Thèse

Présentée par

Wahiba RAMDANE-CHERIF

Pour l'obtention du

Grade de Docteur de L'UTT

Spécialité : Gestion de la production

Problèmes d'optimisation en tournées sur arcs

Soutenue le 12 décembre 2002 devant le jury composé de :

Président:	
Alexandre DOLGUI	Professeur des Universités, Université de Technologie de Troyes
Rapporteurs :	
Enrique BENAVENT	Professeur, Université de Valencia (Espagne)
Pierre DEJAX	Professeur, Ecole des Mines de Nantes
Examinateurs :	
Philippe LACOMME	Maître de Conférences, Université Blaise Pascal (Directeur de thèse)
Christian PRINS	Professeur des Universités, Université de Technologie de Troyes (Directeur de thèse)
Marc SEVAUX	Maître de Conférences, Université de Valenciennes

Table des matières

Glossaire I

Liste des figures V

Liste des tables VII

Introduction 1

CHAPITRE I Etat de l'art en tournées sur arcs

1.1 Introduction 5

1.2 Historique et points d'entrée pour les tournées sur arcs 5

1.3 Transformation des problèmes de tournées sur arcs en problèmes de tournées sur nœuds . 6

1.4 Notations pour les programmes linéaires du chapitre 7

1.5 Le problème du postier chinois 8

1.5.1 Le problème du postier chinois non orienté (UCPP) 8

1.5.2 Le problème du postier chinois orienté (DCPP) 9

1.5.3 Le problème du postier chinois mixte (MCPP) 9

1.6 Le problème du postier rural (RPP) 10

1.6.1 Le problème du postier rural non orienté (URPP) 11

1.6.2 Le problème du postier rural orienté (DRPP) 12

1.6.3 Le problème du postier rural mixte (MRPP) 12

1.7 Autres problèmes sur arcs sans capacité 13

1.7.1 Le problème du postier venteux (WPP) 13

1.7.2 Le problème du postier rural venteux (WRPP) 14

1.7.3 Le problème du postier hiérarchique (HPP) 14

1.7.4 Le problème de la grue (Stacker Crane Problem ou SCP) 14

1.7.5 Autres variantes plus exotiques 15
1.8 Le problème de tournées sur arcs avec contrainte de capacité 15
1.8.1 Présentation du problème 15
1.8.2 Formulations par programmes linéaires 16
1.8.2.1 Formulation de base 16
1.8.2.2 La formulation sparse 17
1.8.2.3 La formulation supersparse 17
1.8.3 Méthodes optimales 18
1.8.4 Bornes inférieures 18
1.8.4.1 Matching Lower Bound (MLB) 18
1.8.4.2 Node Scanning Lower Bound (NSLB) 19
1.8.4.3 Matching Path Lower Bound (MPLB) 19
1.8.4.4 Successive Cut Lower Bound (SCLB) 20
1.8.4.5 Les bornes : LB1, LB2, LB3 et LB4 20
1.8.4.6 Borne basée sur la méthode de coupes 20
1.8.4.7 Hierarchical Relaxations Lower Bound (HRLB) 21
1.8.5 Heuristiques constructives 21
1.8.5.1 Construct-Strike 21
1.8.5.2 Path-Scanning 22
1.8.5.3 Augment-Merge 22
1.8.5.4 Parallel-Insert 23
1.8.5.5 Augment-Insert 23
1.8.5.6 Méthodes de type Cluster-First, Route-Second 23
1.8.5.7 Méthodes de type Route-First, Cluster-Second 23
1.8.6 Métaheuristiques 24
1.8.6.1 La méthode tabou 24
1.8.6.2 Le recuit simulé 26
1.8.6.3 Recherche à voisinage variable (*VNS* pour Variable Neighbourhood Search) .. 26
1.9 Applications et extensions du CARP 27
1.10 Conclusion 29

CHAPITRE II Problèmes étudiés et notations

2.1 Introduction 31
2.2 L'application de référence : la collecte des déchets urbains 31

2.3 Extensions étudiées dans la thèse 34
2.4 Hypothèses de travail 36
2.5 Notations et structure de données 36
2.5.1 Conventions algorithmiques 36
2.5.2 Données 37
2.5.2.1 Réseau mixte (hors données liées à l'activité de collecte) 37
2.5.2.2 Le problème du distancier 41
2.5.2.3 Arcs à traiter 42
2.5.2.4 Dépôts et lieux de vidage 43
2.5.2.5 Véhicules 44
2.5.2.6 Planification 44
2.5.3 Listes 44
2.5.4 Solutions et tournées 45
2.6 Liste récapitulative des notations 46
2.6.1 Réseau 46
2.6.2 Nœuds 46
2.6.3 Arcs 46
2.6.4 Boucle de dépôts et vidages 47
2.6.5 Véhicules 47
2.6.6 Distancier 47
2.6.7 Planification 47
2.6.8 Solution et tournées 47
2.6.9 Traitement d'une tâche 47
2.7 Conclusion 48

CHAPITRE III Algorithmes génétiques pour le CARP

3.1 Introduction 49
3.2 Calcul du distancier 49
3.3 Heuristiques constructives 50
3.3.1 Path-Scanning 51
3.3.1.1 Principe 51
3.3.1.2 Algorithme 51
3.3.1.3 Complexité de la version de base 52
3.3.2 Augment-Merge 52

3.3.2.1 Principe 52
Phase Augment 53
3.3.2.2 Algorithme 54
3.3.2.3 Analyse de complexité 55
3.3.3 Algorithme d'Ulusoy 56
3.3.3.1 Principe 56
3.3.3.2 Algorithme 57
3.3.3.3 Complexité 58
3.4 Rappels sur les algorithmes génétiques 58
3.4.1 Introduction 58
3.4.2 Principe 59
3.4.3 Résultats théoriques 61
3.4.4 Algorithmes génétiques publiés pour des problèmes de tournées 61
3.5 Composants pour des GA pour le CARP 62
3.5.1 Chromosomes : représentations et évaluations 62
3.5.1.1 Chromosome avec délimiteurs de tournées 63
3.5.1.2 Chromosome sans délimiteurs de tournées et évaluation 63
3.5.1.3 Illustation de la flexibilité de Split : minimisation du makespan 64
3.5.2 Croisements étudiés 65
3.5.3 Mutations simples étudiées 68
3.5.4 Remplacement de la mutation par une recherche locale 68
3.5.5 Structure de la population et initialisation 69
3.5.6 Sélection et remplacement 70
3.5.6.1 Algorithme génétique incrémental 70
3.5.6.2 Algorithme génétique générationnel 71
3.5.7 Redémarrages 71
3.6 Sélection des composants et des paramètres 72
3.6.1 Implémentation et jeux de tests 72
3.6.2 Choix des composants et des paramètres de l'algorithme 72
3.7 Résultats des évaluations numériques 75
3.7.1 Format des tableaux de résultats 75
3.7.2 Résultats pour l'algorithme avec chromosomes sans délimiteurs 76
3.7.3 Résultats pour l'algorithme avec chromosomes à délimiteurs 77
3.7.4 Minimisation du makespan 81
3.8 Conclusion 82

CHAPITRE IV CARP stochastique

4.1 Introduction 85
4.2 Définition du problème 86
4.3 Hypothèses de travail 87
4.4 Calcul des réplications 89
4.5 Littérature sur les problèmes de tournées stochastiques 90
4.6 Etude de la robustesse 90
4.6.1 Approche *serrée* 91
4.6.2 Approche *relaxée* 92
4.7 Evaluations numériques 92
4.7.1 Comparaison de la robustesse des deux approches sur les fichiers gdb 93
4.7.1.1 Approche *serrée* 93
4.7.1.2 Approche *relaxée* 94
4.7.1.3 Comparaison de l'approche *serrée* et l'approche *relaxée* sur les fichiers gdb .. 95
4.7.1.4 Exemple sur gdb19 97
4.7.2 Comparaison de la robustesse des deux approches sur les fichiers val 99
4.7.3 Comparaison de la robustesse des deux approches sur les fichiers egl 102
4.7.4 Synthèse des évaluations 105
4.8 Vers un GA intrinsèquement robuste 106
4.8.1 Caractérisation analytique de la robustesse 106
4.8.2 Adaptation du GA 108
4.8.3 Evaluations numériques 108
4.9 Conclusion 108

CHAPITRE V CARP étendu

5.1 Introduction 111
5.2 Définition du ECARP 112
5.3 Méthodes pour les ECARP de taille petite à moyenne 112
5.3.1 Une formulation linéaire du ECARP 113
5.3.1.1 Données 113
5.3.1.2 Variables de décisions 113
5.3.1.3 Modèle linéaire 114
5.3.1.4 Adaptations du modèle linéaire de base 115
5.3.2 Méthode de coupes 117

5.3.2.1 Relaxation de *P* 117
5.3.2.2 Identification des sous-tours 117
5.3.2.3 Elimination des sous-tours 118
5.3.2.4 Structure générale de l'algorithme de coupes 118
5.3.3 Adaptation de l'algorithme génétique 118
5.3.3.1 Gestion des interdictions de tourner et distancier entre arcs 118
5.3.3.2 Prise en compte des autres contraintes du ECARP 120
5.3.3.3 Cas bicritère : surcharge maximale et coût total 120
5.3.4 Evaluation numérique de la méthode de coupes et du GA 121
5.3.4.1 Génération des fichiers *gdbe* 121
5.3.4.2 Evaluation de l'algorithme de coupes et de l'algorithme génétique 122
5.3.4.3 Optimisation hiérarchique à deux objectives par l'algorithme génétique 125
5.4 Heuristiques pour les ECARP de grande taille 126
5.4.1 Path-Scanning étendu (EPS) 126
5.4.2 Augment-Merge étendu (EAM) 127
5.4.3 Heuristique d'Ulusoy étendu (EUH) 130
5.4.3.1 Version respectant l'orientation des arêtes du tour géant 130
5.4.3.2 Version calculant les meilleures orientations des arêtes du tour géant 131
5.4.4 Génération aléatoire des grandes instances LPR 133
5.4.4.1 Génération d'un squelette planaire et non orienté 133
5.4.4.2 Introduction des sens uniques 135
5.4.4.3 Sélection des tâches, des quantités et des arêtes 137
5.4.4.4 Autres données 137
5.4.4.5 Les fichiers générés 138
5.4.5 Evaluation numérique des heuristiques 139
5.5 Conclusion 141

CHAPITRE VI CARP périodique

6.1 Introduction 145
6.2 Travaux publiés sur les problèmes de tournées périodiques 146
6.3 La planification de la collecte des déchets dans la réalité 146
6.4 Description et formalisation du PCARP 147
6.4.1 Version de base du PCARP et complexité 147
6.4.2 Contraintes additionnelles 148

6.4.2.1 Demandes et coûts dépendant du temps 148
6.4.2.2 Espacements 151
6.4.2.3 Nombre de véhicules 152
6.5 Approches de résolution 152
6.5.1 Heuristiques initiales 153
6.5.2 Résolution globale du PCARP avec un GA périodique (PGA) 154
6.5.2.1 Chromosomes 154
6.5.2.2 Evaluation du chromosome 154
6.5.2.3 Population initiale 155
6.5.2.4 Croisement 155
6.5.3 Approche en deux phases 158
6.5.3.1 Programmes linéaires simples pour la phase d'affectation 158
6.5.3.2 Réutilisation de LIH, BIH ou PGA pour l'affectation par BIH et PGA 160
6.6 Evaluation numérique 160
6.6.1 Implémentation et jeux de tests 160
6.6.2 Résultats des évaluations numériques 161
6.7 Conclusion 162
Conclusion générale et perspectives 167
Bibliographie 173

Glossaire

A	ensemble des arcs du graphe interne *G*, indexés de 1 à *m*
AM	Augment Merge : Heuristique constructive pour le CARP
b(*u*)	nœud de début de l'arc *u*
BIH	Best Insert Heuristic : Heuristique constructive pour le PCARP
c(*u*)	coût de traversée de l'arc
c(u,p)	coût de l'arc *u* pour une période *p*
CARP	problème de tournées sur arcs avec contraintes de capacité
Carpet	meilleure méthode pour le CARP au début de nos travaux (méthode tabou de Hertz *et al*., 2000)
clear (*L*)	initialise la liste *L* à vide (les cellules ne sont pas libérées)
comb(*u*)	sous ensembles d'indices de combinaisons permises pour *u*
concat (*L*,*L*')	*fonction* concaténant les cellules de *L*' après celles de *L*
copy (*L*,*i*,*j*)	*fonction* renvoyant une liste avec les cellules *L*(*i*) à *L*(*j*) inclus
cost	coût total, fonction générique applicable à une tournée ou à une solution
$cost(S,\omega_i)$	coût de *S* dans la réplication ω_i
$\overline{cost(S,ne)}$	moyenne expérimentale calculée sur *ne* évaluations de $cost(S,\omega_i)$
D	matrice *m*×*m* des "distances" entre arcs (coûts des chemins optimaux)
d(*u*,*v*)	distance de l'arc *u* à l'arc *v* (*u* et *v* non inclus)
days(*S*,*u*)	une liste ordonnée de jour de traitement de *u* dans une solution S
DGA	Daily GA
done(*u*)	variable booléenne égale à vrai ssi l'arc requis *u* est traité
dump(*u*)	boucle fictive de l'exutoire correspondant à *term*(*u*)
dumploops	sous-ensemble des index des boucles fictives pour les exutoires
dumpnodes	ensemble des nœuds de vidage

e(*u*)	nœud de fin de l'arc *u*
EAM	Extended Augment Merge: AM étendue pour le ECARP
ECARP	CARP étendu
ELOX	Extended Linear Order Crossover
EM	Merge étendue
enqueue (*L*,*e*)	insère *e* à la fin de *L*, équivaut à *insert* (*L*,|*L*|+1,*e*)
EPS	Extended Path Scanning : PS étendue pour le ECARP
EUH	Extended Ulusoy : UH étendue pour le ECARP
exutoire	lieu de vidage
f(*u*)	fréquence de l'arc *u* (nombre de traitement sur l'horizon)
first (*L*)	*fonctions* équivalentes à *L*(1)
G	graphe interne complètement orienté, modélisant le réseau réel
GA (ou MA)	Genetic Algorithm (ou Memetic Algorithm)
H	horizon de planification
HLP	Haut Le Pied : les traversées sans collecte
i, j	indices typiques pour les nœuds
insert (*L*,*k*,*e*)	insère une cellule contenant *e* avant *L*(*k*)
inv(*u*)	arc inverse si *u* fait partie d'une arête à traiter (sinon *inv*(*u*) = 0)
invert (*L*)	inverse l'ordre des cellules dans L
IPGA	Improved Periodic GA
L	autonomie des véhicules (coût maximal permis pour une tournée)
last (*L*)	*fonctions* équivalentes à *L*(|*L*|)
LIH	Last Insert Heuristic : Heuristique constructive pour le PCARP
load	charge totale, fonction générique applicable à une tournée ou à une solution
LOX	Linear Order Crossover
LS	Local Search : procédure de recherche locale
M	Merge : Heuristique AM sans la phase Augment
m	nombre d'arcs du graphe interne *G*
multitour	plusieurs allers-retours aux lieux de vidage par le même véhicule
n	nombre de nœuds dans *G*
ncomb	nombre de combinaisons dans *scomb*
ndumps	nombre de nœuds de vidage
ne	nombre de réplications
np	nombre de périodes ("jours") de l'horizon
ns	nombres total de services sur l'horizon, somme des fréquences

nva	nombre de véhicules disponibles
nvu	nombre de véhicules réellement utilisés, égal au nombre de tournées
$nvu(S,\omega_i)$	nombre de tournées de S dans la réplication ω_i
$\overline{nvu(S,ne)}$	moyenne expérimentale calculée sur *ne* évaluations de $nvu(S,\omega_i)$
OX	Order Crossover
P	matrice $m \times m$ stockant un des chemins optimaux pour tout couple d'arcs
$P(u,v)$	arc prédécesseur de v sur le chemin optimal de u à v
p, q	indices habituels pour les jours
PCARP	CARP périodique
$pen(u,v)$	pénalité pour tourner de u vers v, $v \in suc(u)$
PGA	Periodic GA
$pre(u)$	liste des arcs à partir desquels on peut tourner vers l'arc u
prod(u,p)	production de l'arc u à la période p avec accumulation d'une période à l'autre
PS	Path Scanning : Heuristique constructive pour le CARP
push (L,e)	insère e en tête de L, équivaut à *insert* $(L,1,e)$
Q	capacité commune des véhicules
R	sous-ensemble des arcs requis, $\rho = \|R\|$
$r(u)$	demande de l'arc u (0 pour les arcs qui ne sont pas à traiter)
$r(u,\omega_i)$	demande sur l'arc u dans la réplication ω_i
r(u,p)	demande de l'arc u pour une période p
remove (L,k,e)	détruit la cellule $L(k)$ après avoir renvoyé son élément dans e
réplication	solution stochastique du SCARP (avec simulation de demandes aléatoires)
s	index de la boucle fictive pour le dépôt, $s = (\sigma, \sigma)$
S	solution du CARP composé de *nvu* tournées, $S = (\theta_1, \theta_2, \ldots, \theta_{nvu})$
SCARP	CARP stochastique
scomb	liste de combinaison dans tous le réseau
Split	procédure de découpage et d'évaluation du chromosome sans délimiteurs
$spmax(u)$	espacement maximal entre deux visites consécutives
$spmin(u)$	espacement minimal entre deux visites consécutives
$suc(u)$	listes des arcs vers lesquels on peut tourner à partir de l'arc u
t	index de la boucle fictive si l'exutoire est unique, $t = (\varphi, \varphi)$
$term(u)$	coût minimal pour terminer une tournée après l'arc u
tricot	collecter les deux côtés d'une rue en même temps
$trip(u)$	index de la tournée qui traite u, nul si u n'est pas encore traité
u, v	indices typiques pour les arcs

UH	Ulusoy : Heuristique constructive pour le CARP
V	ensemble des nœuds de G, indexés de 1 à n
$w(u)$	coût de traitement de u (0 pour les arcs qui ne sont pas à traiter)
X1	Croisement à un point de coupure
α	nombre de tâches-arcs
$\Gamma^{-}(i)$	ensemble de prédécesseurs du nœud i
$\Gamma(i)$	ensemble des successeurs du nœud i
δ	coût d'une opération de vidage
θ	une tournée de la solution S
ε	nombre de tâches-arêtes
ρ	nombre d'arcs requis codant des tâches, $\rho = 2\varepsilon + \alpha = \varepsilon + \tau$
τ	nombre total de tâches, $\tau = \varepsilon + \alpha$
φ	index du nœud de vidage s'il n'y en a qu'un
$\sigma(nvu(S,ne))$	écart-type expérimental calculé sur ne évaluations de $nvu(S,\omega_i)$
$\sigma(cost(S,ne))$	écart-type expérimental calculé sur ne évaluations de $cost(S,\omega_i)$
$\omega^{+}(i)$	ensemble des arcs débutant au nœud i

