseaux*. Hermès-Science Publications, 2004.

[Fdi04b] S. Fdida et G. Hebuterne, éditeurs. *Méthodes heuristiques d'analyse de performance des réseaux*. Hermès-Science Publications, 2004.

[Feo89] T.A. Feo, M.G.C. Resende. *A probabilistic heuristic for a computationally difficult set covering problem*. O.R. Letters, vol. 8, pp. 67–71, 1989.

[Fis03] M. Fischetti, A. Lodi. *Local branching*. Math. Programming, vol. 98, pp. 23–47, 2003.

[Fis78] M.L. Fisher, R. Jaikumar. A decomposition algorithm for large-scale vehicle routing. Working paper 78-11-05, Department of Decision Sciences, University of Pennsylvania, 1978.

[Fis81] M.L. Fisher, R. Jaikumar. *A generalized assignment heuristic for vehicle routing*. Networks, vol. 11, pp. 109–124, 1981.

[Fis94] M.L. Fisher. Optimal solution of vehicle routing problems using minimum k-trees. Operations Research, vol. 37, pp. 319-328, 1994.

[Fis95] M.L. Fischer. *Vehicle Routing*. In Network Routing, Handbooks in Operations Research and Management Science, 8, Ball, M. O., T. L. Magnanti, C. L. Monma and G. L. Nemhauser, editor, Elsevier Science, Amsterdam, pp. 1–33, 1995.

[Fle10] G. Fleury, P. Lacomme. *Programmation linéaire avancée : Programmes Java pour Macintosh, Linux et Windows*. Ellipses, 2010.

[Fos76] B.A. Foster and D.M. Ryan. *An Integer Programming Approach to the Vehicle Scheduling Problem*. Operational Research Quarterly, vol. 27 (2), pp.367–384, 1976.

[Gas67] T. Gaskell. *Bases for the vehicle fleet scheduling*. Operations Research Quarterly, vol. 18, pp. 291–295, 1967.

[Gav78] Gavish, B. and S.C. Graves, *The 169raveling salesman problem and related problems*, Working Paper OR-078-78, Operations Research Center, MIT, Cambridge, MA, 1978.

[Gav92] B. Gavish, I. Neuman. *Routing in a network with unreliable components*. IEEE Trans on Communications, vol. 40, pp. 1248–1258, 1992.

[Gél95] S. Gélinas, M. Desrochers, J. Desrosiers, M.M. Solomon. *A new branching strategy for time constrained routing problems with application to backhauling*. Annals of Operations Research, vol. 61, pp. 91–109, 1995.

[Gen02] M. Gendreau, G. Laporte, J.Y. Potvin. *Metaheuristics for the capacitated VRP*. In: P. Toth, D. Vigo, Editors. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, pp. 129–154, 2002.

[Gen94] M. Gendreau, A. Hertz, G. Laporte. *A tabu search heuristic for the vehicle routing problem*. Management Science, vol. 40, pp. 1276–1290, 1994.

[Gen98] M. Gendreau, G. Laporte, J.Y. Potvin. *Metaheuristics for the vehicle routing problem*. Technical report, Les Cahiers du GERAD, Montreal, Canada, 1998.

[Gen99] M. Gendreau, G. Laporte, D. Vigo. *Heuristics for the traveling salesman problem with pickup and delivery*. Computers & Operations Research, vol. 26, pp. 699–714, 1999.

[Geo72] A. M. Geoffrion. *Generalized Benders Decomposition.* Journal of Optimization Theory and Applications, vol. 4(10), pp. 237–260, 1972.

[Geo74] A. M. Geoffrion et G. W. Graves. *Multicommodity Distribution System Design by Benders Decomposition*. Management science, vol. 20(5), pp. 822–844, 1974.

[Gha91] H. Ghaziri. *Solving routing problems by a self-organizing map*. In T. Kohonen, K. Makisara, O. Simula, and J. Kangas, editors, *Artificial Neural Networks*, North-Holland, Amsterdam, pp. 829–834, 1991.

[Gha93] H. Ghaziri. *Algorithmes connexionistes pour l'optimisation combinatoire*. Thèse de doctorat, École Polytechnique Fédérale de Lausanne, Switzerland, 1993.

[Ghi03] G. Ghiani, F. Guerriero, G. Laporte, R. Musmanno. *Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies.* European Journal of Operational Research, vol. 151, pp. 1–11, 2003

[Gil74] B.E. Gillett, L.R. Miller. *A heuristic algorithm for the vehicle-dispatch problem*. Operations Research, vol. 21, pp. 340–349, 1974.

[Glo03] F. Glover and G. Kochenberger. *Handbook of metaheuristics*. Kluwer, Boston, 2003.

[Glo86] F. Glover. *Future paths fur integer programming and links to artificial intelligence* Computers & Operations Research, vol. 5, pp. 533–549, 1986.