Liste des figures

Figure 1.1 – Illustration de Shorten 25
Figure 1.2 – Synthèse des problèmes de base en tournées sur arcs 30
Figure 2.1 – Exemple de réseau de rues en collecte des déchets 32
Figure 2.2 – Exemple de tournée simple et de multitour 33
Figure 2.3 – Problèmes étudiés dans la thèse 35
Figure 2.4 – Représentation interne du graphe 38
Figure 2.5 – Graphe associé à un réseau de rue avec contraintes 39
Figure 2.6 – Codage des rues 40
Figure 2.7 – Graphe interne pour l'exemple de la figure 2.1 41
Figure 2.8 – Plus court chemin entre arcs avec interdictions de tourner 42
Figure 2.9 – Partitions de l'ensemble des arcs 43
Figure 3.1 – Heuristique Augment-Merge 53
Figure 3.2 – Principe de l'heuristique d'Ulusoy 57
Figure 3.3 – Itération de base d'un algorithmes génétique incrémental 60
Figure 3.4 – Les deux types de chromosomes pour le CARP 62
Figure 3.5 – Opérations sur chromosomes binaires et de permutation 65
Figure 3.6 – Exemple de parents avec un enfant obtenu par le croisement LOX 67
Figure 3.7 – Mouvements 2-opt de N5 sur une tournée ou deux 69
Figure 4.1 – Comparaison entre la solution prévue avec des demandes déterministes et une solution perturbée par des demandes aléatoires 88
Figure 4.2 – Résultats des paramètres $\Delta cost(S_2, ne)$, $\Delta nvu(S_2, ne)$ sur les fichiers gdb 97
Figure 4.3 – Comparaison de $extratrip(S_1, ne)$ et $extratrip(S_2, ne)$ sur les fichiers gdb 97

Figure 4.4 – Représentation graphique de la solution déterministe de l'approche serrée du gdb19. 98
Figure 4.5 – Comparaison de extratrip(S_1, ne) et extratrip(S_2, ne) sur les fichiers val 100
Figure 4.6 – Comparaison de extratrip(S_1, ne) et extratrip(S_2, ne) sur les fichiers egl 104
Figure 5.1 – Un cas de traversée multiple d'arête 113
Figure 5.2 – Courbe d'évolution de la fonction objectif du GA sur le problème gdbe17 123
Figure 5.3 – Représentation graphique du problème gdbe19 124
Figure 5.4 – Solution du programme relaxé initial 124
Figure 5.5 – Solution obtenue à l'itération 2 (problème P'_2 : $P'_1 + X_{11} + X_{12}$) 124
Figure 5.6 – Solution obtenue à l'itération 3 (problème P'_3 : $P'_2 + X_{21}$) 125
Figure 5.7 – Recherche des meilleures orientations d'arêtes dans G(i,j) 132
Figure 5.8 – Exemple de squelette de départ avec nsqv = 2 et nsqh = 3 133
Figure 5.9 – Les modifications possibles d'une maille 134
Figure 5.10 – Exemple de modifications multiples de mailles et de graphe modifié 134
Figure 5.11 – Un déplacement de nœud dans un cercle de rayon 0.5×side 134
Figure 5.12 – Graphe obtenu après déplacement des nœuds 135
Figure 5.13 – Graphe avec deux clusters 135
Figure 5.14 – Recherche des arcs protégés 136
Figure 5.15 – Circuit protégé garantissant la forte connexité 137
Figure 5.16 – Déviations par rapport à LB en % par problème et par heuristique 140
Figure 5.17 – Comparaison des écarts des heuristiques sur les fichiers gdb, val, egl et lpr 141
Figure 5.18 – Exemple de lpr-b-01 : n = 28, m = 68, τ = 50 144
Figure 6.1 – Une classification simple des problèmes périodiques en tournées sur arcs 150
Figure 6.2 – Exemple de croisement ELOX 157

Liste des tables

Table 1.1 – Résultats de la meilleure borne pour le CARP sur les instances de la littérature 21
Table 3.1 – Sélection des meilleurs composants 72
Table 3.2 – Les paramètres standards du MA 73
Table 3.3 – Résultats du MA avec chromosomes à délimiteurs sur les fichiers gdb 76
Table 3.4 – Résultats du MA avec chromosomes à délimiteurs sur les fichiers val 77
Table 3.5 – Résultats du MA avec chromosomes sans délimiteurs sur les fichiers gdb 78
Table 3.6 – Résultats du MA avec chromosomes sans délimiteurs sur les fichiers val 79
Table 3.7 – Résultats du MA avec chromosomes sans délimiteurs sur les fichiers egl 80
Table 3.8 – Comparaison globale des performances 81
Table 3.9 – Minimisation du makespan sous contrainte d'une flotte limitée 82
Table 4.1 – Résultat de l'approche serrée sur les fichiers gdb 94
Table 4.2 – Résultat de l'approche relaxée sur les fichiers gdb 95
Table 4.3 – Comparaison des performances relatives des deux approches sur les fichiers gdb 96
Table 4.4 – Description de l'instance gdb19 98
Table 4.5 – Génération aléatoire des données pour la réplication ω_1 sur gdb19 99
Table 4.6 – Comparaison des solutions de l'approche serrée du gdb19 99
Table 4.7 – Résultat de l'approche serrée sur les fichiers val 100
Table 4.8 – Résultat de l'approche relaxée sur les fichiers val 101
Table 4.9 – Comparaison des performances relatives des deux approches sur les fichiers val 102
Table 4.10 – Résultat de l'approche serrée sur les fichiers egl 103
Table 4.11 – Résultat de l'approche relaxée sur les fichiers egl 104
Table 4.12 – Comparaison des performances des deux approches sur les fichiers egl 105
Table 4.13 – Déviations des solutions stochastiques par rapport aux solutions déterministes pour les deux approches 105
Table 4.14 – Loi de probabilité de cost(θ_j,ω_i) 107

Table 5.1 – Caractéristiques des fichiers gdbe 121
Table 5.2 – Exemple de fichier gdbe : gdbe19 122
Table 5.3 – Résolution du ECARP par la méthode de coupes et du GA pour les fichiers gdbe 123
Table 5.4 – Résultats du GA pour le problème bicritère 125
Table 5.5 – Résultats des heuristiques sur les fichiers LPR 139
Table 5.6 – Comparaison des heuristiques sur les fichiers gdb, val, egl et lpr (temps en ms) 141
Table 5.7 – Caractéristiques des fichiers lpr 143
Table 6.1 – Définition des combinaisons 160
Table 6.2 – Synthèse des résultats sur nos 23 instances gdbf 161
Table 6.3 – Résultats du PGA sur les 23 fichiers gdbf 163
Table 6.4 – Détail des résultats du PGA par période 164
Table 6.5 – Résultats du DGA par période (affectation par BIH) 165
Table 6.6 – Résultats du IPGA (affectation par PGA) 166

Introduction

La logistique est apparue dans le domaine militaire au moment où l'on a tenté de rationaliser l'expérience acquise au cours des campagnes napoléoniennes. Au 19 siècle sont nés dans l'armée française les services de l'intendance et du train des équipages. Tous les états-majors comportaient désormais un bureau logistique. Les calculs de besoins, de délais de transport, d'espace de ramassage et de stocks faisaient apparaître une sorte de nouvelle science que l'on n'appelait pas encore recherche opérationnelle mais qui en avait déjà un peu l'esprit et que l'on appela *logistique* pour en consacrer le caractère *logico-mathématique*. L'expérience acquise dans l'armée a été ensuite transposée aux entreprises.

L'activité de transport est le cœur même de la logistique, c'est l'un de ses postes de coûts les plus importants de telle sorte que l'organisation logistique est souvent déterminée par l'optimisation des coûts de transports. Qui dit transport, dit organisation des tournées de véhicules. Une meilleure organisation de ces dernières présente un potentiel d'économies majeur. C'est cette importance accrue des problèmes d'optimisation des tournées dans le secteur de transport qui a attiré de plus en plus les chercheurs et les gestionnaires d'entreprises.

Le problème de tournées de véhicules consiste à déterminer, pour un ensemble donné de ("clients"), par quels véhicules de la flotte ils seront visités et à quel moment dans la tournée ils le seront. Deux grandes classes émergent en fonction de la position des ("clients") : s'ils se trouvent sur des points précis d'un réseau, le problème étudié est un *problème de tournées sur nœuds*. En revanche, si les ("clients") sont localisés sur des liens du réseau, il s'agit d'un *problème de tournées sur arcs*. La deuxième classe apparaît quand les ("clients") se trouvent distribués le long des rues ou lorsque la rue elle-même nécessite un service.

Les tournées sur arcs ont été beaucoup moins étudiées, comparé aux tournées sur nœuds, bien qu'elles permettent de modéliser un grand nombre d'applications réelles comme la collecte des déchets, le déneigement des routes, le relevé de compteurs d'électricité, la distribution de courrier et l'inspection de lignes à haute tension. Vu la complexité des problèmes de tournées sur arcs, ils ont été abordés de façon progressive par la communauté scientifique, en commençant par des cas simplifiés et en ajoutant une à une les difficultés. Des recherches ont par ailleurs porté sur des applications particulières exigeant des modélisations et des techniques de résolution spécifiques. Il aurait fallu du temps avant de pouvoir proposer un modèle plus général nommée *CARP* (pour *Capacitated Arc Routing Problem*), regroupant ainsi la majorité des problèmes de tournées sur arcs.

Le CARP reste encore loin des applications réelles et ne concerne qu'un horizon à court terme. D'où la nécessité de développer des approches de résolution exploitant la multitude de contraintes des problèmes réels plutôt que de tenter d'utiliser les idées qui ne sont efficaces que pour les problèmes peu contraints. Si on regarde ce qui a été déjà proposé pour résoudre le CARP, on trouve principalement des heuristiques constructives, peu de métaheuristiques et encore moins de méthodes exactes. Ainsi, il faut continuer le travail initié par les chercheurs jusqu'à maintenant, en proposant non seulement des généralisations du CARP mais aussi des méthodes efficaces en termes de qualité de solution et de temps de calcul. En effet, il est maintenant possible de répondre à ce besoin grâce aux progrès de la recherche opérationnelle et à la puissance accrue des ordinateurs.

C'est pour toutes ces raisons et parce que le CARP a un intérêt industriel certain mais aussi un intérêt scientifique important, que nous consacrons cette thèse à son étude. Pour rester au plus près des applications réelles et pour pouvoir en faire ressortir toutes leurs complexités, nous choisissons une application de référence qui est la *collecte des déchets urbains*, mais notre objectif est de proposer des modèles et des méthodes de résolution plus générales et plus flexibles pouvant s'adapter à d'autres problèmes en tournées sur arcs.

Notre contribution consiste en une extension des recherches menées jusqu'à présent sur le CARP, par le développement de nouveaux modèles prenant en compte des contraintes additionnelles. Nous proposons d'étendre le CARP de la littérature vers trois problèmes : le *ECARP* (*Extended CARP*), le *SCARP* (*Stochastic CARP*) et le *PCARP* (*Periodic CARP*). Le CARP et les nouveaux problèmes sont extrêmement combinatoires, ce qui nécessite une analyse très poussée pour pouvoir proposer des méthodes efficaces. Notre but est de concevoir des outils et des méthodes de résolution adaptés aux problèmes de grandes tailles avec comme objectif premier de proposer des solutions de bonnes qualités. Nous nous intéressons aussi à la robustesse de ces solutions car cette propriété revêt un intérêt tout particulier pour des problèmes réalistes comme ceux issus du domaine de la collecte des déchets.

Dans un contexte d'aide à la décision, il parait intéressant de pouvoir proposer comme résultat final plusieurs configurations de solutions et de pouvoir prendre en compte des contraintes très spécifiques. Ce mémoire propose les *premiers algorithmes génétiques* pour les problèmes de tournées sur arcs et traite de leurs apports ainsi que de leurs flexibilités dans le processus de généralisation que nous proposons pour le CARP de la littérature. Les algorithmes génétiques travaillent en parallèle sur un ensemble de solutions et peuvent donc offrir au décideur tout un panel de scénarios. Ils combinent ces solutions entre elles pour en engendrer de meilleures, selon des mécanismes inspirés de l'évolution biologique. Il s'agit notamment dans notre étude de construire une plate-forme logicielle commune basée sur les algorithmes génétiques pour le CARP, le SCARP, le ECARP et le PCARP.

Ce mémoire est composé de 6 chapitres.

Le premier est dédié à un état de l'art des problèmes de tournées sur arcs. Il passe en revue les modèles, les méthodes ainsi que les applications étudiées pour ce type de problème.

Le deuxième chapitre est un chapitre intermédiaire, où nous présentons notre motivation pour le choix du CARP, le cadre de l'étude et une description de l'application de collecte des déchets urbains. Nous proposons ensuite des structures de données, des hypothèses de travail et des notations communes pour les problèmes étudiés dans la thèse.

Le troisième chapitre est consacré à des algorithmes génétiques hybrides pour résoudre le CARP de la littérature. Rappelons que ce sont les premiers algorithmes évolutionnistes pour ce type de problème. Nous donnons ainsi les concepts de base qui sont nécessaires pour la compréhension des autres chapitres, puisque chaque nouveau problème proposé aura au moins un algorithme génétique comme méthode de résolution.

Les trois autres chapitres détaillent chacun un des nouveaux problèmes. Dans le chapitre quatre, nous traitons le SCARP. L'objectif de ce dernier est d'étudier la robustesse des solutions et éventuellement de déterminer des moyens pratiques pour augmenter cette robustesse par rapport aux variations des données comme les *quantités*. Nous modélisons mathématiquement ce problème et nous considérons des *quantités incertaines* pour pouvoir étudier la sensibilité aux aléas des solutions trouvées par notre algorithme génétique. Ce genre d'incertitudes et évidemment la règle dans des applications comme la collecte de déchets.

Le chapitre cinq expose le ECARP. Nous considérons un *réseau plus réaliste* avec des rues à double sens et à sens unique, des interdictions de tourner, des lieux de déchargement multiples, etc. Nous proposons une méthode de coupes basée sur la programmation en nombres entiers pour les problèmes de taille moyenne et une généralisation des meilleures heuristiques connues pour le CARP (*Path-Scanning*, *Augment-Merge* et la *méthode d'Ulusoy*). Une extension de l'algorithme génétique pour résoudre le ECARP est aussi proposée. Pour tester la performance de nos méthodes sur ce nouveau problème, nous présentons un générateur aléatoire qui imite les réseaux routiers et fournit des instances plus réalistes et de grande taille.

Dans le dernier chapitre, nous étudions un problème tactique qui est une extension temporelle du CARP appelée PCARP ou *problème de planification en tournées sur arcs*. Un *horizon* de plusieurs *périodes* ou *jours* est considéré et une *fréquence de collecte* est donnée pour chaque rue (exemple 3 fois par semaine). Le but est d'affecter les rues aux périodes, en respectant les fréquences, et de construire les tournées dans chaque période de façon à minimiser un coût total sur l'horizon. Nous définissons le premier modèle formel pour ce problème réel. Nous proposons de prendre en compte plusieurs généralisations comme des demandes et des coûts dépendants du temps et des espacements entre les collectes. Nous présentons deux approches de résolution. La première procède en deux phases, qui consistent d'abord à résoudre le problème d'affectation (niveau tactique) puis à résoudre le CARP obtenu dans chaque période avec notre algorithme génétique (niveau opérationnel). La deuxième approche consiste à traiter simultanément les niveaux tactiques et opérationnels par une version étendue de l'algorithme génétique.

La conclusion présente une synthèse des travaux que nous avons réalisés et propose quelques perspectives de recherches futures.

CHAPITRE 1
Etat de l’art en tournées sur arcs

1.1 Introduction

Dans le monde technologique dans lequel nous vivons, la croissance des activités de distribution et de collecte de produits a créé une pression sur de nombreuses entreprises et collectivités. Ces dernières doivent résoudre des problèmes d’optimisation de grande taille liés au secteur du transport. C’est ce besoin d’optimisation dans ce secteur très concurrentiel qui a enrichi la recherche en problèmes de tournées de véhicules. On y distingue deux familles de problèmes selon l'emplacement des tâches à effectuer : si elles se trouvent en des points précis d’un réseau, on a affaire à un *problème de tournées sur nœuds*. En revanche, si elles sont situées sur des liens (arcs ou arêtes), il s’agit d’un *problème de tournées sur arcs*.

Comme cette thèse est consacrée aux problèmes de tournées sur arcs, l'état de l'art décrit dans ce chapitre, concerne uniquement ce genre de tournées. Les deux premières parties évoquent brièvement l’origine des problèmes de tournées sur arcs et la possibilité, toute théorique, de les convertir en problèmes sur nœuds. La troisième partie présente les principaux problèmes de tournées sur arcs. Elle passe en revue les modèles utilisés et les méthodes de résolution, exactes et approchées. On s’intéresse plus particulièrement au problème de tournées sur arcs avec contraintes de capacité (*CARP* ou *Capacitated Arc Routing Problem*). C’est un problème général qui contient tous les autres problèmes de tournées sur arcs en cas particuliers. Enfin, la quatrième et dernière partie évoque quelques applications du CARP ainsi que des généralisations récentes de ce problème.

1.2 Historique et points d’entrée pour les tournées sur arcs

La ville de *Königsberg* (maintenant Kaliningrad) avait *sept ponts* sur la rivière Pregel. Les amateurs de questions mathématiques et de promenades digestives se demandaient s'il était possible de faire un tour dans la ville en passant une fois exactement sur chaque pont. Leonhard Euler (1736), en construisant le graphe représentatif non orienté, constata qu'il avait des noeuds de degré impair et démontra que cela rendait impossible une telle promenade. Ce qui a donné le premier résultat formel de la théorie des graphes : «*Un parcours est eulérien s'il passe exactement une fois par chaque arête, un graphe est dit eulérien s'il admet un tel parcours fermé, ce qui revient à être connexe et à avoir tous les nœuds de degré pair*». Depuis, des conditions d'existence d'un parcours eulérien ont été découvertes pour les graphes orientés et mixtes, voir par exemple Ford et Fulkerson (1962).

Si le graphe est eulérien, l'extraction d'un parcours passant une seule fois par tous les liens (arcs ou arêtes) du graphe est relativement simple : des méthodes ont été proposées très tôt par Hierholzer (1873) et une synthèse des différentes approches peut être trouvée dans Fleischner (1991). Si le graphe est non eulérien, un parcours qui couvre tous les liens contient forcément des copies de certains liens. Le problème pour les graphes valués est alors de définir un parcours de coût minimal passant au moins une fois par chaque lien. Il s'agit du problème du *postier chinois*, défini par Guan (1962) dans le cas d'un graphe non orienté. C'est le problème d'optimisation le plus simple en tournées sur arcs.

La littérature consacrée aux problèmes de tournées sur arcs est moins riche que celle relative aux problèmes de tournées sur nœuds. Néanmoins, elle bénéficie déjà de quelques synthèses (*surveys*) comme celle d'Eilselt et *al*. (1995a, 1995b) et celle d'Assad et Golden (1995). Plus récemment, Hertz et Mittaz (1998) ont présenté une revue très complète des méthodes heuristiques. La dernière synthèse parue a été coordonnée par Dror (2000), avec le livre *Arc Routing: Theory, Solutions and Applications*.

1.3 Transformation des problèmes de tournées sur arcs en problèmes de tournées sur nœuds

Les problèmes de tournées sur nœuds comme le *VRP* (*Vehicle Routing Problem*) sont plus étudiés que les problèmes sur arcs, probablement à cause de leurs applications précoces en logistique de distribution. De nombreux modèles et méthodes de résolution ont été proposés, citons les articles de Bodin et *al.* (1983), de Christofides (1985), de Laporte et Nobert (1984) et de Laporte (1992a, 1992b, 1996, 1998). Il est donc tentant de transformer les problèmes de tournées sur arcs en problèmes de tournées sur nœuds, afin de réutiliser les algorithmes déjà disponibles.

Pearn *et al*. (1987) sont les premiers à avoir proposé une telle transformation, pour des réseaux non orientés. Elle consiste à remplacer chaque arête par trois nœuds et à répartir la quantité de l'arête sur ces trois nœuds. Laporte (1995) a détaillé une autre transformation pour certains problèmes de tournées sur arcs dans un réseau non orienté. Le point commun entre les problèmes traités par Laporte est l'absence de contrainte de capacité. La méthode consiste à créer deux nœuds à partir de chaque arête, en ne multipliant ainsi que par deux la taille du problème, au lieu de trois. Il reste ensuite à résoudre un problème de tournées généralisé (appelé *GRP* ou *General Routing Problem*), qui consiste à déterminer un tour de coût minimal passant au moins une fois sur chaque arête et sur chaque nœud.

Dror et Langevin (1997) ont converti le problème du postier rural orienté (*cf.* section 1.6) en un problème du voyageur de commerce généralisé, dans lequel les nœuds du graphe sont répartis en groupes et où il suffit de visiter un nœud par groupe. Cette approche est intéressante mais ne permet pas de gérer des contraintes de capacité. Mullaseril et Dror (1996) ont travaillé sur un problème de tournées sur arcs avec contraintes de capacité et fenêtres horaires, dans un graphe orienté. Ils ont proposé une méthode transformant tout arc du graphe en un nœud à visiter dans un problème de tournées sur nœuds, en conservant ainsi la taille du problème. Gueguen (1999) a étendu cette technique aux graphes non orientés et aux graphes mixtes, avec ou sans contraintes de capacité.

Il faut souligner que ces techniques de conversion ont seulement un intérêt théorique : elles ne sont pas utilisées en pratique. En effet, dans certains cas, une partie de la structure du

problème est perdue et l'augmentation de taille rend souvent le problème insoluble. De plus, ces transformations ne sont plus valides dès qu'on introduit des contraintes supplémentaires, ce qui est un gros handicap pour traiter des applications réelles.

1.4 Notations pour les programmes linéaires du chapitre

Avant de passer en revue les principaux problèmes de tournées sur arcs, nous introduisons des notations permettant une présentation homogène des formulations à base de programmes linéaires en nombres entiers. Nous allons aussi utiliser le vocabulaire de la théorie des graphes, mais il est supposé connu (voir par exemple le livre de Prins, 1993).

Le réseau est noté $G = (V, U = A \cup E)$. V est un ensemble de n nœuds indexés par i (avec un dépôt au nœud 1), A un ensemble de ma arcs et E un ensemble de me arêtes. On a un graphe *orienté* si E est vide, *non orienté* si A est vide et *mixte* si A et E sont non vides. On appelle *liens* les $m = ma + me$ éléments (arcs ou arêtes) de U. Pour un graphe orienté, nous indexons les variables par a et, dans le cas non orienté et mixte, par e. Quand cela est plus clair, on utilise des index de nœuds i et j au lieu de e ou a (par exemple x_{ij} au lieu de x_e, x_a).

Pour un nœud i ou un lien e, on note :

$d^+(i)$	le demi-degré extérieur (nombre d'arcs partant de i)
$d^-(i)$	le demi-degré intérieur (nombre d'arcs arrivant en i)
$d(i) = d^+(i) + d^-(i)$	le degré (nombre d'arcs *et* d'arêtes incidents à i)
V_{odd}	l'ensemble des nœuds de degré impair
c_e	la *longueur* ou coût de traversée du lien
r_e	la *demande* (quantité à traiter) sur e

On note $G_R = (V_R, R)$ le graphe induit par les liens *requis* (de demandes non nulles), avec :

V_R	le sous-ensemble des nœuds extrémités des liens requis
E_R	le sous-ensemble des arêtes requises
A_R	le sous-ensemble des arcs requis
$R = E_R \cup A_R$	le sous-ensemble des liens requis

Les véhicules, supposés identiques, sont indexés par $k \in I = \{1,\ldots, K\}$. On définit :

K	le nombre de véhicules disponibles
W	la capacité commune des véhicules
r_{tot}	la demande totale du réseau (somme des r_e)
$K^* = \lceil r_{tot} / W \rceil$	le nombre minimal de véhicules pour traiter r_{tot}

Les notations suivantes concernent un sous-ensemble de nœuds $S \subseteq V \setminus \{1\}$:

$\delta^+(S)$	ensemble des arcs (i,j) sortant de S
$\delta_R^+(S)$	restriction de $\delta^+(S)$ aux arcs requis, $\delta_R^+(S) = \delta^+(S) \cap R$
$\delta^-(S)$, $\delta_R^-(S)$	ensemble des arcs entrant dans S et sa restriction à R
$\delta(S)$, $\delta_R(S)$	ensemble des arêtes incidentes à S et sa restriction à R
$b(S) = \lvert\delta^-(S)\rvert - \lvert\delta^+(S)\rvert - \lvert\delta(S)\rvert$	bilan de l'ensemble S

$b_R(S) = \|\delta_R^-(S)\|-\|\delta_R^+(S)\|-\|\delta_R(S)\|$	bilan de l'ensemble S pour les liens requis
$E(S)$, $A(S)$	ensembles d'arêtes et d'arcs avec extrémités dans S
$E_R(S)$, $A_R(S)$	restrictions de $E(S)$ et $A(S)$ aux liens requis
$r_{tot}(S)$	quantité totale à traiter sur $E_R(S) \cup \delta_R(S)$
$K(S)=\lceil r_{tot}(S)/W \rceil$	nombre minimal de véhicules pour traiter $r_{tot}(S)$
$x(S) = \sum_{(e\in S)} x_e$	agrégation de la variable x_e sur l'ensemble S.

Pour alléger, on n'écrira pas les accolades si $S = \{i\}$: par exemple, $\delta^+(i)$ au lieu de $\delta^+(\{i\})$.

1.5 Le problème du postier chinois

Le problème du postier chinois (*CPP* ou *Chinese Postman Problem*) est résoluble en temps polynomial si le graphe est orienté ou non orienté. Dans le cas d'un graphe mixte, il devient NP-difficile. Nous rappelons les modèles et méthodes de résolutions pour les versions non orientée (*UCPP* ou *Undirected Chinese Postman Problem*), orientée (*DCPP* ou *Directed Chinese Postman Problem*) et mixte (*MCPP* ou *Mixed Chinese Postman Problem*).

1.5.1 Le problème du postier chinois non orienté (UCPP)

Ce problème consiste à déterminer un cycle de coût total minimal passant au moins une fois par chacune des arêtes d'un graphe non orienté G. Les méthodes de résolution exploitent la remarque suivante : le cycle recherché correspond à un cycle eulérien dans un multigraphe $G' = (V, E')$, comprenant les arêtes originales plus les copies d'arêtes nécessaires pour avoir un graphe eulérien, c'est-à-dire pour rendre pair le degré de chaque nœud. Ainsi, le UCPP revient à construire un graphe eulérien G' minimisant le coût total des copies.

Edmonds et Johnson (1973) ont montré que le problème peut se formuler par le programme linéaire suivant. La variable x_e est le nombre de copies de l'arête e. La fonction-objectif est le coût total du parcours à une constante près (le coût total des arêtes). Les contraintes (1.2) stipulent que le nombre de copies incidentes à un nœud i doit être pair si et seulement si le nœud est pair. Les variables w_i servent à exprimer les sommes modulo 2.

$$\text{Minimiser } \sum_{e\in E} c_e x_e \tag{1.1}$$

$$\forall i \in V : x(\delta(i)) - 2w_i = \begin{cases} 1 \text{ si } |\delta(i)| \text{ impair} \\ 0 \text{ sinon} \end{cases} \tag{1.2}$$

$$\forall e \in E : x_e \in \mathbb{N} \tag{1.3}$$

$$\forall i \in V : w_i \in \mathbb{N} \tag{1.4}$$

Edmonds et Johnson proposent en fait un algorithme travaillant directement sur le graphe. Si G n'est pas eulérien, on calcule un couplage parfait de coût minimal dans un graphe complet formé des nœuds de degrés impairs et dans lequel le coût de l'arête $e = [i, j]$ est égal au coût du plus court chemin entre les nœuds i et j dans G. Ce couplage est calculable par exemple avec l'algorithme de Lawler (1976), de complexité $O(n^3)$ pour un graphe de n nœuds. Ensuite, on rend G eulérien en dupliquant les arêtes le long des plus courts chemins calculés

par le couplage, ce qui donne *G'*. Il suffit enfin d'extraire un parcours eulérien de *G'* avec un algorithme comme celui de Fleury (Kaufmann, 1967). La complexité totale est $O(n^3)$.

1.5.2 Le problème du postier chinois orienté (DCPP)

L'objectif est de déterminer ici un circuit de coût minimal traversant au moins une fois chaque arc d'un graphe orienté *G*. Ce circuit existe si *G* est fortement connexe. Le graphe est eulérien si et seulement s'il est *équilibré*, c'est-à-dire si $b(i) = d^-(i) - d^+(i)$ est nul pour tout nœud *i*. Sinon, il faut ajouter des copies d'arcs pour équilibrer le graphe à moindre coût. Edmonds et Johnson (1973) ont montré que ce problème d'équilibrage peut se modéliser par le problème de flot suivant, dans lequel x_a est le nombre de copies de l'arc *a*.

$$\text{Minimiser } \sum_{a \in A} c_a x_a \tag{1.5}$$

$$\forall i \in V : \sum_{a \in \delta^+(i)} x_a - \sum_{a \in \delta^-(i)} x_a = b(i) \tag{1.6}$$

$$\forall a \in A : x_a \in \mathbb{N} \tag{1.7}$$

Il suffit ensuite d'extraire un circuit eulérien du graphe équilibré avec une version orientée de l'algorithme de Fleury (Christofides, 1975) ou avec la procédure de Van Aardenne-Ehrenfest et De Bruijn (1951). Le problème complet peut être résolu en $O(n^3)$. Lin et Zhao (1988) ont proposé un algorithme qui travaille directement sur le graphe de départ et qui appelle m-n+1 fois une procédure en $O(n^2)$.

En résumé, *UCPP* et DCPP sont des problèmes faciles (polynomiaux). Signalons aussi que la description complète de leur polyèdre est connue. Dans la suite de cette section, on peut voir que ce n'est plus le cas pour tous les autres problèmes de tournées sur arcs.

1.5.3 Le problème du postier chinois mixte (MCPP)

L'objectif du MCPP est de passer au moins une fois à moindre coût sur chaque lien d'un graphe mixte *G*. Ford et Fulkerson ont montré en 1962 la propriété suivante : un graphe mixte et connexe est eulérien si et seulement s'il est *pair* (tous les nœuds ont un degré pair) et *équilibré*. La condition d'équilibrage est plus complexe que dans le cas orienté : pour tout sous-ensemble propre de nœuds *S*, $b(S) = |\delta^-(S)|-|\delta^+(S)|-|\delta(S)|$ doit être nul.

Edmonds et Johnson ont proposé en 1973 des algorithmes pour résoudre le MCPP polynomialement dans deux cas : l) quand le graphe est eulérien et 2) quand il n'est pas équilibré mais pair. A part ces deux cas, le MCPP est NP-difficile (Papadimitriou, 1976). Il est encore relativement peu traité dans la littérature.

Une des premières formulations linéaires pour le MCPP a été donnée par Kapauf et Koehler (1979). La même année, Minieka a proposé un modèle de flots, mais sans résultats numériques. Christofides *et al*. (1984) ont développé une autre formulation avec une variable par arc, deux par arête et une par nœud, résolue grâce à un algorithme combinant branch-and-bound et relaxation lagrangienne. Grötschel et Win (1992) ont utilisé une méthode de coupes conçue à l'origine pour le postier venteux (*cf*. ce problème en 1.7.1).

Un élégant modèle de Nobert et Picard (1996), basé sur la condition de Ford et Fulkerson, mérite d'être détaillé. La variable x_e est le nombre de copies du lien e à ajouter pour rendre G eulérien, $2z_i$ est le degré final du nœud i et $odd(i)$ une constante binaire valant 1 ssi le degré de i est impair. Les contraintes (1.9) et (1.10) traduisent la condition de Ford et Fulkerson.

$$\text{Minimiser} \sum_{e \in U} c_e x_e \tag{1.8}$$

$$\forall i \in V : x(\delta^+(i)) + x(\delta^-(i)) + x(\delta(i)) + odd(i) = 2z_i \tag{1.9}$$

$$\forall S \subset V, S \neq \varnothing : x(\delta^+(S)) - x(\delta^-(S)) + x(\delta(S) \geq b(S) \tag{1.10}$$

$$\forall i \in V, \forall e \in U : x_e, z_i \in \mathbb{N} \tag{1.11}$$

Nobert et Picard ont développé une méthode de coupes basée sur ce modèle. Elle a pu résoudre 148 instances sur un ensemble de 180 tests, avec $10 \leq n \leq 169$, $2 \leq ma \leq 2876$ et $15 \leq me \leq 1849$. Corberan *et al.* (2000a) ont également conçu un algorithme de coupes, développé à l'origine pour le problème de tournées généralisées mixte. D'autres méthodes polyédrales pour le MCPP peuvent être trouvées dans Eglese et Letchford (2000).

Quant aux méthodes heuristiques, elles consistent en général à se ramener aux cas polynomiaux : DCPP, UCPP ou graphe mixte non équilibré mais pair. Frederickson (1979) a ainsi proposé deux heuristiques avec une performance relative au pire égale à 2. Il a prouvé en plus que l'algorithme mixte qui consiste à appliquer les deux heuristiques pour sélectionner la meilleure solution a une garantie de performance de 5/3. Ces heuristiques sont exposées de manière pédagogique par Hertz et Mittaz (2000). Des versions améliorées peuvent être trouvées dans Pearn et Lui (1995, 1999) et dans Pearn et Chou (1999).

La meilleure heuristique semble actuellement une méthode de type *GRASP* (*Greedy Randomized Adaptive Search Procedure*) présentée par Corberan *et al.* (2000c). Elle procède en deux phases. La première est une phase de construction randomisée. Elle commence par orienter toutes les arêtes selon un poids affecté aux deux nœuds aux extrémités de cette arête puis résout un DCPP sur le graphe transformé. La seconde phase tente d'améliorer la solution obtenue. Ces deux phases sont exécutées successivement plusieurs fois. La solution finale est la meilleure trouvée au cours de ces itérations. Les tests ont été réalisés sur des instances générées aléatoirement comprenant jusqu'à 200 nœuds et 600 liens.

1.6 Le problème du postier rural (RPP)

Dans les problèmes de postiers chinois des sections précédentes, tous les liens étaient requis. Dans le problème du postier rural (*RPP* ou *Rural Postman Problem*) introduit par Orloff en 1974, seul un sous-ensemble R de liens requis doit être obligatoirement traversé. Ce problème plus réaliste possède de nombreuses applications comme la collecte des ordures ménagères ou la livraison de lait.

Le problème est polynomial dans trois cas : quand le graphe G est orienté avec $R = A$ (on retrouve un DCPP), quand G est non orienté avec $R = E$ (on retrouve un UCPP) et quand le graphe $G_R = (V_R, R)$ induit par les liens requis est connexe. Sinon, le RPP est NP-difficile dans le cas général (Lenstra et Rinnoy Kan ,1976). La difficulté semble croître avec le nombre p de composantes connexes de G_R.

Partant de nos recherches bibliographiques, nous présentons quelques formulations et algorithmes développés pour le *URPP* (*Undirected Rural Postman Problem*), le *DRPP* (*Directed Rural Postman Problem*) et le *MRPP* (*Mixed Rural Postman Problem*).

1.6.1 Le problème du postier rural non orienté (URPP)

Le premier modèle linéaire publié est celui de Christofides *et al*. (1981). Il suppose un pré-traitement qui convertit tout URPP avec $V_R \neq V$ en une instance avec $V_R = V$. On relie d'abord toute paire de nœuds de V_R par une arête de coût égal au coût du plus court chemin dans G. Ensuite, on conserve une seule arête pour toute paire d'arêtes parallèles de même coût et on efface toute arête non requise $e = [i, j]$ telle qu'il existe un nœud k avec $c_{ij} = c_{ik} + c_{kj}$.

$$\text{Minimiser} \sum_{e \in E} c_e x_e \tag{1.12}$$

$$\forall S \subset V, \delta_R(S) = \varnothing : x(\delta(S)) \geq 2 \tag{1.13}$$

$$\forall i \in V : x(\delta(i)) \equiv \left|\delta_R(i)\right| \bmod 2 \tag{1.14}$$

$$\forall e \in E : x_e \geq 0 \tag{1.15}$$

La variable x_e est le nombre de copies de l'arête e ajoutées pour rendre G eulérien. Les contraintes (1.14) forcent les nœuds à avoir un degré final pair tandis que les contraintes (1.13) obligent le parcours à relier toutes les composantes connexes de G_R.

Peu de méthodes exactes pour le URPP ont été publiées. La première, combinant branch-and-bound et relaxation lagrangienne, est due à Christofides *et al*. (1981). Elle a été testée sur 24 instances générées aléatoirement, avec $9 \leq n \leq 84$, $13 \leq m \leq 184$ et $4 \leq |E_R| \leq 78$. Sanchis (1990) et Corberan et Sanchis (1994) ont développé un branch-and-cut capable de résoudre optimalement 23 instances de Christofides sur 24. Ils ont également résolu 2 cas inspirés par la ville d'Albaida (Espagne), avec $n = 113$, $m = 171$ et $p = 10$ ou 11 composantes.

De nouvelles inégalités valides nommées *path-bridge inequalities* ont ensuite été introduites par Letchford (1996), permettant de résoudre toutes les instances de Corberan et Sanchis optimalement et sans branchement, sauf l'une d'entre elles. Signalons que Corberan *et al.* (1998) ont conçu une puissante méthode de coupes pour le *GRP* (*General Routing Problem*), qui peut traiter le URPP comme cas particulier. Le branch-and-cut le plus récent a été introduite par Ghiani et Laporte (2000). Basé sur un programme linéaire en 0-1, il a pu résoudre des instances aléatoires encore plus grandes, avec $50 \leq n \leq 350$.

A cause de la grande difficulté du URPP, les chercheurs déploient beaucoup d'efforts pour développer des algorithmes approchés. Citons par exemple les heuristiques constructives de Frederickson (1979), de Pearn et Wu (1995) et de Hertz *et al*. (2000), ainsi que les procédures d'amélioration *shorten*, *drop-add* et *2-opt* conçues par Hertz *et al*. (1999). La dernière référence compare des heuristiques initiales et des recherches locales. De cette étude, il ressort que la meilleure combinaison est l'heuristique de Frederickson suivie de la recherche locale 2-*opt*. L'heuristique de Frederickson est particulièrement intéressante : sa garantie de performance vaut 3/2.

Fernandez *et al*. (1998) décrivent une métaheuristique originale, basée sur une méthode de Monte Carlo. Une tournée commence à un nœud choisi au hasard et est prolongée

nœud par nœud suivant certaines probabilités. Si la tournée en construction est parvenue à un nœud *i*, le choix du prochain nœud adjacent *j* dépends du coût de l'arête (*i*, *j*), de l'état de l'arête (requise ou pas) et de la présence ou pas d'arêtes requises dans les arêtes incidentes à *j*. La méthode consiste à générer un grand nombre de tournées et à retenir la meilleure.