[Glo89] F. Glover. *Tabu Search : part I*. ORSA Journal of Computing, vol. 1(3), pp. 190–206, 1989.

[Glo90] F. Glover. *Tabu Search : part II*. ORSA Journal of Computing, vol. 2(1), pp. 4–32, 1990.

[Goe89] M. Goetschalckx, C. Jacobs-Blecha. *The vehicle routing problem with backhauls*. European Journal of Operational Research, vol. 42, pp. 39–51, 1989.

[Gol77] B.L. Golden, T.L. Magnanti and H.Q. Nguyen. *Implementing vehicle routing algorithms*. Networks, vol. 7, pp. 113-148, 1977.

[Gol85] D.E. Goldberg and R. Lingle. *Alleles, loci and the traveling salesman problem*. In J.J. Grefenstette, editor, *Proceedings of the First International Conference on Genetic Algorithms*, Lawrence Erlbaum, Hillsdale, NJ, pp. 154–159, 1985.

[Gol89] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley Publishing Company, Inc., 1989.

[Gol98] B.L. Golden, E.A. Wasil, J.P. Kelly, I.M. Chao. *Metaheuristics in vehicle routing*. In: Crainic, T.G., Laporte, G., editor, *Fleet Management and Logistics*. Kluwer Academic, Boston, pp. 33–56, 1998.

[Gold89] A.V. Golberg, R.E. Tarjan. *Finding minimum-cost circulations by canceling negative cycles*. Journal of the ACM (JACM), vol. 36(4), pp. 873–886, 1989.

[Gour04] E. Gourdin, B. Liau, A. Ouorou, D. Nace. *Optimisation des réseaux de télécommunications*. Bulletin de la Société Française de Recherche Opérationnelle et d'Aide à la Décision, vol. 12, pp. 8–12, 2004.

[Gri07] I. Gribkovskaia, Ø. Halskau, G. Laporte, M. Vlcek. *General solutions to the single vehicle routing problem with pickups and deliveries*. European Journal of Operational Research, vol. 180, pp. 568–584, 2007.

[Gro90] M. Grotchel, C.L. Monna. *Integer polyedra arising from certain network design problems with connectivity constraints*. SIAM Disc Math 3, pp. 502–524, 1990.

[Gui95] G. Owen. *Game Theory*. Academic Press, 3rd edition, 1995.

[Had95] E. Hadjiconstantinou, N. Christodes, A. Mingozzi. *A new exact algorithm for the vehicle routing problem based on q-Paths and k-Shortest paths relaxations*. Annals Operation Research, vol. 61, pp. 21–43, 1995.

[Hag05] A. Haghani, S. Jung. *A dynamic vehicle routing problem with time-dependent travel times*. Computers & Operations Research, vol. 32, pp. 2959-2986, 2005.

[Hai85] M. Haimovich, A.H.G. Rinnooy Kan. *Bounds and heuristics for capacitated routing problems*. Mathematics of Operations Research, vol. 10, pp. 527–542, 1985.

[Han01] P. Hansen, N. Mladenovic. *Variable neighborhood search: principles and applications*. Eur. J. Operation Research, vol. 130, pp. 449–467, 2001.

[Han03] P. Hansen, N. Mladenovic. *Variable neighbourhood search*. In: F. Glover and G. Kochenberger, Editors, *Handbook of Metaheuristics*, Kluwer, Dordrecht, pp. 146–184, 2003.

[Har88] N.P. Harilaos. *Vehicle Routing: Methods and Studies, chapter Dynamic Vehicle Routing Problems*. Elsevier Science Publishers B.V. (North Holland), pp. 223–248, 1988.

[Hel05] T. Helbling, N. Batini, and R. Cardarelli, *World Economic Outlook, April 2005: Globalization and external imbalances*, International Monetary Fund, 2005.

[Her03] H. Hernández-Pérez and J. J. Salazar-González. *The one-commodity pickup-and-delivery 171elecommu salesman problem*. In M. Jünger, G. Reinelt, and G. Rinaldi, editors, Combinatorial Optimization-Eureka, You Shrink!, Lecture Notes in Computer Science, Springer, vol. 2570, pp. 89–104, 2003.

[Her04a] H. Hernández-Pérez and J. J. Salazar-González. *A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery*. Discrete Applied Mathematics, vol. 145, pp. 126–139, 2004.

[Her04b] H. Hernández-Pérez and J. J. Salazar-González. *Heuristics for the one-commodity pickup-and-delivery traveling salesman problem*. Transportation Science, vol. 38(2), pp. 245–255, 2004.

[Her07] H. Hernández-Pérez and J.J. Salazar-González. *The one-commodity pickup and delivery traveling salesman problem: Inequalities and algorithms.Networks*, vol. 50(4), pp. 258–272, 2007.