Fernandez *et al*. (1998) proposent plusieurs formules de calcul des probabilités de sélection, en général inversement proportionnelles au coût de l'arête. La méthode a été testée sur les 24 instances de Christofides *et al*. (1981) et sur les 2 instances ajoutées par Corberan et Sanchis (1994), dont les solutions optimales sont toutes connues. Les résultats montrent qu'avec l'utilisation de plusieurs paramètres, l'optimum est atteint au moins une fois sur toutes les instances sauf deux ! Avec un seul réglage des paramètres, 19 instances sont résolues optimalement et, dans le pire cas, la solution est à 2 % de l'optimum.

1.6.2 Le problème du postier rural orienté (DRPP)

Voici un exemple de modèle linéaire pour le DRPP, étudié par Christofides *et al*. (1986) puis Ball et Magazine (1988). Comme pour le URPP, la formulation suppose un pré-traitement pour avoir $V_R = V$.

$$\text{Minimiser} \sum_{a \in A} c_a x_a \tag{1.16}$$

$$\forall i \in V : x(\delta^+(i)) + \left|\delta_R^+(i)\right| = x(\delta^-(i)) + \left|\delta_R^-(i)\right| \tag{1.17}$$

$$\forall S \text{ union de composantes de } G_R : x(\delta^+(S)) \geq 1 \tag{1.18}$$

$$\forall a \in A : x_a \in \mathbb{N} \tag{1.19}$$

Christofides *et al*. (1986) ont développé une méthode arborescente. Le polyèdre a été étudié par Savall (1990) puis Guan (1993). Campos et Savall (1995) ont développé un algorithme de coupes qui est capable de résoudre 21 instances parmi 60 problèmes générés aléatoirement, avec 30 composantes connexes, 240 nœuds et 801 arcs.

Les heuristiques disponibles sont soit spécifiques au DRPP (Christofides *et al*., 1986 ; Ball et Magazine, 1988 ; Campos et Savall, 1995) soit obtenues en adaptant des heuristiques initialement conçues pour le URPP (Romero, 1997 ; Corberan *et al*., 2000c ; Mittaz, 1999).

1.6.3 Le problème du postier rural mixte (MRPP)

Notons que le MRPP se réduit au UCPP si $A = \varnothing$ et $R = E$, au DCPP si $E = \varnothing$ et $R = A$ et au *MCPP* si $R = E \cup A$ avec $A \neq \varnothing \neq E$. De plus, évidemment, le MRPP se réduit au URPP si $A = \varnothing$ et $R \subset E$ et au DRPP si $E = \varnothing$ et $R \subset A$. Le MRPP contient donc en cas particulier tous les problèmes de tournées sur arcs sans capacités. Il est donc NP-difficile.

Romero (1997) et Corberan et al. (2000a) ont modélisé le MRPP par le programme linéaire suivant. Il suppose un graphe transformé pour que chaque nœud soit incident à au moins un lien requis et pour que chaque arête soit requise (en remplaçant toute arête non requise par deux arcs opposés). x_e est le nombre de copies du lien *e* ajouté à *G* pour le rendre eulérien. Rappelons que $b_R(S) = |\delta_R^-(S)| - |\delta_R^+(S)| - |\delta_R(S)|$.

$$\text{Minimiser} \sum_{e \in U} c_e x_e \tag{1.20}$$

$$\forall i \in V : x(\delta(i)) + \left|\delta_R(i)\right| + x(\delta^+(i)) + \left|\delta_R^+(i)\right| + x(\delta^-(i)) + \left|\delta_R^-(i)\right| \equiv 0 \bmod 2 \tag{1.21}$$

$$\forall S \subset V : x(\delta^+(S)) - x(\delta^-(S)) + x(\delta(S)) \geq b(S) \tag{1.22}$$

$$\forall S \text{ union de composantes de } G_R : x(\delta^+(S)) \geq 1 \tag{1.23}$$

$$\forall e \in U : x_e \in \mathbb{N} \tag{1.24}$$

La première méthode exacte, basée sur la résolution d'un problème de voyageur de commerce asymétrique, a été élaborée par Laporte (1992a, 1992b). Plusieurs chercheurs se sont ensuite penchés sur une attaque directe, basée sur une étude polyédrale : Corberan *et al.* (1997), Corberan et Sanchis (1994, 1998), Letchford (1997a, 1997b) et Romero (1997). La meilleure méthode de coupes (Corberan *et al.*, 2000a) a été évaluée avec succès sur 100 instances générées aléatoirement, avec $20 \leq n \leq 100$, $15 \leq me \leq 200$, $55 \leq ma \leq 350$ et $p = 15$.

1.7 Autres problèmes sur arcs sans capacité

1.7.1 Le problème du postier venteux (WPP)

Le problème du postier venteux (*WPP* ou *Windy Postman Problem*) a été introduit par Minieka (1979). Comme dans le UCPP, Il s'agit de parcourir au moins une fois les arêtes d'un graphe non orienté mais cette fois le coût de traversée des arêtes dépend du sens. Brucker (1981) et Guan (1984) ont montré que ce problème est NP-difficile mais résoluble en temps polynomial pour un graphe eulérien. Il est intéressant de remarquer que le WPP contient en cas particulier les problèmes du postier chinois orienté, non orienté et mixte.

Soit $G = (V, E)$ le graphe à traiter et $H = (V, A)$ le graphe obtenu en remplaçant chaque arête par deux arcs opposés. Soit x_{ij} une variable indiquant le nombre de fois où l'arc (i, j) est parcouru et rappelons que $\delta(i)$ est l'ensemble des arêtes incidentes au nœud i. Suivant Win (1987) et Grötschel et Win (1992), le WPP peut se formuler de la manière suivante. Les contraintes (1.26) laissent la liberté de parcourir les arêtes dans l'un ou l'autre sens tandis que les contraintes (1.27) assurent la conservation du flot en chaque nœud du graphe. L'objectif est bien évidemment de minimiser la somme des coûts de parcours des arcs choisis.

$$\text{Minimiser} \sum_{(i,j) \in A} (c_{ij} x_{ij} + c_{ji} x_{ji}) \tag{1.25}$$

$$\forall (i, j) \in A : x_{ij} + x_{ji} \geq 1 \tag{1.26}$$

$$\forall i \in V : \sum_{(i,j) \in \delta(i)} (x_{ij} - x_{ji}) = 0 \tag{1.27}$$

$$\forall (i, j) \in A : x_{ij}, x_{ji} \in \mathbb{N} \tag{1.28}$$

Différents auteurs dont Grötschel et Win (1988, 1992), Win (1987, 1989) et Gendreau *et al.* (1990) ont étudié la structure du polytope du WPP et ont proposé des inégalités valides utilisables dans des algorithmes de coupes. L'un de ces algorithmes (Grötschel et Win, 1992) a été testé sur 36 instances générées aléatoirement avec $52 \leq n \leq 264$ et $78 \leq m \leq 489$. Il en résoud 31 optimalement.

1.7.2 Le problème du postier rural venteux (WRPP)

Le problème du postier rural venteux (*WRPP* ou *Windy Rural Postman Problem*) est similaire au précédent sauf qu'ici seul un sous-ensemble d'arêtes E_R est à traiter. Son étude est toute récente (Benavent *et al.*, 2002a). La formulation utilisée généralise celle donnée pour le WPP par Grötschel et Win (1992). Les auteurs proposent des inégalités valides, un algorithme de coupes et trois heuristiques.

Les trois heuristiques ont été améliorées par Benavent *et al.* (2002b). Dans ce dernier article, les auteurs décrivent un algorithme de type *GRASP* qui consiste d'abord à générer aléatoirement plusieurs solutions en introduisant un facteur aléatoire dans les trois heuristiques de Benavent *et al.* (2002a). Ensuite, chaque solution est codée de façon compacte pour subir une recherche locale combinant des voisinages de type 2-*opt* et *or-opt*.

Une autre façon d'améliorer la solution consiste à calculer, pour une séquence donnée d'arêtes, le meilleure sens de traversée pour chaque arête. Ce sous-problème est résoluble en temps polynomial. Dans la même publication, les auteurs utilisent cette technique dans une méthode de recherche distribuée (*scatter search*), qui semble actuellement l'algorithme le plus efficace pour le WRPP.

1.7.3 Le problème du postier hiérarchique (HPP)

Dans le postier hiérarchique (*Hierarchical Postman Problem* ou *HPP*), les liens d'un graphe mixte sont partitionnés en k classes $\{A_1,\ldots, A_k\}$ munies d'une relation d'ordre $<$. Le graphe possède également une source s et un puits t. Le parcours doit traiter les liens d'une classe A_p avant ceux d'une classe A_q si $A_p < A_q$. Ce problème est NP-difficile. On le rencontre dans le déneigement de réseaux routiers, les classes correspondant à des catégories de routes avec différentes priorités (autoroutes puis routes nationales par exemple).

Il a été formulé pour la première fois par Dror *et al.* (1987), qui ont même identifié un cas particulier polynomial quand les trois conditions suivantes sont vérifiées : a) G est complètement orienté ou bien non orienté, b) la relation de précédence est complète (*linear ordering*) et c) chaque classe forme une composante connexe. Un algorithme pour ce cas polynomial a été proposé par Ghiani et Improta (2000a).

Une méthode exacte basée sur la programmation dynamique a été développée par Gélinas (1992) et testée sur des problèmes avec $n = 16$, $m = 48$, et $k = 1, 2, 3, 4, 6, 8$ et 10. Un problème réel de déneigement pour Montréal (avec $n = 171$, $m = 544$, et $k = 4$) a été modélisé sous forme d'un HPP et résolu optimalement par Campbell et Langevin en 1999.

1.7.4 Le problème de la grue (Stacker Crane Problem ou SCP)

Il s'agit cette fois d'un réseau non orienté dans lequel il faut traverser au moins une fois chaque arête mais dans un sens imposé, un peu comme une grue ayant à déplacer un certain nombre de cargaisons, d'où le nom. Ce problème peut être modélisé par un graphe mixte $G' = (V, E \cup A)$ dans lequel l'ensemble A contient une copie de chaque arête du graphe

initial, orientée selon le sens imposé. Le but est de parcourir au moins une fois chaque arc de *A*. Ce problème est également NP-difficile (Frederickson *et al.* 1978). C'est en fait un cas particulier du problème du postier rural. Lukka et Salminen (1987) ont adapté une heuristique d'insertion, développée initialement pour le *TSP*, pour résoudre ce problème.

1.7.5 Autres variantes plus exotiques

Il existe des versions du *CPP* avec un nombre fixe de postiers : le *MM k-CPP* (*Min-Max k-Chinese Postman Problem*) et le *M k-CPP* (*Mixed k-Chinese Postman Problem*). Dans ces problèmes, une solution réalisable est un ensemble de *k* tournées commençant et finissant au dépôt et couvrant tous les liens. Dans le MM k-CPP, l'objectif est de minimiser le coût de la plus longue tournée, dans un graphe non orienté. Une heuristique a été proposée par Frederickson *et al*. (1978). Dans le M k-CPP, l'objectif est de minimiser le coût total des tournées, dans un graphe mixte. Pearn (1994) et Zhang (1992) ont mis en évidence quelque cas particuliers polynomiaux.

En 2000, Dror et Houari ont défini un problème NP-difficile appelé problème du postier chinois généralisé (*GCPP* ou *Generalized Chinese Postman Problem*), avec un graphe non orienté dans lequel les arêtes sont partitionnées en sous-ensembles. L'objectif est de trouver un ensemble de tournées qui traversent au moins une arête de chaque sous-ensemble.

Une généralisation du URPP appelée *RPPDC* (*Rural Postman Problem with Deadline Classes*) a été étudiée par Letchford et Eglese (1998). L'ensemble *R* est partitionné en classes $R_1, \ldots, R_k$, chacune avec une heure-limite de fin de traitement. L'objectif est de trouver une tournée traversant au moins une fois chaque arête requise et respectant la deadline de chaque classe. Comme le HPP, le problème se pose dans le déneigement de réseaux routiers.

1.8 Le problème de tournées sur arcs avec contrainte de capacité

1.8.1 Présentation du problème

Nous présentons à partir de maintenant les problèmes de tournées sur arcs dits *avec capacités* ou *multi-véhicules*. Cette fois, des quantités (à livrer ou à collecter) sont associées aux liens du graphe. La capacité limitée des véhicules oblige à construire plusieurs tournées. Le problème de base de ce type est le *CARP* (*Capacitated Arc Routing Problem*).

Le CARP de base est défini dans la littérature sur un graphe non orienté valué $G = (V, E, C)$. L'ensemble des nœuds comprend un dépôt *s* où est basée une flotte homogène de *K* véhicules identiques de capacité *W*, en nombre suffisant. Le nombre de véhicules *K* peut être imposé ou être une variable de décision. Un sous-ensemble d'arêtes *R* doit être obligatoirement traversé pour être traité. La traversée (avec ou sans traitement) d'une arête $[i, j]$ par un véhicule coûte c_{ij} et chaque arête de *R* a une demande r_{ij}.

Le CARP consiste à déterminer un ensemble de tournées de coût total minimal, tel que : a) chaque tournée est assurée par un véhicule qui part et revient au dépôt, b) la somme des demandes traitées par une tournée ne dépasse pas *W* et c) chaque arête de *R* est effectivement traitée, par une seule tournée et lors d'un seul passage (pas de traitement partiel).

Le CARP orienté ou non est NP-difficile (réduction du problème de Bin-Packing, *cf.* Dror, 2000). Signalons que le cas $R = E$ a été introduit dès 1973 par Christofides, sous le nom de *CCPP* (*Capacitated Chinese Postman Problem*).

1.8.2 Formulations par programmes linéaires

1.8.2.1 Formulation de base

Nous rappelons ici la première formulation pour le CARP, introduite par Golden et Wong (1981). Elle concerne un graphe non orienté. Les nœuds sont indicés de 1 (le dépôt) à *n*. Soit *A* l'ensemble des arcs obtenus en remplaçant chaque arête par deux arcs opposés, x_{ijk} une variable binaire égale à 1 ssi le véhicule *k* traverse l'arête [*i, j*] de *i* vers *j* et l_{ijk} une variable binaire valant 1 ssi le véhicule *k* traite [*i, j*] de *i* vers *j*.

$$\text{Minimiser} \sum_{k\in I}\sum_{(i,j)\in A} c_{ij}x_{ijk} \tag{1.29}$$

$$\forall i \in V, \forall k \in I : \sum_{j\in\Gamma(i)} (x_{ijk} - x_{jik}) = 0 \tag{1.30}$$

$$\forall (i, j) \in A, \forall k \in I : x_{ijk} \geq l_{ijk} \tag{1.31}$$

$$\forall (i, j) \in R : \sum_{k\in I} (l_{ijk} + l_{jik}) = 1 \tag{1.32}$$

$$\forall k \in I : \sum_{(i,j)\in A} l_{ijk} r_{ij} \leq W \tag{1.33}$$

$$\forall S \neq \varnothing, S \subset \{2,...,n\} \text{ et } \forall k \in I : \begin{cases} \sum_{i,j\in S} x_{ijk} - n^2 y_s^k \leq |S| - 1 \\ \sum_{i\in S}\sum_{j\in\bar{S}} x_{ijk} + u_s^k \geq 1 \\ u_s^k + y_s^k \leq 1 \end{cases} \tag{1.34}$$

$$u_s^k, y_s^k, x_{ijk}, l_{ijk} \in \{0,1\} \tag{1.35}$$

L'objectif (1.29) consiste à minimiser la somme des coûts de parcours. Les contraintes (1.30) garantissent qu'un véhicule qui arrive en un nœud doit en repartir. Les contraintes (1.31) garantissent qu'un arc traité est aussi traversé. Les contraintes (1.32) imposent que chaque arête requise soit traitée, par un seul véhicule et dans un seul sens. Les contraintes (1.33) assurent le respect de la capacité des véhicules. Enfin l'interdiction des sous-tours illégaux est imposée par les contraintes (1.34) qui nécessitent des variables binaires supplémentaires y_s^k et u_s^k.

Cette formulation du CARP utilise trop de variables : la résolution exacte avec des solveurs commerciaux ne sera possible que pour de petites instances. Les principales avancées en termes de modélisation sont dues à Belenguer et Benavent (1992, 1994, 1998) qui ont travaillé sur deux formulations plus compactes, dites *sparse* et *supersparse*, que nous présentons dans les deux sous-sections suivantes.

1.8.2.2 La formulation sparse

Cette formulation est plus compacte que la précédente. Elle utilise $K(m + |R|)$ variables. La variable binaire x_{ke} vaut 1 ssi le véhicule k traite l'arête requise e et la variable entière y_{ke} est le nombre de fois où k traverse e sans la traiter.

$$\text{Minimiser } \sum_{k\in I}\sum_{e\in E_R} c_e x_{ke} + \sum_{k\in I}\sum_{e\in E} c_e y_{ke} \tag{1.36}$$

$$\forall e \in E_R : \sum_{k\in I} x_{ke} = 1 \tag{1.37}$$

$$\forall k \in I : \sum_{e\in E_R} r_e x_{ke} \le W \tag{1.38}$$

$$\forall S \subseteq V - \{1\}, \forall f \in E_R(S), \forall k \in I : x_k(\delta_R(S)) + y_k(\delta(S)) \ge 2x_{kf} \tag{1.39}$$

$$\forall S \subseteq V - \{1\}, \forall k \in I : x_k(\delta_R(S)) + y_k(\delta(S)) \equiv 0 \bmod 2 \tag{1.40}$$

$$x_{ke} \in \{0,1\}, y_{ke} \in \mathbb{N} \tag{1.41}$$

Les contraintes (1.37) et (1.38) assurent respectivement que chaque arête est bien traitée et que la capacité des véhicules est respectée. Les contraintes de connectivité (1.39) stipulent que, si une tournée k traite une arête e, alors elle doit traverser toute coupe séparant e du dépôt. Les contraintes (1.40) imposent un degré pair à chaque nœud.

1.8.2.3 La formulation supersparse

Cette formulation ne nécessite qu'une seule variable z_e par arête, représentant le nombre total de traversées sans traitement. Avec les notations précédentes, la formulation supersparse pour le CARP est la suivante :

$$\text{Minimiser } \sum_{e\in E} c_e z_e \tag{1.42}$$

$$\forall S \subseteq V - \{1\} : z(\delta(S)) \ge 2K(S) - |\delta_R(S)| \tag{1.43}$$

$$\forall S \subseteq V - \{1\}, |\delta_R(S)| \text{ pair} : z(\delta(S)) \ge 1 \tag{1.44}$$

$$\forall e \in E : z_e \in \mathbb{N} \tag{1.45}$$

Les contraintes (1.43) sont des contraintes de capacité signifiant qu'au moins $K(S)$ véhicules doivent traverser les arêtes de la coupe $\delta(S)$ pour traiter la demande totale de l'ensemble $E_R(S) \cup \delta_R(S)$. Notons que le nombre de véhicules est non fixé. Les contraintes de parité sont exprimées par (1.44). Ce modèle ne prend pas en compte les chemins utilisés pour aller traiter un ensemble d'arêtes d'une coupe donnée. De ce fait, il n'est pas complet et il peut donner des solutions entières non faisables pour le CARP. Cependant, ce modèle utilise seulement m variables et permet d'avoir les meilleures bornes connues pour le CARP (voir la section 1.8.4.6).