[Her08] H. Hernández-Pérez, I. Rodríguez-Martín and J. J. Salazar-González. *A hybrid GRASP/VND heuristic for the One-Commodity Pickup-and-Delivery Traveling Salesman Problem*. Computers & Operations Research, vol. 36(5), pp. 1639–1645, 2008.

[Hil01] F. Hillier and G. Liberman, *Introduction to operations research*, 7th ed., McGraw-Hill, 2001.

[Hil95] S.P. Hill. *Branch and cut methods for the symmetric capacitated vehicle routing problem.* School of Mathematics and Statistics, Curtin University of Technology, 1995.

[Hir05] R.L. Hirsch, R.H. Bezdek, R.M. Wendling, *Peaking of World Oil Production: Impacts, Mitigation and Risk Management*, DOE NETL. 2005.

[Hol75] J.H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, Michigan; re-issued by MIT Press (1992), 1975.

[Hos10a] M. I. Hosny and C. L. Mumford. *Solving the one-commodity pickup and delivery problem using an adaptive hybrid VNS/SA approach*. In Proceedings of the 11th International Conference on Parallel Problem Solving From Nature (PPSN2010), LNCS, Springer-Verlag, pp. 189–198, 2010.

[Hos10b] M. I. Hosny. *Investigating Heuristic and Meta-heuristic Algorithms for Solving Pickup and Delivery Problems*. PhD Thesis, Cardiff University, 2010.

[Ioa95] I. Ioachim, J. Desrosiers, Y. Dumas, M.M. Solomon and D. Villeneuve. *A request clustering algorithm for door-to-door handicapped transportation*, Transportation Science, vol. 29, pp. 63-78, 1995.

[Jai97] Jaillet P., Song G., Yu G., *Airline network design and hub location problems*. Location Science, vol. 4 (3), pp. 195–212, 1997.

[Jin08] M. Jin, K. Liu, B. Eksioglu. *A column generation approach for the split delivery vehicle routing problem*. Operations Research Letters, vol. 36 (2), pp. 265–270, 2008.

[Jou08] L. Jourdan, M. Basseur, E.-G. Talbi. *Hybridizing exact methods and metaheuristics: A taxonomy*. European Journal of Operational Research, vol. 199 (3), pp. 620–629, 2009.

[Kal85] B. Kalantari, A.V. Hill, S.R. Arora. *An algorithm for the traveling salesman problem with pickup and delivery customers*. European Journal of Operational Research, vol. 22, pp. 377–386,1985.

[Kaw98] H. Kawamura, M. Yamamoto, T. Mitamura, K. Suzuki, and A. Ohuchi. *Cooperative search on pheromone communication for vehicle routing problems*. IEEE Transactions on Fundamentals, E81-A, pp. 1089–1096, 1998.

[Khu72] B.M. Khumalala. *Warehouse location problem efficient branch and bound algorithm*. Management Sciences B, vol. 18, pp. 718–731, 1972.

[Kil98] P. Kilby, P. Prosser and P. Shaw. *Dynamic VRPs: A study of scenarios*. Report APES-06-1998, 1998.

[Kin97] G.A.P. Kindervater, M.W.P Savelsbergh. *Vehicle routing: Handling edge exchanges*. In: Aarts, E.H.L., Lenstra, J.K. editors, *Local Search in Combinatorial Optimization*. Wiley, Chichester, pp. 337–360, 1997.

[Kir83] S. Kirkpatrick, C. D. Gelatt, M. P. Vecchi. *Optimization by Simulated Annealing*. Science, vol. 220(4598), pp. 671–680, 1983.

[Kle75] L. Kleinrock. *Queuing Systems, vol. 1: Theory*. N.T WILEY, Ed, 1975.

[Kos05] M. Koskas. *Algorithmes de min-max et jeux*. PASCHOS, V.Th., Ed., Optimisation combinatoire Problèmes paradigmatiques et applications, chapitre 10, Hermes, Paris, 2005.

[Lap02] G. Laporte, F. Semet. *Classical heuristics for the capacitated VRP*. In: Toth, P., Vigo, D., editors, *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, pp. 109–128, 2002.

[Lap85] G. Laporte, Y. Nobert, M. Desrochers. *Optimal routing under capacity and distance restrictions*. Operation Research, vol. 33, pp. 1050–1073, 1985.

[Lap86] G. Laporte, H. Mercure, Y. Nobert. *An exact algorithm for the asymmetrical capacitated vehicle routing problem*. Networks, vol. 16, pp. 33–46, 1986.

[Lap87] G. Laporte, Y. Nobert. *Exact algorithms for the vehicle routing problem*. Ann. Discrete Math, vol. 31, pp. 147–184, 1987.

[Lap92] G. Laporte. *The vehicle routing problem: An overview of exact and approximate algorithms*. European Journal of Operational Research, vol. 59 (3), pp. 345–358, 1992.