Pour plus de détails sur les différentes formulations proposées pour le CARP, nous recommandons l'état de l'art très récent dans le livre coordonné par Dror (2000).

1.8.3 Méthodes optimales

Les seules méthodes exactes développées pour le CARP sont les méthodes arborescentes (*branch-and-bound*) de Hirabayashi *et al.* (1992) et de Kiuchi *et al. (*1995), qui appliquent en chaque nœud de l'arborescence une procédure nommée *Node Duplication Lower Bound* (*NDLB*). Seules des petites instances ont été résolues par ces méthodes. Une autre tentative de résolution exacte a été proposée par Welz (1994). Basée sur une formulation sparse différente de celle de Benavent, elle utilise un *branch-and-bound* avec relaxation des contraintes de connectivité et de parité. Welz a pu ainsi résoudre des instances, avec $k = 2$, $n = 27$, $m = 82$; $k = 3$, $n = 16$, $m = 48$ et $k = 4$, $n = 12$, $m = 50$.

Une autre technique appelée *branch-and-cut*, qui combine *branch-and-bound* et coupes, a été appliquée au CARP dans les travaux de Belenguer et Benavent (1992, 1994, 1998, 2002). Des inégalités valides ainsi que des procédures de séparation ont été définies. Dans cette méthode, Belenguer et Benavent utilisent la formulation sparse avec les contraintes de capacité et de parité de la formulation supersparse. 16 sur les 24 instances de Benavent *et al*. (1992) sont résolues de façon optimale. Dans le cadre des études polyèdrales pour le CARP, nous pouvons citer aussi les synthèses de Eglese et Letchford (2000) et de Benavent *et al.* (2000).

1.8.4 Bornes inférieures

Différentes bornes inférieures ont été proposées par Golden et Wong (1981), Assad *et al*. (1987), Pearn (1988) et Win (1987). La plupart de ces bornes ont été améliorées par Benavent *et al*. (1992).

La première borne inférieure proposée pour le CARP est celle du postier chinois non orienté, $LBCPP = C(R) + C_{euler}$. $C(R)$ est le coût total des arêtes à traiter et C_{euler} le coût des arêtes ajoutées pour rendre le graphe eulérien. De cette idée de base découlent plusieurs autres bornes, présentées ci-après.

1.8.4.1 Matching Lower Bound (MLB)

La borne du postier chinois peut être raffinée car on sait qu'il y aura au minimum K^* tournées (ou véhicules). En plus de l'ajout des arêtes pour rendre le graphe eulérien utilisé dans le calcul de LBCPP, Christofides (1973) a proposé d'intégrer aussi des copies du dépôt en nombre suffisant pour satisfaire la contrainte sur le nombre minimum de véhicules. Soit C_{vehi} le coût des copies d'arêtes incidentes au dépôt ajoutées pour satisfaire cette contrainte. La borne de Christofides se calcule comme suit :

$$MLB = C(R) + C_{euler} + C_{vehi} \tag{1.46}$$

En 1981, Golden et Wong ont constaté que le calcul de C_{euler} et C_{vehi} ne peut se faire de façon indépendante. Ils ont proposé de calculer un couplage de coût minimal dans le graphe des nœuds de degrés impairs avec q copies du dépôt : $q = 2K^* - d(1)$ si $d(1)$ est impair, sinon $q = 2K^* - d(1) - 1$. Dans le cas ou $q > 0$, le calcul du couplage se fait dans un graphe

complet $\overline{H}$ incluant non seulement les nœuds de degré impair V_{odd} mais aussi deux ensembles *A* et *B*. Chacun des deux ensembles contient *q* copies du dépôts. Le coût des arêtes [*i*, *j*] ayant les deux extrémités dans V_{odd} ou une extrémité dans V_{odd} et une autre dans *A* est égale au coût du plus court chemin de *i* à *j*, nommé s_{ij}. Toutes les arêtes ayant une extrémité dans *A* et une autre dans *B* ont le même coût, c'est la valeur du coût minimal des arêtes incidentes au dépôt. Enfin, les arêtes ayant les deux extrémités dans *B* ont un coût égal à 0. Donc, si $MP(\overline{H})$ est la valeur du couplage de coût minimal dans $\overline{H}$, alors (1.47) définit une borne pour le CARP:

$$MLB = C(R) + MP(\overline{H}) \tag{1.47}$$

1.8.4.2 Node Scanning Lower Bound (NSLB)

Assad *et al*. (1987) ont observé que si $q > 0$, le véhicule emprunte des chemins vers des arêtes à traiter et donc le nombre de chemins se terminant à un nœud *i* ne peut pas être plus grand que le nombre d'arêtes requises incidentes à ce nœud. Cette idée est la base de la borne *NSLB*. Soit $d(i, R) = |\delta_R(S)|$ le nombre d'arêtes requises incidentes à *i*. Sachant que les nœuds de G_R sont ordonnés par distance croissante au dépôt, qu'ils sont renumérotés selon cet ordre et que *i** est le plus petit entier tel que $d(2, R) + d(3, R) + \ldots + d(i^*, R) \geq q$, alors le calcul de NSLB est comme suit :

$$NSLB = C(R) + s_{12} \times d(2, R) + \ldots + s_{1i^*} \times d(i^*, R) \tag{1.48}$$

1.8.4.3 Matching Path Lower Bound (MPLB)

Pearn (1988) a combiné les idées des deux bornes précédentes pour proposer sa borne *MPLB*. Soit *q* un nombre pair, tel que $0 \leq q \leq |V_{odd}|$. La détermination de la borne se fait selon deux phases. Dans la première phase, un problème de couplage est résolu dans $V_{odd} \cup A(q)$, tel que $A(q)$ contienne *q* copies du dépôt. Les coûts des arêtes dans le graphe de couplage correspondent aux plus courts chemins dans le graphe original sauf pour celles de *A(q)* qui ont un coût égal à l'infini. Soit $MP(V_{odd} \cup A(q))$ la valeur du couplage obtenue. Les copies d'arêtes qui correspondent à la solution du couplage sont ajoutées et le nouveau degré du nœud dépôt est $\overline{d}(1)$. La deuxième phase procède de la même façon que pour le calcul de NSLB proposé par Assad *et al*. (1987). D'où la borne MPLB :

$$MPLB = \min\{LB(q),\ q = 0, 2, 4, \ldots, |V_{odd}|\} \tag{1.49}$$

Avec :

$$LB(q) = C(R) + Mp(V_{odd} \cup A(q)) + \sum_{i=2}^{r} s_{1i} \times d(i, R) \tag{1.50}$$

$$r = \min\left\{j : d(2, R) + \ldots + d(j, R) \geq 2K^* - \overline{d}(1)\right\} \tag{1.51}$$

1.8.4.4 Successive Cut Lower Bound (SCLB)

Win (1987) a proposé de prendre en compte une succession de coupes au lieu de considérer uniquement la coupe au dépôt comme dans les procédures précédentes. En fait c'est une généralisation de la borne MLB donnée par Golden et Wong (1981).

1.8.4.5 Les bornes : LB1, LB2, LB3 et LB4

Benavent *et al.* (1992) ont amélioré toutes les bornes précédentes en proposant 4 nouvelles bornes nommées *LB*1, *LB*2, *LB*3 et *LB*4. Les trois premières font appel à la résolution d'un problème de couplage, quant à la dernière elle utilise la programmation dynamique. *LB*3 et *LB*4 prennent en compte la contrainte d'une flotte limitée de véhicule.

Benavent *et al.* (1992) ont montré que leur borne *LB*1 est équivalente à une autre borne nommée *Node duplicated lower bound* (*NDLB)* qui a été proposée par Saruwatari *et al.* (1992) mais que leur borne à l'avantage d'être plus rapide et plus facile à mettre en œuvre que NDLB.

1.8.4.6 Borne basée sur la méthode de coupes

Belenguer et Benavent (1998) ont présenté des inégalités valides qui induisent des facettes du polytope du CARP. Certaines de ces facettes proviennent directement de l'étude du problème d'affectation généralisée (*GAP* ou *General Assignment Problem*). Les auteurs proposent d'utiliser ces inégalités dans un algorithme de coupes générant ainsi des bornes inférieures généralement de meilleure qualité que celles connues jusque là et ceci dans des temps de calcul relativement courts. En 2002, Belenguer et Benavent ont développé de nouvelles inégalités valides nommées *DP*1, *DP*2, *DP*3 (*DP* pour *Disjoint-Path)*, également utilisées dans un algorithme de coupes. Leur approche de résolution commence par une relaxation des contraintes (1.43) dans la formulation supersparse du CARP pour ensuite ajouter toutes les contraintes violées dans la solution du programme linéaire. Il s'agit des contraintes de capacité, de parité et des contraintes DP1, DP2, DP3.

Les inégalités de type *Disjoint-Path* améliorent de manière considérable la qualité des bornes inférieures prouvant ainsi que pour 47 des 87 instances de la littérature testées, les solutions trouvées auparavant par des méthodes heuristiques étaient optimales. La plus grande instance résolue comporte 50 nœuds et 97 arêtes. Les résultats obtenus peuvent être détaillés dans la table 1.1. La première colonne indique l'origine des instances, la deuxième donne le nombre de problème par instance et les trois dernières représentent respectivement le nombre d'optimums prouvés, l'écart moyen et le pire écart par rapport à la meilleure solution connue pour le CARP.

Ensemble d'instances	*Nbr-prob*	*Taille des problèmes*	*Nbr-opt*	*Ecart moyen* (%)	*Pire écart* (%)
Benavent *et al*.(1992)	34	$24 \leq n \leq 50$, $35 \leq \|R\| \leq 97$	22	0.41	3.13
Golden *et al*. (1983)	23	$7 \leq n \leq 27$, $35 \leq \|R\| \leq 55$	20	0.14	1.75
Kiuchi *et al*. (1995)	6	$6 \leq n \leq 10$, $\|R\| = 15$	5	0.58	3.48
Li et Eglese (1996)	24	$77 \leq n \leq 140$, $51 \leq \|R\| \leq 190$	0	2.40	4.27

Table 1.1 – *Résultats de la meilleure borne pour le CARP sur les instances de la littérature*

1.8.4.7 Hierarchical Relaxations Lower Bound (HRLB)

Partant de l'idée que le dépôt doit être traversé un certain nombre de fois, Amberg et Voβ (2002) ont développé récemment le concept de borne inférieure hiérarchique. Ils génèrent des relaxations hiérarchiques RL_1, RL_2,…, RL_i. RL_i considère toutes les contraintes de RL_{i-1}. A chaque itération une contrainte de capacité est ajoutée. Les résultats sur les instances de la littérature non encore résolues montrent l'efficacité de cette méthode. Les auteurs précisent que leurs bornes peuvent être adaptées au CARP avec une flotte hétérogène et plusieurs dépôts.

1.8.5 Heuristiques constructives

Comme pour tous les problèmes NP-difficile, la recherche d'heuristiques performantes est l'une des principales préoccupations des spécialistes du domaine. Le principe d'une méthode heuristique est de trouver, en un temps raisonnable *une* solution de bonne qualité. Dans ce qui suit, nous décrirons les principales heuristiques connues pour le CARP. Nous reviendrons en détail sur certaines d'entre elles dans le chapitre 3 (*Path-Scanning*, *Augment-Merge* et l'heuristique d'*Ulusoy*).

1.8.5.1 Construct-Strike

Cette méthode a été proposée par Christofides (1973), puis améliorée par Pearn (1989) dans sa version *Modified Construct-Strike*. Elle fonctionne en construisant progressivement des cycles qu'elle enlève du graphe seulement si ce dernier reste connexe. La méthode d'origine à une complexité en $O(n^3.m)$ et est à 17.9 % en moyenne de la borne inférieure max (MLB, NSLB). La version de Pearn est plus coûteuse, en $O(n^4.m)$ mais donne une meilleure déviation de 7.6%. C'est actuellement la meilleure heuristique constructive pour le CARP dans le cas où le graphe est dense et où les demandes sont relativement petites.

1.8.5.2 Path-Scanning

Golden *et al.* (1983) ont développé une méthode gloutonne qui construit des tournées une par une. Chaque tournée part du dépôt et est prolongée à chaque itération vers une arête à traiter compatible avec la capacité résiduelle du véhicule. Pour une tournée parvenue en un nœud *i*, on donne la priorité aux arêtes non encore traitées partant de *i*. S'il n'y en a plus, on examine le plus proche nœud ayant des arêtes non encore traitées compatibles. Les arêtes possibles [*i, j*] sont départagées par un des cinq critères suivants. L'algorithme est exécuté 5 fois (une fois par critère) et donne à la fin la meilleure solution.

Critère 1 : maximiser la distance de *j* au dépôt ;
Critère 2 : minimiser cette distance ;
Critère 3 : minimiser la productivité (r_{ij}/c_{ij}) ;
Critère 4 : maximiser la productivité (r_{ij}/c_{ij}) ;
Critère 5 : égale au critère 1 si $load \leq W/2$ et au critère 2 sinon (*load* est la demande totale traitée par la tournée en cour de construction).

Une version modifiée par Pearn (1989) appelée *Random Path Scanning* sélectionne les arêtes selon un choix aléatoire d'un critère parmi les cinq décrits précédemment. L'algorithme est exécuté *k* fois avec à chaque fois des probabilités de sélection différentes pour les 5 critères. La complexité est en $O(n^3)$. La version modifiée est à 8 % de la borne max(*MLB*, *NSLB*) comparée à la méthode originale qui est de 11%.

1.8.5.3 Augment-Merge

Cette méthode a également été proposée par Golden *et al.* (1983). Elle s'inspire de la méthode de Clark et Wright (1964) pour les problèmes de tournées sur nœuds. On part d'une solution triviale de $|R|$ tournées : une tournée par arête à traiter. Le nombre de véhicules utilisés est progressivement réduit grâce à deux phases *Augment* et *Merge*.

Dans la phase *Augment*, les auteurs exploitent le fait que les plus courts chemins d'une tournée peuvent passer sur l'arête à traiter d'une tournée plus petite. La 1ère tournée peut donc absorber la petite sans augmentation de coût. Le coût de la solution diminue du coût de la tournée absorbée. Pour de meilleurs résultats, les auteurs préconisent de trier les tournées initiales par coûts décroissants puis, pour toute tournée prise dans l'ordre du tri et si cette tournée passe sur l'unique arête à traiter d'une autre tournée, alors cette arête est absorbée si la capacité du véhicule le permet.

Dans la phase *Merge*, on effectue tant que c'est possible des *fusions* de deux tournées si cela diminue le coût total. Pour deux tournées, il y a quatre concaténations possibles : il y a deux ordres possibles de concaténation et deux sens de parcours des tournées. A chaque itération, la concaténation donnant un gain maximal est effectuée. Cette méthode a une complexité en $O(n^3)$.

1.8.5.4 Parallel-Insert

Cette méthode développée par Chapleau *et al*. (1984) se base sur l'idée de *Path-Scanning*. Elle était initialement destinée pour la construction des itinéraires de bus pour le ramassage scolaire avec comme contrainte supplémentaire une limite sur le temps total de travail ainsi qu'un nombre fixe de véhicules à ne pas dépasser.

La méthode commence par construire une tournée avec l'arête requise la plus éloignée du dépôt. Les arêtes requises restantes sont ensuite insérées en utilisant deux stratégies : la première choisit, pour une arête donnée, la tournée qui va l'accueillir ainsi que la position d'insertion dans cette tournée, en respectant les contraintes de capacité et d'autonomie. Dans le cas où l'insertion n'est pas possible dans les tournées existantes, une nouvelle tournée avec cette unique arête est crée. La deuxième stratégie est appliquée quand tous les véhicules sont utilisés. Elle consiste à parcourir les tournées par ordre décroissant de charge et, pour une tournée donnée, à choisir l'arête requise restante réalisant la meilleure insertion.

1.8.5.5 Augment-Insert

Il s'agit d'une extension en $O(n^3)$ des algorithmes *Augment-Merge* et *Parallel-Insert*, proposée par Pearn (1991). La méthode procède en deux phases. Dans la première, des tournées faisables sont générées en utilisant la phase augment de *Augment-Merge*. En fin de cette phase, les tournées ayant une seule arête requise seront ignorées et leurs arêtes seront réinsérées par la stratégie d'insertion de *Parallel-Insert*. Cette heuristique donne de meilleurs résultats sur les graphes peu denses avec des demandes relativement grandes.

1.8.5.6 Méthodes de type Cluster-First, Route-Second

Ce genre de méthode commence par partitionner les arêtes requises entre les véhicules, puis construit une tournée pour chaque véhicule. Benavent *et al*. (1990) ont développé une heuristique de ce type. La première phase appelée *Cycle-Assignement* revient à résoudre un problème d'affectation généralisé. Elle s'inspire de l'algorithme proposé par Fisher et Jaikumar (1981) pour le VRP. La deuxième phase résoud heuristiquement les problèmes de postier rural obtenus.

1.8.5.7 Méthodes de type Route-First, Cluster-Second

Contrairement aux méthodes Cluster-First, Route-Second, ce type d'heuristique consiste à résoudre un problème de postier rural sur l'ensemble des arêtes à traiter, en relaxant la contrainte de capacité des véhicules, puis de découper le tour géant ainsi obtenu en tournées plus petites réalisables pour le CARP. Ulusoy (1985) et Hertz *et al*. (2000) ont proposé des heuristiques de ce genre.

1.8.6 Métaheuristiques

Contrairement aux méthodes constructives dont l'objectif est de construire une seule solution, les méthodes de recherche locale améliorent progressivement une solution initiale, obtenue par exemple avec une heuristique constructive. Chaque itération d'une recherche locale consiste à explorer un voisinage de la solution actuelle pour trouver une meilleure solution. Selon les cas, il peut s'agir de la première amélioration trouvée ou de la meilleure. La recherche s'arrête si on ne trouve pas de meilleures solutions : la solution actuelle est donc localement optimale (au sens du voisinage). Dans le cas contraire, la nouvelle solution devient la solution courante et on recommence.

On appelle métaheuristiques (du grec, *meta* = qui englobe) des méthodes conçues pour échapper aux minima locaux. Le terme *meta* s'explique aussi par le fait que ces méthodes sont des structures générales dont il faut instancier les composants en fonction du problème (par exemple, le voisinage, les solutions de départ ou les critères d'arrêt).

Les métaheuristiques font appelle à la notion de voisinage. On entend par voisinage d'une solution toute solution proche suivant un certain critère et créée par une ou plusieurs modifications sur la solution en question. Par exemple, si S est une solution réalisable du CARP, un voisinage possible $N(S)$ de S contient l'ensemble des solutions que l'on peut obtenir à partir de S en déplaçant le traitement d'une arête d'un véhicule à un autre.

Nous présentons dans cette section les métaheuristiques qui ont été appliquées au CARP : la méthode tabou, le recuit simulé et la recherche à voisinage variable.

1.8.6.1 La méthode tabou

Cette méthode a été élaborée par Glover (1989, 1990). Elle est basée sur la notion de mouvements interdits (tabou). La recherche tabou examine les solutions voisines de la solution courante et choisit la plus proche de celle-ci. Pour éviter un comportement cyclique de la méthode, les solutions qui ont été récemment examinées sont classées tabou, c'est-à-dire interdites pendant un certain nombre d'itérations. En pratique, ce sont des attributs d'une solution plutôt que la solution entière qui sont classées tabou.