[Lap95] G. Laporte, I. H. Osman. *Routing problems: A bibliography*. Annals of Operations Research, vol. 61, pp. 227–262, 1995.

[Lar01] A. Larsen. *The dynamic vehicle routing problem*. Ph.D. Thesis, Institute of Mathematical Modeling, IMMPHD-2000-73, Technical University of Denmark, 2001.

[Las70] L.S. Lasdon. *Optimization Theory for Large Systems*. MacMillan, New-York, 1970.

[Las02] L.S. Lasdon. *Optimization Theory for Large Systems* (reprint of the 1970 Macmillan ed.). Mineola, New York: Dover Publications, 2002.

[Len81] J. Lenstra and A. Rinnoy Kan. *Complexity of Vehicle Routing and Scheduling Problems*. Networks, vol. 11, pp. 221–227, 1981.

[Li05] F. Li, B.L. Golden, E.A. Wasil. *Very large-scale vehicle routing: New test problems, algorithms and results*. Computers & Operations Research, vol. 32, pp. 1165–1179, 2005.

[Lin09] S.W. Lin, Z.J. Lee, K.C. Ying, C.Y. Lee. *Applying hybrid meta-heuristics for capacitated vehicle routing problem*. Expert Systems with Applications, vol. 36 (2), pp. 1505–1512, 2009.

[Lin65] S. Lin. *Computer solutions of the traveling salesman problem*. Bell Systems Computer Journal, vol. 44, pp. 2245–2269, 1965.

[Lit63] J.D.C. Little, K.G. Murty, D.W. Sweeney, and C. Karel. *An algorithm for the traveling salesman problem*. Operations Research, vol. 11, pp. 972-989, 1963.

[Liu05] K. Liu. *A Study on the split delivery vehicle routing problem*. PhD thesis, Mississippi State University, 2005.

[Lou03] H.R. Lourenço, O. Martin, and T. Stützle. *Iterated local search*. In *Handbook of Metaheuristics*, edited by F. Glover and G. Kichenberger, Kluwer, pp. 321–353, 2003.

[Lun96] K. Lund, O.B.G. Madsen, M.M. Solomon. *Vehicle routing problems with varying degrees of dynamism*. Technical Report, Institute of Mathematical Modelling, Technical University of Denmark, 1996.

[Lys04] J. Lysgaard, A.N. Letchford, R.W. Eglese. *A new branch-and-cut algorithm for the capacitated vehicle routing problem*. Mathematical Programming, vol. 100, pp. 423–45, 2004.

[Mad95] O.B.G. Madsen, H.F. Ravn, J.M. Rygaard. *A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives*. Annals of Operations Research, vol. 60, pp. 193–208, 1995.

[Mah05] A.R. Mahjoub, *Méthodes polyédrales*. PASCHOS, V.Th., Ed., Optimisation combinatoire : concepts fondamentaux, chapitre 8, Hermes, Paris, 2005.

[Mah98] P. Mahey, A. Ouourou, L. Leblanc, J. Chifflet. *A new proximal decomposition algorithm for routing in telecommunication networks*. Networks, vol. 31, pp. 227–238, 1998.

[Mar08] G. Martinovic, I. Aleksi, A. Baumgartner. *Single-commodity vehicle routing problem with pickup and delivery service*. Mathematical Problems in Engineering 2008, pp. 1–17, 2008.

[Mat91] Y. Matsuyama. *Self-organization via competition, cooperation and categorization applied to extended vehicle routing problems*. In Proceedings of the International Joint Conference on Neural Networks, Seattle, WA, pp. 385–390, 1991.

[Mcs03] *Mobility CarSharing Suisse*, communiqué de presse, 2003.

[Mel07] P. Melquiot, *Fret SNCF crée un système de Haut Débit Ferroviaire pour faire du fret ferroviaire un acteur majeur du développement durable en France et en Europe*, communiqué de presse, 2007.

[Mes05] D. Mester, O. Bräysy. *Active guided evolution strategies for large scale vehicle routing problem with time windows*. Computers & Operations Research, vol. 32, pp. 1593–1614, 2005.

[Mil60] C.E. Millier, A.W. Tucker et R.A. Zemlin. *Integer programming formulations and traveling salesman problems*. Journal of the Association for Computing Machinery, vol. 4, pp.326–329, 1960.

[Min75] M. Minoux. *Plus courts chemins avec contraintes.* Annales des Télécommunications, vol. 30, pp. 383–394, 1975

[Min81] M. Minoux. *Optimum synthesis of a network with non simultaneous multicommodity flow requirements.* In HANSEN, pp. 269–277, 1981.

[Min89] H. Min. *The multiple vehicle routing problem with simultaneous delivery and pickup points*. Transport Research A, vol. 23, pp. 377–386, 1989.

[Min94] A. Mingozzi, N. Christofides, E. Hadjiconstantinou. *An exact algorithm for the vehicle routing problem based on the set partitioning formulation*. Technical report, Department of Mathematics, University of Bologna, Italy, 1994.

[Mit01] S. Mitrovic-Minic. *The Dynamic Pickup and Delivery Problem with Time Windows*. Simon Fraser University, 2001.

[Mit05] S. Mitra. *An algorithm for the generalized vehicle routing problem with backhauling*. Asia-Pacific Journal of Operational Research, vol. 22, pp. 153–169, 2005.

[Mit08] S. Mitra. *A parallel clustering technique for the vehicle routing problem with split deliveries and pickups*. Journal of the Operational Research Society, vol. 59, pp. 1532–1546, 2008.

[Mla97] N. Mladenovic, P. Hansen. *Variable neighborhood search*. Computers and Operations Research, vol. 24, pp. 1097–1100, 1997.

[Mol76] R.H. Mole, S.R. Jameson. *A sequential route-building algorithm employing a generalized savings criterion*. Operational Research Quarterly, vol. 27, pp. 503–511, 1976.

[Nad00] D. Naddef, G. Rinaldi. *Branch-and-cut algorithms for the capacitated VRP*. In: P. Toth, D. Vigo, Editors. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia, pp. 53–84, 2002.

[Nag05] G. Nagy, S. Salhi. *Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries*. European Journal of Operational Research, vol. 162, pp. 126–141, 2005.

[Nas51] J. Nash, *Non cooperative games*. Annals of Mathematics, vol. 54, pp. 286–295, 1951.

[Nel85] M.D. Nelson, K.E. Nygard, J.H. Griffin and W.E. Shreve. *Implementation techniques for the vehicle routing problem*. Computers & Operations Research, vol. 12, pp. 273-283, 1985.

[Nov09] C. Novoa et R. Storer. *An approximate dynamic programming approach for the vehicle routing problem with stochastic demands*. European Journal of Operational Research, vol. 196 (2), pp. 509-515, 2009.

[Now05] M.A. NoWak. *The Pickup and Delivery Problem with Split Loads*. Ph.D. Thesis, Industrial and Systems Engineering Georgia Institute of Technology, 2005.

[Now08] M.A. Nowak, O. Ergun, C.C. III White. *Pickup and delivery with split loads*. Transportation Science, vol. 42, pp. 32–43, 2008.

[Now09] M.A. Nowak, O. Ergun, C.C. III White. *An empirical study on the benefit of split loads with the pickup and delivery problem*. European Journal of Operational Research, vol. 198, pp. 734–740, 2009.

[Oli87] I.M. Oliver, D.J. Smith, and J.R.C. Holland. *A study of permutation crossover operators on the traveling salesman problem*. In J.J. Grefenstette, editor, *Proceedings of the Second International Conference on Genetic Algorithms*, Lawrence Erlbaum, Hillsdale, NJ, pp. 224–230, 1987.

[Onc09] T. Oncan, I. K. Altinel, G. Laporte, *A comparative analysis of several asymmetric traveling salesman problem formulations*, Computers & Operations Research, vol. 36 (3), pp. 637–654, 2009.

[Opp08] J. Oppen, A. Lokketangen. *A tabu search approach for the livestock collection problem*. Computers & Operations Research, Part Special Issue: Search-based Software Engineering, vol. 35 (10), pp. 3213–322, 2008.

[Or76] I. Or. *Traveling salesman-type combinatorial problems and their relation to the logistics of blood banking*. PhD thesis, Department of Industrial Engineering and Management Science, Northwestern University, Evanston, 1976.

[Orl76] C. Orloff. *Route-constrained fleet scheduling*, Transportation Science, vol. 10, pp. 149-168, 1976.

[Orm07] A.J. Orman, H.P. Williams. *A survey of different integer programming formulations of the 174elecommu salesman problem*. In: Kontoghiorghes EJ, Gatu C, editors. Optimisation, economics and financial analysis. Advances in computational management science, Heidelberg: Springer, vol. 9, pp. 93–106, 2007.

[Osm93] I.H. Osman. *Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem*. Annals of Operations Research, vol. 41, pp. 421–451, 1993.

[Osm96] I.H. Osman, J.P. Kelly. *Meta-heuristics: An overview*. In I. H. Osman & J. P. Kelly (Eds.), *Meta-heuristics: Theory and applications*, Boston: Kluwer Academic Publishers, pp. 1–21, 1996.

[Ouo00] A. Ouorou, P. Mahey et J. Vial. *A survey of algorithms for convex multicommodity flow problems*. Management science, vol. 46(1), pp. 126–147, 2000.

[Pae88] H. Paessens. *The savings algorithm for the vehicle routing problem*. European Journal of Operational Research, vol. 34, pp. 336-344, 1988.