Récemment les méthodes tabou ont été développées pour des variantes du CARP et pour le CARP lui-même par Eglese et Li (1996), Belenguer *et al.* (1997) et Hertz *et al.* (2000).

Greistorfer (2003) a appliqué une méthode tabou hybride qui utilise le principe du *Scatter-Search* (*TSCS* pour *Tabu Scatter Search*) pour résoudre le CCPP. L'algorithme maintient une population de bonnes solutions bien dispersées trouvées au cour de la recherche tabou. Ces solutions sont combinées entre elles pour générer d'autres points de départ pour le tabou. La méthode a été testée sur des instances générées aléatoirement et sur les instances de DeArmon (1981) (*cf.* table 1.1).

Hertz *et al*. (2000) ont proposé une méthode tabou *Carpet* qui donne d'excellents résultats sur toutes les instances de la littérature. Elle est basée sur des voisinages appelés *Cut*, *Switch*, *Postopt*, *Shorten*, *Drop*, *Add* et *Drop-Add*. Nous donnons ci-dessous son principe général ainsi qu'un de ces voisinages le plus sophistiqué, *Shorten*.

a) Shorten

Contrairement aux autres voisinages appliqués au problème de tournées sur arcs, qui utilisent le principe des méthodes déjà proposées pour les tournées sur nœuds, *Shorten* est une nouvelle technique de recherche locale très sophistiquée développée spécialement pour les problèmes de tournées sur arcs. Elle exploite la structure souvent en *spaghettis* des tournées sur arcs.

Cette procédure de recherche locale tente de réduire la longueur d'une tournée sans modifier l'ensemble des arêtes traitées. Premièrement, elle consiste à remettre le traitement de chaque arête requise à sa dernière apparition dans la tournée (si elle apparaît plusieurs fois). Deuxièmement, tous les circuits du parcours qui commencent par une arête requise [*i*, *j*] et qui finissent par une arête non à traiter [*k*, *i*] sont inversés. Enfin, *Shorten* supprime tous les cycles inutiles n'ayant aucune arête requise (*cf.* figure 1.1).

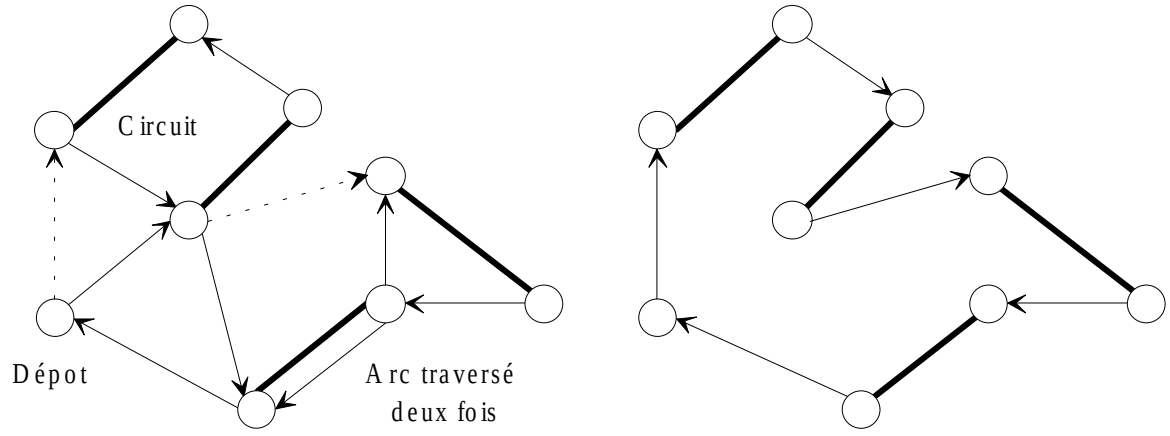


Figure 1.1 – *Illustration de Shorten*

b) Carpet

Soit x une solution du CARP, $g(x)$ le coût total de la solution, $E(x)$ la somme des violations de capacité et β un paramètre de pénalité pour la violation de la contrainte de capacité.

Carpet consiste d'abord à résoudre un RPP par l'heuristique de Frederickson (1979), en relaxant la capacité des véhicules, puis à construire une solution du CARP par la technique de décomposition *Cut* et enfin, à appliquer les procédures *Switch* et *Postopt.* La procédure *Postopt,* fait appel à la recherche locale *Shorten* qui améliore considérablement la solution. D'autres voisinages initialement conçus pour le RPP comme *Drop*, *Add* et *Drop-Add* sont aussi utilisées.

Carpet minimise la fonction objectif $f(x) = g(x) + \beta \times E(x)$. Le paramètre β est réglé automatiquement : initialement égal à 1, il est divisé par 2 après λ itérations faisables, sinon il est doublé après λ itérations infaisables. L'algorithme proposé surpasse en qualité toutes les méthodes connues (l'optimum étant trouvé sur plusieurs jeux de données de la littérature). C'est pourquoi, tout au long de cette thèse, nous utilisons *Carpet* comme méthode de référence.

1.8.6.2 Le recuit simulé

Le recuit simulé est basé sur un algorithme de simulation de recuit des métaux et s'inspire des modèles de la physique statistique (Metropolis *et al.*, 1953). Il a été développé par Kirkpatrick (1983). Afin de s'échapper à de possibles optima locaux, le recuit simulé permet de retenir d'une manière contrôlée des solutions diminuant la valeur de la fonction objectif.

La procédure générale du recuit simulé est la suivante. Soit S la solution courante et $N(S)$ le voisinage de S. Une solution $S' \in N(S)$ est générée aléatoirement, et la différence des valeurs de la fonction objectif $\Delta f = f(S') - f(S)$ est calculée. Si $\Delta f < 0$, alors S' est choisie comme solution courante (les solutions meilleures sont toujours acceptées). Si $\Delta f > 0$ et si $e^{\Delta f/t} > q$ (où $0 < q < 1$ est une valeur générée aléatoirement selon une loi uniforme de distribution), alors S' est la nouvelle solution courante. Ici t, appelé *température*, agit comme un simple paramètre de contrôle et est en général décroissant durant le processus de recherche locale afin de diminuer la probabilité de choisir une solution moins bonne.

Le premier qui a appliqué cette méthode au CARP est Li (1992), il s'agit d'ailleurs de la première métaheuristique proposée pour le CARP. Eglese (1994) a également testé la méthode du recuit simulé pour un problème de tournées sur arcs avec plusieurs dépôts, pour un problème de sablage de route. Des contraintes additionnelles liées au sablage sont prises en compte, comme des contraintes de limite de temps de travail et la possibilité pour les véhicules de s'approvisionner en sel dans des sites spéciaux.

1.8.6.3 Recherche à voisinage variable (*VNS* pour Variable Neighbourhood Search)

La technique *VNS* a été conçue par Mladenovic et Hansen (1997). L'idée de base est de commencer la recherche avec un voisinage N_0, de définir un certain nombre p de voisinages de cardinaux croissants N_1, N_2,…, N_p, de passer au voisinage suivant à chaque blocage sur un minimum local, et de revenir à N_0 après chaque amélioration.

Hertz et Mittaz (2001) ont eu l'idée de coupler cette technique avec la méthode *Carpet* pour résoudre le CARP. Un voisinage $N_k(x)$ de la solution x est défini pour k tournées de la solution avec les procédures *Postopt* et *Shorten* (Hertz *et al.* 2000). Ces procédures sont cette fois-ci appliquées à k tournées au lieu d'être appliquées à la solution complète. Hertz propose par exemple de faire varier k entre 1 et K*.

Les tests réalisés sur les instances de DeArmon (1981) et les instances générées aléatoirement par Hertz *et al*. (2000) permettent de conclure que *VNS* donne des solutions de meilleure qualité que *Carpet* sur les grandes instances. Elle produit les mêmes résultats que

Carpet sur les instances de DeArmon, avec un temps de calcul moyen diminué de 20 % par rapport à *Carpet*. Sur des instances plus grandes générées par Hertz, *VNS* surpasse *Carpet*.

1.9 Applications et extensions du CARP

Nous nous somme intéressés dans la section précédente aux problèmes de base en tournées sur arcs avec contrainte de capacité. Beaucoup d'applications concernent les réseaux routiers. Les demandes sont alors des quantités à collecter (ordures ménagères) ou à apporter (sablage de routes en cas de verglas), et les coûts sont souvent des distances ou des temps de parcours. Il s'agit d'applications où il est nécessaire d'entretenir des rues ou des routes ou d'y offrir des services comme la distribution de courrier, la collecte des déchets, le nettoyage de la voirie, la collecte de lait, le relevés de compteurs d'eau ou d'électricité, l'inspection de lignes à haute tension, le déneigement des rues et trottoirs, le fauchage de bas-côtés, le désherbage, l'itinéraires des souffleuses à neige, etc.

Depuis quelques années, ce type de problèmes intéresse de plus en plus les chercheurs et les gestionnaires d'entreprises. Dans cette section nous nous penchons sur quelques études d'applications réelles du CARP et de ses cas particuliers. D'autres extensions plutôt théoriques qui n'ont pas fait l'objet de tests sur des données réelles seront aussi présentées.

Guan (1962), qui a introduit le problème du postier chinois, lui a déjà apporté un cadre d'application en distribution du courrier. Sinon, les premières applications des problèmes de tournées sur arcs apparaissent dans la littérature au début des années 70, incluant en particulier les services publics. On peut citer les travaux de Beltrani et Bodin (1974), qui proposent un guide de modélisation et d'implémentation pour la propreté urbaine de la ville de New York. Ces travaux ont été repris par Bodin et Kursh (1978, 1979) et par Eglese et Murdock (1991) pour une application de nettoyage de la voirie. Cette dernière a fait l'objet d'autres études sur des cas réels comme Liebling (1970, 1973) pour le nettoyage des rues dans la ville de Zurich et Bodin *et al.* (1989) pour la ville d'Oyster.

Quant à la collecte des déchets, elle comprend de nombreuses contraintes supplémentaires par rapport aux problèmes académiques de tournées sur arcs. Elle a été abordée de différentes façons : Male (1973) a considéré le *M-Postman Problem*, Shuster et Schur (1974) ont développé une heuristique pour tenir compte de quelques extensions comme les demi-tours et, en 1991, Muralidharan et Pandit ont décrit la collecte des produits recyclables où les quantités à collecter peuvent être localisées sur les nœuds ou sur les liens du réseau. D'autres applications des problèmes de tournées sur arcs à la collecte des déchets peuvent être trouvées dans les travaux de McBride (1982), d'Alvarez *et al.* (1993) et de Tung et Pinnoi (2000). Nous avons notamment une collaboration avec M. C. Mourão et T. Almeida, qui ont étudié la collecte des ordures ménagères dans la ville de Lisbonne (Mourão et Almeida, 2000).

Le déglaçage et le déneigement des voies de circulation est une autre application importante parce que les opérations de déneigements urbain sont coûteuses lorsque de grandes quantités de neige s'amoncellent le long des rues : la neige doit être enlevée et transportée vers des sites de déversement. Li et Eglese (1996), Haslam et Wright (1991), Eglese et Li (1992) et Eglese (1994) ont traité ce type de problème. Dans le premier article par exemple, plusieurs contraintes additionnelles sont prises en compte comme un réseau mixte, des priorités sur les liens (arcs ou arêtes) ainsi que des sites de transferts différents du dépôt. Des

priorités sur les arcs ont aussi été considérées dans les travaux de Stricker (1970), Lemieux et Campagna (1984), Alfa et Liu (1988) et de Haslam et Wright (1991).

Une synthèse sur les opérations de déglaçage et de déneigement et des principales applications réussies aux Etats-Unis, au Canada et au Royaume–Uni sont présentées en détails dans Campbell et Langevin (1995a, 1995b, 1999). Une application à la province d'Antwerp (Belgique) peut être trouvée dans Lotan *et al.* (1996). Récemment, Labelle *et al*. (2000) ont décrit brièvement le cas de la ville de Montréal. Ils ont présenté des modèles et des algorithmes efficaces pour affecter les secteurs aux sites de déversement.

Parmi les autres applications du CARP sur les réseaux routiers, on peut noter : le relevé de compteurs d'eau ou d'électricité qui a été étudié par Stern et Dror (1979), Roy and Rousseau (1988) et Wunderlich *et al.* (1992) ; le transport scolaire dans Ferland *et al*. (1984) et la distribution du courrier dans Levy et Bodin (1988) et Roy et Rousseau (1989).

En dehors des réseaux routiers, le CARP a été utilisé pour d'autres applications n'ayant aucun rapport avec la gestion des tournées. A titre d'exemple, on peut citer l'optimisation de l'impression sur un traceur ou imprimante à laser (Ghiani and Improta, 2000b), un problème de séquencement de tâches par Anily *et al*. (2000) et la programmation de robots posant des bandes conductrices sur des circuits imprimés par Ball et Magazine (1988).

Vu la faiblesse des modèles théoriques devant la complexité des applications réelles et par souci de généralisation, plusieurs extensions ont été proposées. Par exemple, Benavent et Soler (1999) ont décrit le DRPP avec interdictions de tourner. Des travaux similaires sur le MRPP ont fait l'objet d'études de Corberan *et al*. (2000b) et de Clossey *et al*. (2001). Une extension du CARP appelée *CARPIF* (*Capacitated Arc Routing Problem with Intermediate Facilities*) a été étudiée dans Ghiani *et al*. (2001). Dans cette extension, les auteurs considèrent plusieurs sites où on peut décharger ou se réapprovisionner, en plus du dépôt. Le problème consiste à construire les tournées successives d'un véhicule : la 1ère tournée commence au dépôt, la dernière finit au dépôt et toute autre tournée sauf la dernière peut se terminer à l'un des sites intermédiaires. Il faut noter que ce problème est appelé CARPIF à tort, car il y a bien un seul véhicule. Un nom plus adapté serait *Multitrip Rural Postman Problem*.

Amberg *et al.* (2000) ont traité le CARP avec plusieurs dépôts. Un modèle similaire nommé *LRPP* (*Location Rural Postman Problem*) est étudié dans Ghiani et Laporte (1999). Ce problème se retrouve dans des contextes où l'on doit déterminer simultanément les tournées ainsi que la localisation de certains sites, ce qui peut se produire dans la distribution du courrier, l'enlèvement des déchets et l'entretien des chaussées.

1.10 Conclusion

Nous avons présenté dans ce chapitre une revue de la littérature des problèmes de tournées sur arcs et nous avons décrit un certain nombre de modèles et de méthodes pour les résoudre. Nous proposons de récapituler les principaux problèmes en tournées sur arcs abordés dans ce chapitre dans la figure 1.2.

Cette revue de la littérature nous a conduit à plusieurs remarques :

- le CARP est la pierre angulaire de la majorité des applications des problèmes de tournées sur arcs car c'est le modèle le plus général, qui contient tous les autres problèmes comme cas particuliers.

- malgré l'intérêt pratique du CARP, les modèles théoriques étudiés jusqu'à maintenant restent restrictifs par rapport aux applications réelles, qui nécessitent la prise en compte de plusieurs contraintes additionnelles.

- les méthodes et les algorithmes développés doivent être améliorés en terme de temps de calcul et généralisés pour étendre leur domaine d'application. Les heuristiques constructives semblent avoir été l'approche la plus utilisée, comparée aux méthodes exactes et aux métaheuristiques. De plus, l'approche évolutionniste comme les algorithmes génétiques n'a jamais été utilisée.

L'ensemble de ces points a conduit aux recherches qui sont exposées dans les chapitres suivants. Notre motivation est de traiter des modèles plus réalistes en tournées sur arcs et de pouvoir proposer des solutions industriellement exploitables. Ainsi, notre étude concerne une généralisation des travaux antérieurs en tournées sur arcs vers de nouveaux problèmes.

Le chapitre suivant présente ces nouveaux problèmes, une description de leur principale application (la collecte de déchets) et des notations qui vont permettre une présentation homogène du reste de la thèse. Chacun des nouveaux problèmes fait ensuite l'objet d'un chapitre spécifique.

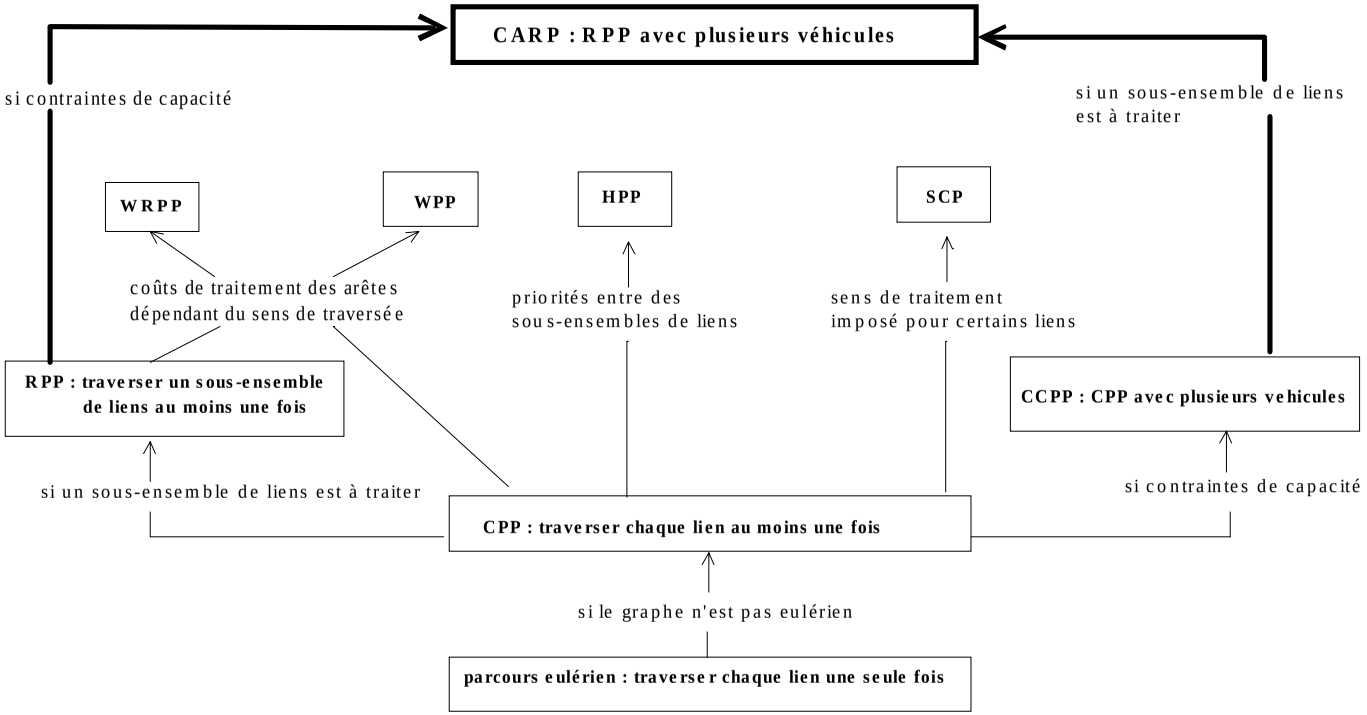


Figure 1.2 – *Synthèse des problèmes de base en tournées sur arcs*

CHAPITRE 2
Problèmes étudiés et notations

2.1 Introduction

Après avoir présenté une revue de la littérature des problèmes de tournées sur arcs et avant de rentrer dans la présentation de nos réalisations, nous consacrons ce chapitre à une vue d'ensemble du cadre de la thèse et des problèmes étudiés. Nous y plaçons aussi les définitions et les notations qui seront ensuite utilisées dans les autres chapitres.