[Par08] S.N. Parragh, K. Doerner, R.F. Hartl. *A survey on pickup and delivery models Part I: transportation between customers and depot*. Working paper, Chair of Production and Operations Management, University of Vienna, 2008.

[PK98] *Protocole de Kyoto à la convention-cadre des Nations Unies sur les changements climatiques,* Convention-Cadre des Nations Unies sur les Changements Climatiques, 1998.

[Pot02] J.Y. Potvin, K.A. Smith. *Artifical neural networks for combinatorial optimization*. In F. Glover and G. Kochenberger editors, Handbook of Metaheuristics, International Series in Operations Research & Management Science, Kluwer Academic Publishers, Norwell, MA, vol. 57, pp. 251–285, 2002.

[Pot03] J.Y. Potvin, K.A. Smith. *Artificial neural networks for combinatorial optimization*. In: Glover, F., Kochenberger, G., editors, *Handbook of metaheuristics*, Kluwer Academic Publishers, Boston, pp. 429–455, 2003.

[Pot95] J.Y. Potvin, J.M. Rousseau. *An exchange heuristic for routing problems with time windows*. Journal of the Operational Research Society, vol. 46, pp. 1433–1446, 1995.

[Pot96] J.Y. Potvin. *Genetic algorithms for the traveling salesman problem*. Annals of Operations Research, vol. 63, pp.339–370, 1996.

[Pow07] W.B. Powell, B. Bouzanene-Ayari, H.P. Simpo. *Dynamic models for freight transportation*. In: C. Barnhart, G. Laporte, Editors, *Handbooks in Operations Research and Management Science: Transportation*, vol. 14. North Holland, pp.285–365, 2007.

[Pow87] W.B. Powell. *An operational planning model for the dynamic vehicle allocation problem with uncertain demands*. Transportation Research, Part B, vol. 21, pp. 217–232, 1987.

[Prin04] C. Prins. *A simple and effective evolutionary algorithm for the vehicle routing problem*, Computers & Operations Research, vol. 31, pp. 1985–2002, 2004.

[Prin09] C. Prins. *A GRASP×Evolutionary Local Search Hybrid for the Vehicle Routing Problem*, in: F.B. Pereira and J. Tavares (Ed.), Bio-inspired Algorithms for the Vehicle Routing Problem, Studies in Computational Intelligence, publisher Springer, Berlin, vol. 161, pp. 35–53, 2009.

[Psa80] H. N. Psaraftis. *A Dynamic Programming Solution to the single Many-to-many Immediate Request Dial-a-Ride Problem*. Transporta-tion Science, vol. 14, pp. 130–154, 1980.

[Psa83a] H. N. Psaraftis. *Analysis of an O(n2) heuristic for the single vehicle many-to-many Euclidean dial-a-ride problem*. Transportation Research B, vol. 17, pp. 133–145, 1983.

[Psa83b] H. N. Psaraftis. *An exact algorithm for the single-vehicle, many-to-many dial-a-ride problem with time windows*. Transportation Science, vol. 17, pp. 351–357, 1983.

[Psa88] H. N. Psaraftis. *Dynamic vehicle routing problem*. In: Golden BL, Assad AA, editors, *Vehicle routing: methods and studies*. Amsterdam: North-Holland, pp. 293–318, 1988.

[Psa95] H. N. Psaraftis. *Dynamic vehicle routing: Status and prospects*. Annals of Operation Research, vol. 61, pp. 143–164, 1995.

[Quil05] A. Quilliot, F. Bendali, J. Mailfert. *Flots entiers et multiflots fractionnaires couplés par une contrainte de capacité*. RAIRO Operations Research, vol 39 (3), pp. 185–224, 2005.

[Ree03] C. Reeves. *Genetic algorithms*. In: F. Glover and G. Kochenberger, Editors, Handbook of Metaheuristics, Kluwer, Dordrecht, pp. 55–82, 2003.

[Rei04] M. Reimann, K. Doerner, R.F. Hartl. *D-ants: Savings based ants divide and conquer for the vehicle routing problem*. Computers & Operations Research, vol. 31, pp. 563–591, 2004.

[Reg96] C. Rego and C. Roucairol. *A parallel tabu search algorithm using ejection chains for the vehicle routing problem*. In : Meta-Heuristics. Springer US, pp. 661–675, 1996.

[Rek06] B. Rekiek, A. Delchambre, H.A. Saleh. *Handicapped person transportation: An application of the grouping genetic algorithm*. Engineering Applications of Artificial Intelligence, vol. 19, pp. 511–520, 2006.

[Rena96] J. Renaud, F.F. Boctor, G. Laporte. *An improved petal heuristic for the vehicle routing problem*. Journal of the Operational Research Society, vol. 47, pp. 329–336, 1996.

[Roc95] Y. Rochat, E.D. Taillard. *Probabilistic diversification and intensification in local search for vehicle routing*. Journal of Heuristics, vol. 1, pp. 147–167, 1995.

[Rod08] C.J. Rodhes, *The oil question: nature and prognosis*, Science Progress, vol. 91 (4), pp. 317–375, 2008.

[Rop06] S. Ropke, D. Pisinger. *A unified heuristic for a large class of vehicle routing problems with backhauls*, European Journal of Operational Research, vol. 171, pp. 750–775, 2006.

[Rop09] S. Ropke, J. Cordeau. *Branch and Cut and Price for the Pickup and Delivery Problem with Time Windows*. Transportation Science, vol. 43(3), pp. 267-286, 2009.

[Reb00] R. Rebai, *Optimisation de réseaux de 176elecommunications avec sécurisation*. Thèse PARIS-DAUPHIINE, 2000.

[Rec73] I. Rechenberg. *Evolutions strategie*. Fromman-Holzboog, Stuttgart, Germany, 1973.

[Rul97] K.S. Ruland, E.Y. Rodin. *The pickup and delivery problem: Faces and branch-and-cut algorithm*. Computers & Mathematics with Applications, vol. 33 (12), pp. 1–13, June 1997.

[Rya93] D.M. Ryan, C. Hjorring, F. Glover. *Extensions of the petal method for vehicle routing*. Journal of Operational Research Society, vol. 44, pp. 289–296, 1993.

[Sal99] S. Salhi, G. Nagy. *A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling*. Journal of the Operational Research Society, vol. 50, pp. 1034–1042, 1999.

[Sav95] M. Savelsbergh, M. Sol. *The general pickup and delivery problem*. Transportation Science vol. 29 (1), pp.17–29, 1995.

[Sav98] M. Savelsbergh, M. Sol. *DRIVE: Dynamic routing of independent vehicles*. Operations research, vol. 46, pp. 474–490, 1998.

[Sch77] M. Schwartz. *Computer communication network design and analysis*. Prentice Hall, Englewood Cliff. NJ, 1977.

[Sch06] S. Scheuerer. *A tabu search heuristic for the truck and trailer routing problem*. Computers & Operations Research, Part Special Issue: Optimization Days 2003, vol. 33 (4), p. 894–909, 2006.

[Sch95] L.J. Schmitt. *An evaluation of a genetic algorithmic approach to the vehicle routing problem*. Working paper, Department of Information Technology Management, Christian Brothers University, Memphis, TN, 1995.

[Sec00] N. Secomandi. *Comparing neuro-dynamic programming algorithms for the vehicle routing problem with stochastic demands*. Computers & Operations Research, vol. 27, pp. 1201–1225, 2000.

[She02] H.D. Sherali, P.J. Driscoll. *On tightening the relaxations of Miller–Tucker–Zemlin formulations for asymmetric traveling salesman problems,* Operations Research, vol. 50, pp. 656–69, 2002.

[Soc08] K. Socha, M. Dorigo. *Ant colony optimization for continuous domains*. European Journal of Operational Research, vol. 185 (3), pp.1155–1173, 2008.

[Spi85] J.E. Spingarn, *Applications of the method of partial inverse to convex programming decomposition*. Mathematical Programming, vol. 32, pp. 199-223, 1985.

[Swi99] M.R. Swihart, J.D. Papastavrou. *A stochastic and dynamic model for the single-vehicle pick-up and delivery problem*. European Journal of Operational Research, vol. 114, pp. 447–464, 1999.

[Tai93] É.D Taillard. *Parallel iterative search methods for vehicle routing problems*. Networks, vol. 23, pp. 661–673, 1993.

[Tar02] C.D. Tarantilis, C.T. Kiranoudis. *Bone route: Adaptive memory method for effective fleet management*. Annals of Operations Research, vol. 115, pp. 227–241, 2002.

[Teo00] D. Teodorovic, G. Radivojevic. *A fuzzy logic approach to dynamic dial-a-ride problem*. Fuzzy Sets and Systems, vol. 116, pp. 23–33, 2000.

[Tha95] S. Thangiah. *Vehicle routing with time windows using genetic algorithms*. In Application Handbook of Genetic Algorithms: New Frontiers, Vol. 2, pp. 253-277, CRC Press, Boca Raton, 1995.

[Tim04] C.D. Timon, Y.L. Eldon, C. Defrose. *Dynamic vehicle routing for online B2C delivery*. Omega, ISSN 0305-0483, DOI: 10.1016/j.omega.2004.03.005, vol. 33 (1), pp. 33–45, 2005

[Tor72] C. Toregas and C. ReVelle. *Location under time or distance constraints*, Papers of the Regional Science Association, vol. 28, pp. 133-143, 1972.