2.2 L'application de référence : la collecte des déchets urbains

L'état de l'art du chapitre 1 montre que le CARP est le problème académique le plus réaliste pour modéliser des applications en tournées sur arcs. Cependant, il est encore trop simple pour pouvoir traiter de nombreuses contraintes réelles. La présente thèse a pour objectif principal d'étendre le CARP à des nouveaux problèmes se rapprochant plus du monde réel et de développer des méthodes d'optimisation efficaces pour les résoudre.

Sans perte de généralité, nous avons choisi comme application de référence la *collecte des déchets urbains* (*cf.* figure 2.1). La collecte des déchets acquiert une importance croissante à cause de ses implications économiques mais aussi environnementales. Les déchets peuvent prendre des formes diverses : déchets industriels banals, valorisables ou recyclables (comme le papier, les plastiques, le verre), déchets industriels spéciaux (exemple, déchets nucléaires), déchets organiques, ordures ménagères ou produits manufacturés en fin de vie (automobiles, appareils électroniques, électroménager).

En ce qui nous concerne, nous considérons uniquement la collecte des déchets urbains parce qu'elle représente la principale application des problèmes de tournées sur arcs. Le choix de cette application de référence permet de rendre ainsi plus représentatif et plus clair l'objet de notre étude. Bien entendu, notre but est de développer des modèles largement indépendants de cette application pour pouvoir traiter éventuellement d'autres problèmes comme le salage de routes ou l'inspection de lignes électriques. En fait, le réseau peut être quelconque (routier, électrique ou ferroviaire), du moment qu'il puisse être décrit sous forme d'un *graphe*. Les *nœuds* du graphe modélisent par exemple des carrefours en ville (déchets urbains), des carrefours ou localités en campagne (déneigement), des pylônes, transformateurs et centrales (réseaux électriques). Les *arcs* ou *arêtes* représentent des tronçons de rues ou de routes entre carrefours (réseaux routiers), des câbles (réseaux électriques), etc.

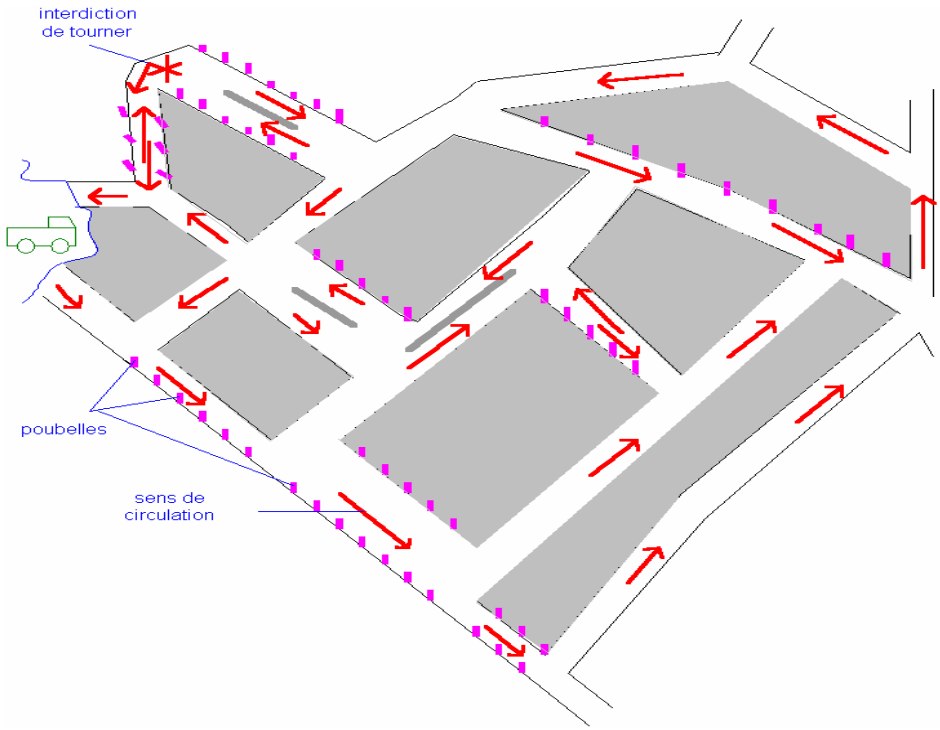


Figure 2.1 – *Exemple de réseau de rues en collecte des déchets urbains.*

Le problème de collecte des déchets urbains est défini sur un réseau de rues découpé en secteurs de collecte. Par « rue », on veut dire en fait un tronçon d'une rue réelle, reliant deux carrefours. Une rue peut être à sens unique ou à double sens, étroite ou large. Il peut y avoir des interdictions de tourner, comme des demi-tours interdits à certains carrefours ou coûteux en terme de temps. Chaque rue a une quantité de déchets à collecter (éventuellement nulle), qui peut être répartie d'un seul côté ou des deux côtés. Selon le cas, cette répartition peut nécessiter un passage ou deux. Il y a ainsi quatre façons de collecter une rue (*cf.* figure 2.5 du 2.5.2.1) : côté gauche uniquement, côté droit uniquement, côté gauche et côté droit en deux passages, côté gauche et côté droit en parallèle et dans n'importe quel sens (collecte dite *bilatérale*, *en tricot* ou *en zig-zag*).

En général, la collecte d'une ville est planifiée sur 5, 6 ou 7 jours de la semaine. Pour chaque rue, un nombre de passages par semaine et un écart raisonnable entre jours de passage sont définis. On suppose qu'on connaît aussi la quantité de déchets, le coût de collecte et le coût de traversée sans collecte (appelé par la profession *haut-le-pied* ou *HLP*). Il peut y avoir des coûts supplémentaires, par exemple quand on veut tourner à gauche dans les carrefours avec feux. Les « coûts » les plus utilisés en pratique sont en fait des longueurs (si on veut minimiser la distance totale parcourue) ou des temps (si on veut minimiser la durée totale des tournées). On peut aussi avoir un coût financier incluant les salaires (liés au temps de travail des chauffeurs), le carburant (lié à la distance), le coût d'amortissement des véhicules, etc.

La collecte est effectuée par une flotte de *véhicules* (nous dirons souvent *camions*) basés à un *nœud-dépôt* (ou *garage*). Les camions peuvent être de différents types, avec une *capacité* (charge utile) et une *autonomie*, qui résulte par exemple des horaires de travail de l'équipage ou du kilométrage permis par un plein de carburant. Chaque camion a un *gabarit* qui définit sa taille et limite l'accès à certaines rues. Chaque rue a aussi un gabarit, défini par

la largeur, la hauteur et le poids maximum des camions qui peuvent y passer. Le rayon de braquage des camions peut empêcher de tourner dans certaines rues.

Une tournée de collecte est assurée par un seul camion. Le camion part du dépôt, effectue un ou plusieurs tours se terminant à un *exutoire* ou *lieu de vidage*, puis revient au dépôt. En pratique, il peut y avoir plusieurs exutoires, équivalents ou réservés à certains types de déchets, avec éventuellement des capacités de stockage limitées pour chaque type. Les contraintes de capacité des véhicules s'exercent sur chaque tour, tandis que les contraintes d'autonomie concernent la tournée entière.

La figure 2.2 illustre deux exemples de tournées. Celle de gauche comprend un seul tour. Celle de droite est une tournée composée de plusieurs tours, appelée aussi *multitour*. Dans ce dernier cas, on peut différencier le premier tour, de type dépôt – collecte – vidage, des suivants, de type vidage – collecte – vidage. Remarquons que tous les tours sont du même type si le dépôt coïncide avec l'exutoire, ce qui est une situation assez fréquente.

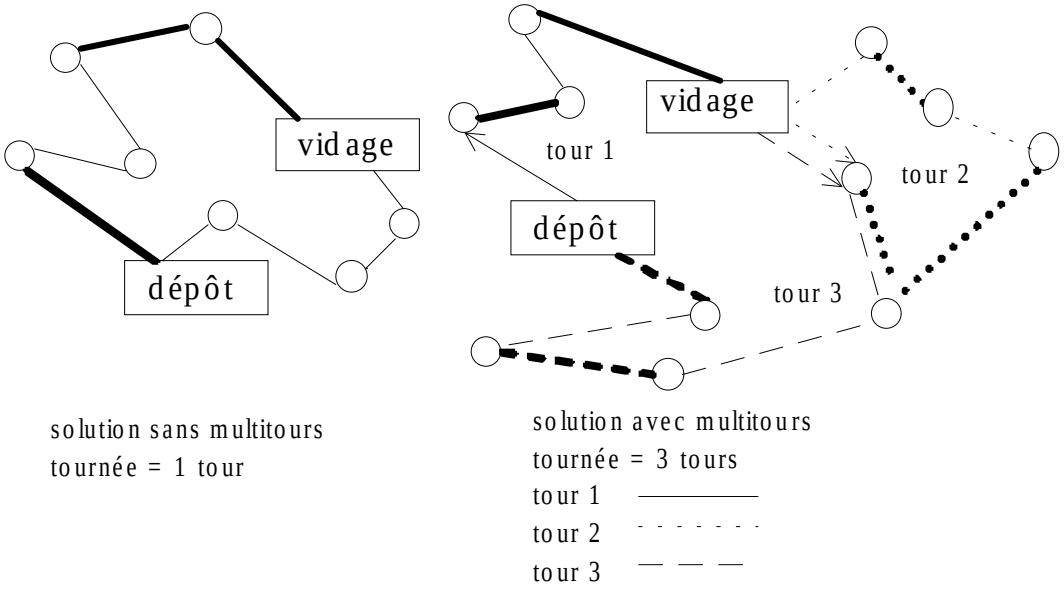


Figure 2.2 – *Exemple de tournée simple et de multitour.*
En traits gras, les traversées avec collecte. En traist fins, les traversées en HLP.

Les camions doivent collecter toutes les rues ayant une quantité non nulle de déchets. Toute rue à collecter doit être traitée par un seul véhicule et lors d'un seul tour. Par contre, toute rue peut être traversée plusieurs fois en haut-le-pied, par un même camion ou par des camions différents. L'objectif est de calculer un ensemble de tournées collectant toutes les rues, en minimisant le coût total. Le nombre de camions utilisés peut être imposé (plein emploi des chauffeurs), être plafonné à une valeur maximale ou être une variable de décision, éventuellement à minimiser.

2.3 Extensions étudiées dans la thèse

Notre bref exposé de la collecte des déchets urbains permet de montrer que le CARP est un problème trop restrictif. Par exemple, il considère un graphe non orienté, dont les arêtes ne permettent de modéliser que des rues collectées en tricot, et les interdictions de tourner ne sont pas prises en compte. En plus, les articles sur le CARP concernent une seule période (une journée par exemple), alors que l'optimisation tactique implique une planification des tournées sur un horizon de plusieurs jours. D'autres aspects pratiques importants, comme des données imprécises sur les quantités, doivent être également pris en compte pour obtenir des solutions industriellement exploitables.

Ce constat nous a motivé pour développer de nouveaux modèles prenant en compte des contraintes additionnelles, afin de se rapprocher le plus possible des applications réelles. A part des travaux sur le CARP de base, nous proposons d'étendre ce dernier vers trois nouveaux problèmes. Plus précisément, la thèse comprend quatre thèmes principaux (*cf.* figure 2.3) :

- le premier que nous abordons et que nous présentons dans le chapitre 3 est le CARP de base. Nous avons voulu proposer des méthodes de résolution plus efficaces et plus souples que celles de la littérature, pouvant être ensuite étendues à des généralisations du CARP.

- le deuxième, étudié dans le chapitre 4, est une deuxième généralisation du CARP au cas où les demandes sont incertaines : il s'agit du *SCARP* (*Stochastic CARP*). Ce modèle permet de traiter les cas où les quantités à collecter ne sont que des estimations. L'objectif est de concevoir des solutions plus robustes, c'est-à-dire dont le coût total est peu affecté par les variations aléatoires des quantités.

- le troisième, que nous traitons dans le chapitre 5, est notre première généralisation du CARP vers des problèmes plus réalistes, le *ECARP* (*Extended CARP*). Ce CARP étendu traite des réseaux plus complexes et intègre plus de contraintes réelles.

- le quatrième thème et troisième généralisation est le *PCARP* (*Periodic CARP*), prolongement naturel de nos travaux vers la planification à moyen terme des tournées (horizon de plusieurs périodes ou jours). Le but est d'affecter les tâches aux périodes et de construire les tournées dans chaque période, de façon à minimiser un coût total sur l'horizon. Le chapitre 6 lui est consacré.

Le CARP et les nouveaux problèmes que nous proposons (*ECARP, SCARP* et *PCARP*) sont très combinatoires et les applications que nous visons sont hors de portée des méthodes exactes. De nouvelles techniques de résolution, appelées *métaheuristiques* sont apparues depuis deux décennies pour résoudre les problèmes d'optimisation combinatoire. Elles ont prouvé sur de multiples exemples leur efficacité et leur robustesse. Elles peuvent produire des solutions quasi-optimales, à condition d'être bien conçues et de disposer au minimum de bornes inférieures pour évaluer les résultats. En tournées sur arcs, nous avons recensé quelques travaux utilisant le recuit simulé ou la méthode tabou, mais aucun qui concerne des algorithmes de type évolutionniste.

Ce constat nous a motivé pour développer des algorithmes génétiques. Notre méthodologie de travail a consisté à développer d'abord un algorithme génétique pour le CARP de base, puis à l'étendre aux autres problèmes de la thèse. Ces extensions sont évidemment non triviales mais elles démontrent la grande flexibilité des méthodes génétiques. Chaque problème étendu aura donc au moins un algorithme génétique comme méthode de résolution. A cause de ce rôle très important joué par l'algorithme génétique pour le CARP de base, nous avons choisi de lui consacrer le chapitre 3. Cette façon de procéder permet d'introduire de manière pédagogique de nombreux concepts utiles pour les autres chapitres.

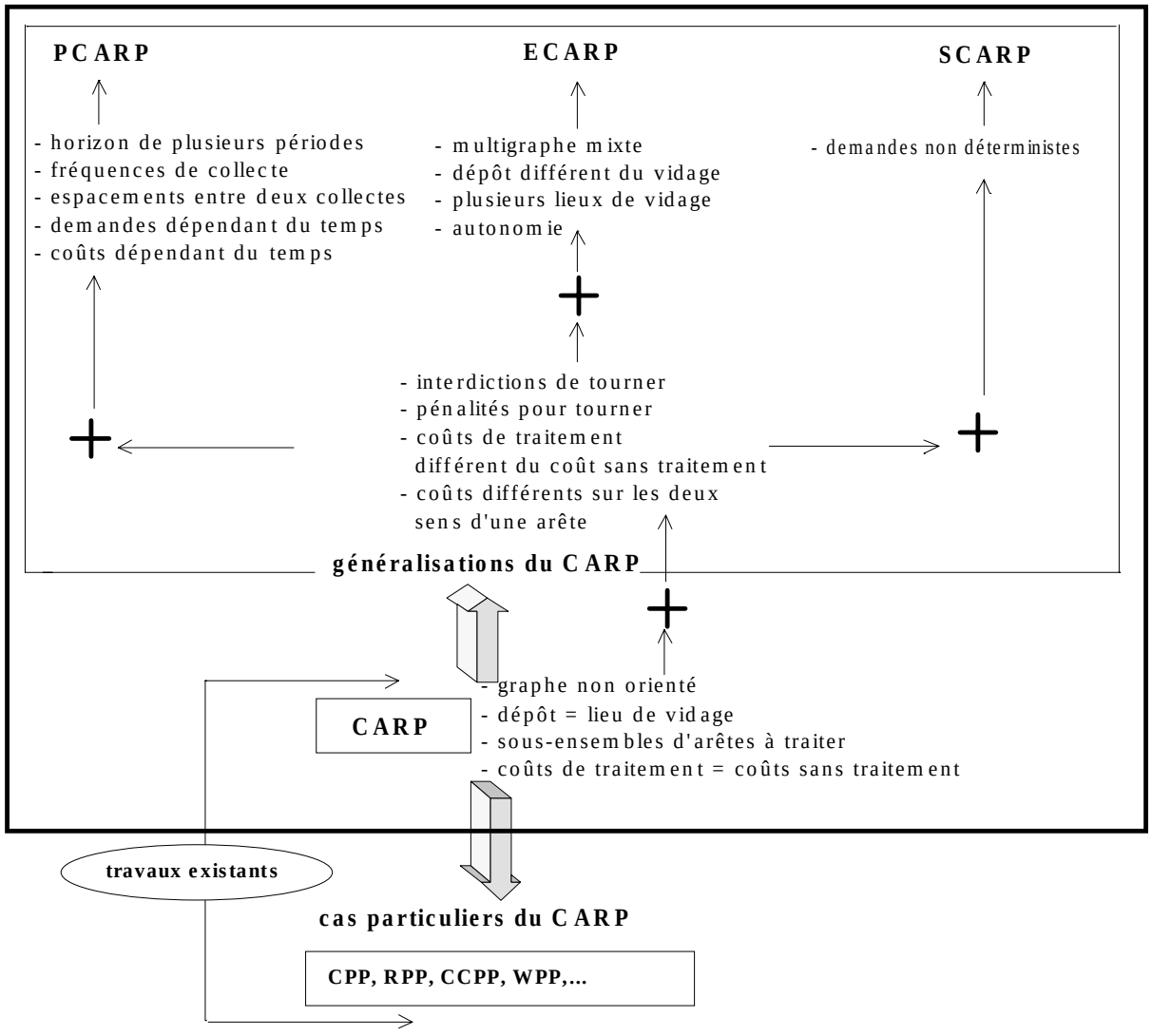


Figure 2.3 *– Problèmes étudiés dans la thèse.*

2.4 Hypothèses de travail

Nous pensons qu'il est intéressant et utile de préciser ici les différentes hypothèses qui vont justifier les notations adoptées pour la suite de la thèse. Malgré ces restrictions, nous verrons que nos modèles peuvent gérer des problèmes déjà très réalistes et variés.

- Nous supposons un seul secteur de collecte, une flotte homogène et on ne gère pas les multitours. En effet, flottes hétérogènes et multitours soulèvent de sérieuses complications comme l'affectation des véhicules aux tournées.

- Nous ne traitons pas le cas de plusieurs dépôts, qui pose également des problèmes d'affectation de véhicules. En collecte de déchets, ce cas est rare. En général (à Troyes par exemple), il existe un seul dépôt par secteur de collecte.

- Nous ne prenons pas en compte des coûts fixes liés à l'utilisation des véhicules. Il suffirait pour les prendre en compte d'ajouter ces coûts aux arcs issus du dépôt.

- Nous ne considérons pas de fenêtres de temps.