[Tot02] P. Toth and D. Vigo. *The Vehicle Routing Problem*. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia 2002.

[Tot02b] P. Toth, D. Vigo. *Models, relaxations and exact approaches for the capacitated vehicle routing problem*. Discrete Applied Mathematics, vol. 123 (1-3), pp. 487–512, 2002.

[Tot03] P. Toth, D. Vigo. *The granular tabu search and its application to the vehicle routing problem*. INFORMS Journal on Computing, vol. 15, pp. 333–346, 2003.

[Tot96] P. Toth, D. Vigo. *Fast local search algorithms for the handicapped persons transportation problem*. In: I.H. Osman, J.P. Kelly, editors. *Metaheuristics: Theory and Applications*. Kluwer, Boston, MA, pp. 677–690, 1996.

[Tot98] P. Toth, D. Vigo. *Exact solution of the Vehicle Routing Problem*, Dans *Fleet Management and Logistics*, T.G. Crainic & G. Laporte, Kluwer, editors, Boston, pp. 1–31, 1998.

[Tot99] P. Toth, D. Vigo. *A heuristic algorithm for the symmetric and asymmetric vehicle routing problem with backhauls*. European Journal of Operational Research, vol. 113, pp. 528–543, 1999.

[Tou07] N. Touati, L. Létocart et A. Nagih. *Méthodes de décomposition pour l'optimisation discrète*. Rapport scientifique LIPN, 2007.

[Vol83] A. Volgenant, R. Jonker. *The symmetric traveling salesman problem and edge exchange in minimal 1-trees*. European Journal of Operational Research, vol. 12, pp. 395–403, 1983.

[Vou97] C. Voudouris. *Guided local search for combinatorial problems. Dissertation, University of Essex*, United Kingdom, 1997.

[Wai08] H.A. Waisanen, D. Shah, M.A. Dahleh. *A dynamic pickup and delivery problem in mobile networks under information constraints*. IEEE Transactions on Automatic Control, vol. 53, pp. 1419–1433, 2008.

[War94] P. Wark, J. Holt. *A repeated matching heuristic for the vehicle routing problem*. Journal of Operational Research Society, vol. 45, pp. 1156–1167, 1994.

[Weo08] *World Energy Outlook 2008*, International Energy Agency, 2008.

[Wil89] J.A.G. Willard. *Vehicle routing using r-optimal tabu search.MSc dissertation*, The Management School, Imperial College, London, 1989.

[Wol07] S. Wolf and P. Merz. *Evolutionary local search for the super-peer selection problem and the p-hub median problem.* In T. Bartz-Beielstein et al., editor, Hybrid Metaheuristics, number 4771 in Lecture Notes in Computer Science, pp. 1–15. Springer-Verlag, Berlin / Heidelberg, 2007.

[Wol98] L.A. Wolsey, *Integer Programming*. John Wiley & Sons, Inc. 1998.

[Wre71] A. Wren. *Computers in Transport Planning and Operation*. Ian Allan, London, 1971.

[Wre72] A. Wren. Holliday. *Computer scheduling of vehicles from one or more depots to a number of delivery points*. Operational Research Quarterly, vol. 23, pp. 333–344, 1972.

[Xu03] H. Xu, Z.L. Chen, S. Rajagopal, S. Arunapuram. *Solving a practical pickup and delivery problem*. Transportation Science, vol. 37 (3), pp. 347–364, 2003.

[Xu96] J. Xu, J.P. Kelly. *A network flow-based tabu search heuristic for the vehicle routing problem*. Transportation Science, vol. 30, pp. 379–393, 1996.

[Yag73] B. Yaged. *Minimum cost routing for dynamic network models*. Networks, vol. 3, pp. 315–331, 1973.

[Yan04] J. Yang, P. Jaillet, H.S. Mahmassani. *Real-time multivehicle truckload pickup and delivery problems*. Transportation Science, vol. 38, pp. 135–148, 2004.

[Yan87] C. Yano, T. Chan, L. Richter, T. Cutler, K. Murty and D. McGettigan. *Vehicle routing at quality stores. Interfaces*, vol. 17, pp. 52-63, 1987.

[Yan98] J. Yang, P. Jaillet, H.S. Mahmassani. *On-line algorithms for truck fleet assignment and scheduling under real-time information*. Transportation Research, Record 1667, pp. 107–113, 1998.

[Yon05] L. Yon. *Modèles et outils pour la conception stratégique de réseaux de transports publics*. Thèse de Doctorat, LIMOS, Université Blaise Pascal, Clermont-Ferrand, 2005.

[Zha09] F. Zhao, S. Li, J. Sun, D. Mei. *Genetic algorithm for the one-commodity pickup and-delivery traveling salesman problem*. Computers & Industrial Engineering, vol. 56 (4), pp. 1642–1648, 2009.