2.5 Notations et structure de données

Nous avons fait l'effort de définir des notations communes pour tous les problèmes étudiés pour faciliter la lecture et uniformiser la présentation. Il s'agit souvent de notations nouvelles car les notations classiques deviennent vite insuffisantes à cause du grand nombre de contraintes gérées. Cette section introduit et explique ces notations. Nous avons recherché des identificateurs mnémotechniques, avec si possible une seule lettre pour alléger. Il n'y a aucune différence entre majuscules et minuscules, en vue de la programmation des algorithmes dans des langages comme Pascal ou Fortran. Les conventions suivantes vont être utilisées : pour toute variable x indicée par u, on écrira au choix $x(u)$ ou x_u. Pour la valeur minimale, maximale et totale des $x(u)$, on écrira respectivement x_{min}, x_{max} et x_{tot}.

2.5.1 Conventions algorithmiques

Les algorithmes sont donnés en police Courier, sans italiques, et avec les structures suivantes : **if** … **then** … **endif** ou **if** … **then** … **else** … **endif**, **while** … **endwhile**, **repeat** … **until**, **for** … **do** … **endfor**, **case** … **endcase**, **loop** … **exit when** … **endloop** (boucle avec sortie par le milieu). Ces structures sont mises en évidence en gras. Comme elles sont bien délimitées par deux mots-clés, les begin…end à la Pascal sont inutiles. On peut écrire une structure courte sur une seule ligne, par exemple : **if** a = b **then** c := 1 **endif**.

L'instruction **for** nécessite quelques commentaires. Elle fait normalement varier un indice dans un intervalle donné : **for** u := deb **to** fin. Comme dans tous les langages de programmation, on suppose que les valeurs des expressions *deb* et *fin* ne sont évaluées qu'une fois, à l'entrée de la boucle. Les formes **for all** u **in** A et **for all** u **in** 1..|A| n'impliquent aucun ordre particulier de balayage. Les formes **for** avec **to** doivent être utilisées si un ordre

spécifique de balayage est nécessaire. Un **for** avec un **if** sur l'index de boucle peut être condensé en un **for** … **with** ou **for** … **such that**, par exemple **for** $u := 1$ **to** m **with** $r(u) > 0$.

Les instructions des algorithmes utilisent nos notations mais peuvent être des traitements non détaillés, comme *Trier T en ordre croissant*. Les points-virgules sont requis seulement si plusieurs instructions se suivent sur une même ligne. L'affectation est notée := pour la distinguer de l'égalité et les affectations multiples sont acceptées : a, b := 0. Les commentaires suivent // ou un !. Les expressions binaires sont évaluées en *court-circuit* (de gauche à droite, avec arrêt quand le résultat est acquis). Ainsi, une boucle de recherche d'un entier x dans une table T de n éléments peut s'écrire *Tant que* $(i \leq n)$ et $T(i) \neq x$: si i dépasse la fin de T, le contenu indéfini de $T(i)$ n'est pas évalué.

2.5.2 Données

2.5.2.1 Réseau mixte (hors données liées à l'activité de collecte)

Le CARP de la littérature considère un graphe non orienté $G = (V, E)$. Le dépôt est aussi l'exutoire. La flotte est homogène. E comprend un sous-ensemble R d'arêtes à traiter, dans n'importe quel sens. Chaque arête a une demande et un coût de traversée (égal au coût de traitement et ne dépendant pas du sens de passage). Il faut calculer un ensemble de tournées de coût minimal, traitant tous les arcs de R, en respectant la capacité des véhicules.

Ce CARP dit «de base» ne permet pas de gérer des contraintes réelles comme par exemple une flotte hétérogène de véhicules, un réseau routier complexe, des interdictions de tourner. Il permet de traiter seulement un réseau de rues à double sens, à collecter en tricot, et avec minimisation de la distance parcourue. La plupart des méthodes publiées sont cependant adaptables pour un réseau complètement orienté $G = (V, A)$. Dans ce cas on peut modéliser des sens uniques, des rues à double sens à côtés collectables séparément, mais plus des rues collectables en tricot. Pour la thèse, nous nous intéressons à un CARP étendu (*ECARP*), avec :

- un multigraphe mixte combinant des arêtes et des arcs ;
- des interdictions de tourner et des pénalités pour tourner ;
- un dépôt différent du lieu de vidage, voire même plusieurs lieux de vidage ;
- des coûts de traitement distinct des coûts de traversée en haut-le-pied (sans traitement) ;
- des coûts pouvant différer sur les deux sens d'une arête ;
- une limite de coût sur chaque tournée (autonomie).

Ce modèle enrichi permet déjà de traiter des réseaux très réalistes de type centre-ville et de minimiser au choix la distance parcourue ou le temps de travail.

Nous avons pensé d'abord à utiliser les notations de la littérature pour les problèmes mixtes comme le MRPP (aucun article sur les problèmes mixtes à plusieurs véhicules n'a encore été publié). Le réseau y est noté $G = (V, E, A)$, avec A l'ensemble des arcs et E celui des arêtes. Un arc ou une arête a un seul coût, ce qui suppose que toute arête ait un coût indépendant du sens. Mais cette notation qui met à part les arêtes dans E a des inconvénients :

- les problèmes *venteux* (avec des coûts dépendant du sens) ne sont pas modélisables ;
- les interdictions de tourner ne peuvent être prises en compte ;
- la partition est ambiguë car une arête non à collecter peut être codée au choix par une arête dans *E* ou deux arcs opposés dans *A*.

Nous proposons donc de toujours convertir le réseau donné en un *graphe complètement orienté* (*graphe interne*), dans lequel chaque arête est remplacée par deux arcs opposés pour modéliser les deux sens possibles de traversée. Chaque arête à traiter est codée par deux arcs opposés *u* et *v*, liés par un pointeur *inv* (comme *inverse*), tel que *inv*(*u*) = *v* et *inv*(*v*) = *u*. Pour une arête non traitable, il n'y aura aucune différence après conversion avec deux arcs opposés *u* et *v* déjà présents dans le graphe de départ : on posera *inv(u)* = *inv*(*v*) = 0. De tels arcs opposés mais non liés seront dits *indépendants*.

La figure 2.4 représente un graphe de départ et le graphe interne obtenu après conversion des arêtes et indexation des arcs résultants de 1 à 5. L'arête *c* n'est pas à traiter : elle est codée par deux arcs indépendants 3 et 5 avec *inv*(3) = *inv*(5) = 0. L'arête *b*, à traiter, est codée par deux arcs liés 2 et 4. Dans toutes les figures de la thèse, deux arcs liés codant une arête à traiter seront toujours entourés par une petite ellipse.

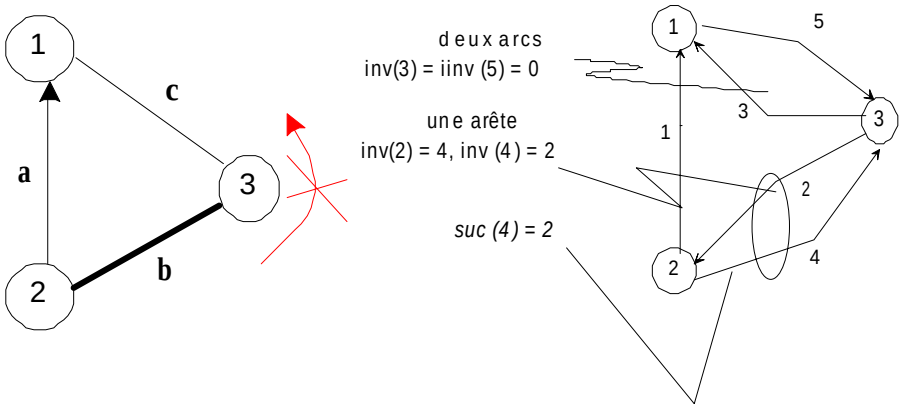


Figure 2.4 – *Représentation interne du graphe.*
A gauche le graphe de départ avec les liens à traiter (traits gras). A droite le graphe interne.

En résumé, un arc joue un double rôle dans notre modèle : il sert à définir une possibilité de déplacement d'un point à un autre, indiquant aux algorithmes qu'il peut être utilisé en *HLP* et il sert aussi à indiquer les tâches (arcs à collecter). Cette dualité est plus claire dans les problèmes de déneigement. Un tronçon d'autoroute d'une ville A vers une ville B avec 3 voies peut être représenté par un seul arc si l'autoroute n'a pas besoin d'être traité (cela suffit pour indiquer la possibilité d'aller de A à B). Il sera représenté par 3 arcs parallèles (3 tâches) si chaque voie nécessite un passage de chasse-neige.

Considérons le réseau de rues de la figure 2.5, avec le graphe interne associé. Les valeurs sur les arcs sont l'index de l'arc et la quantité à collecter. Les quantités nulles indiquent les arcs non collectés. La rue à sens unique (*a*) a deux côtés à traiter. Supposée trop large pour une collecte en tricot, chaque côté doit être traité séparément : elle est donc codée par deux arcs parallèles 1 et 2. La rue à double sens (*b*) a deux côtés à traiter, mais on suppose qu'elle est assez étroite pour une collecte en tricot : elle est modélisée par les arcs liés 6 et 7. La rue à double sens (*c*), supposée trop large pour une collecte en tricot, donne les arcs 3 et 4. La rue

(*d*) est à double sens mais avec un seul côté à traiter. Elle est codée par les arcs 5 (quantité nulle) et 6. La rue à double sens (*e*) n'est pas à traiter mais donne deux arcs 8 et 9 pour ses deux sens de traversée. La rue à sens unique (f) avec un côté à collecter est codée par un seul arc (10). Il en serait de même avec deux côtés collectables en tricot.

Notre modélisation des configurations possibles pour la collecte d'une rue est résumée dans la figure 2.6. Comme application de cette modélisation à l'exemple de la figure 2.1, on obtient le graphe interne de la figure 2.7 où les arcs à traiter sont en gras, le dépôt et le vidage sont représentés par une boucle (voir 2.5.2.4).

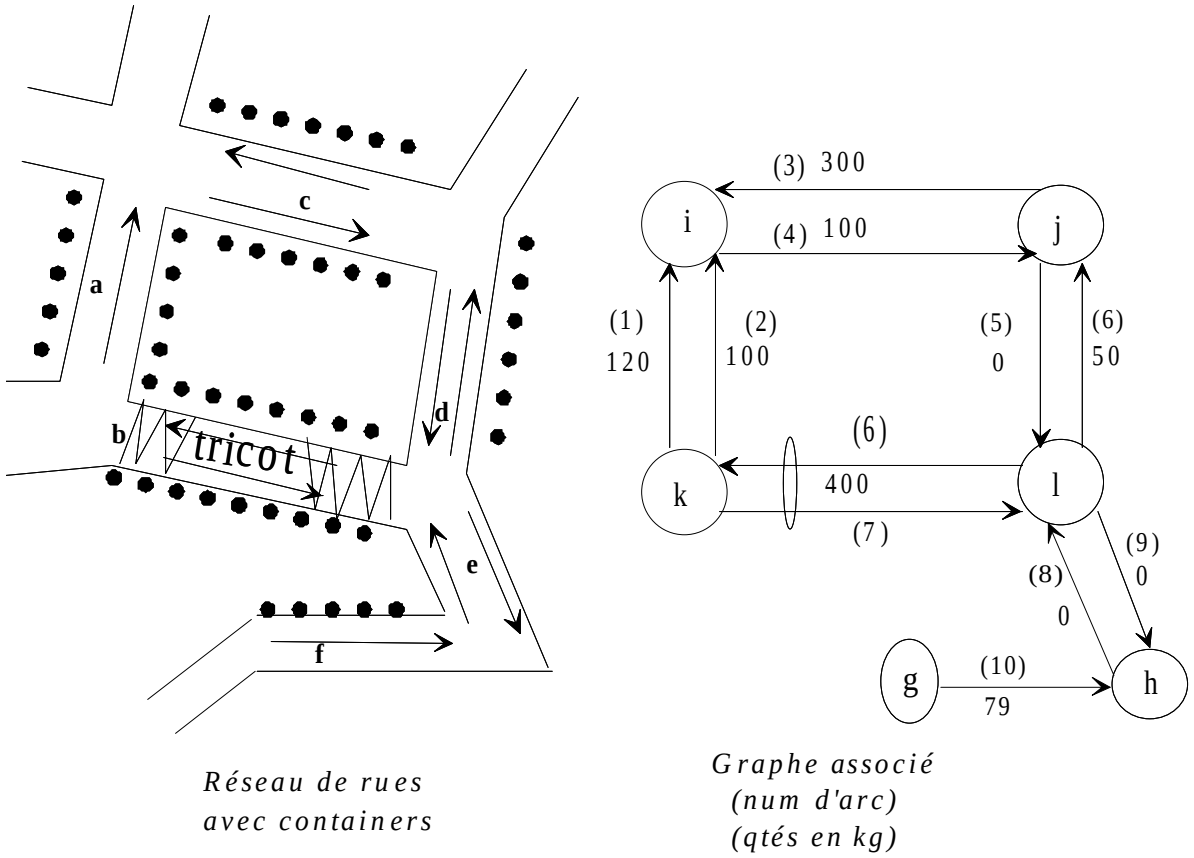


Figure 2.5 - *Graphe associé à un réseau de rue avec contraintes.*

Dans la suite, le graphe interne $G = (V, A)$ est supposé fortement connexe. V est un ensemble de n nœuds (carrefours) et A est un ensemble de m arcs (tronçons de rues). Si G est un 1-graphe sans interdictions de tournées, il peut être défini classiquement, avec pour tout nœud i une liste de nœuds successeurs $\Gamma(i)$, une liste de nœuds prédécesseurs $\Gamma^{-1}(i)$, ou une liste d'arcs partant de i, $\omega^{+}(i)$. G peut cependant être un p-graphe, avec des arcs multiples entre nœuds. A cause de cela et des interdictions de tourner, noter un arc comme un couple de nœuds (i, j) est ambigu. On préfère donc désigner les arcs par des index entre 1 et m.

Les indices d'arcs seront souvent notés u, v, z car i, j, k rappellent trop les nœuds. Tout arc u a les attributs suivants :

- un nœud de début $b(u)$ comme *begin node ;*
- un nœud de fin $e(u)$ comme *end node* ;
- un coût de traversée en haut-le-pied $c(u) \in \mathbb{N}$, souvent un temps ou une distance.

G est entièrement défini en donnant pour tout arc *u* la liste *suc*(*u*) des arcs successeurs permis ou celle *pre*(*u*) des prédécesseurs permis. S'il n'est pas permis de tourner d'un arc *u* vers un arc contigu *v* (*e*(*u*) = *b*(*v*)), alors $v \notin suc(u)$. Les interdictions de tourner sont donc transparentes. Ainsi, dans l'exemple de la figure 2.4, le seul successeur de 4 est 2 à cause de l'interdiction de tourner de *b* à *c*. Pour tout couple d'arcs contigus (*u*,*v*) avec $v \in suc(u)$, on définit une pénalité entière *pen*(*u*,*v*), nulle par défaut. Elle permet de modéliser des attentes, par exemple quand on veut tourner à gauche à un carrefour avec feux

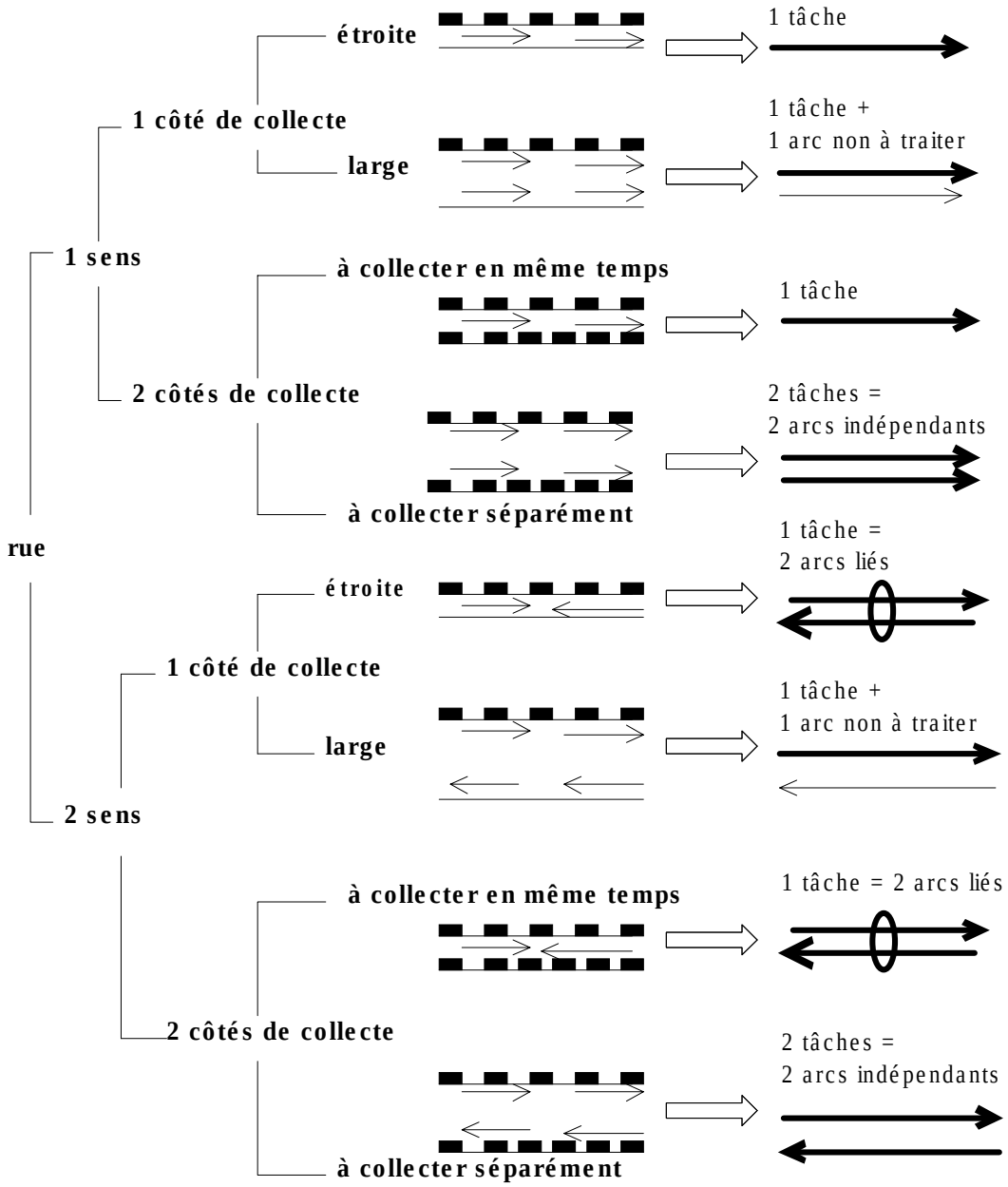


Figure 2.6 - *Codage des rues.*

2.5.2.2 Le problème du distancier

Les algorithmes de résolution ont besoin de connaître le coût des plus courts chemins dans le réseau. Les plus courts chemins (*en coût de traversée*) sont pré-calculés dans une matrice distancier *D* (coûts des chemins) et *P* (composition des chemins). Nous supposons que *D* et *P* font donc partie des données. Pour le CARP de base, des chemins entre nœuds suffisent, les deux matrices *D* et *P* sont $n \times n$. $d(i,j)$ est le coût des plus courts chemins du nœud *i* au nœud *j*. On peut calculer ces matrices avec l'algorithme classique de Dijkstra (Cormen *et al*., 1990), rappelé au début du chapitre 3.

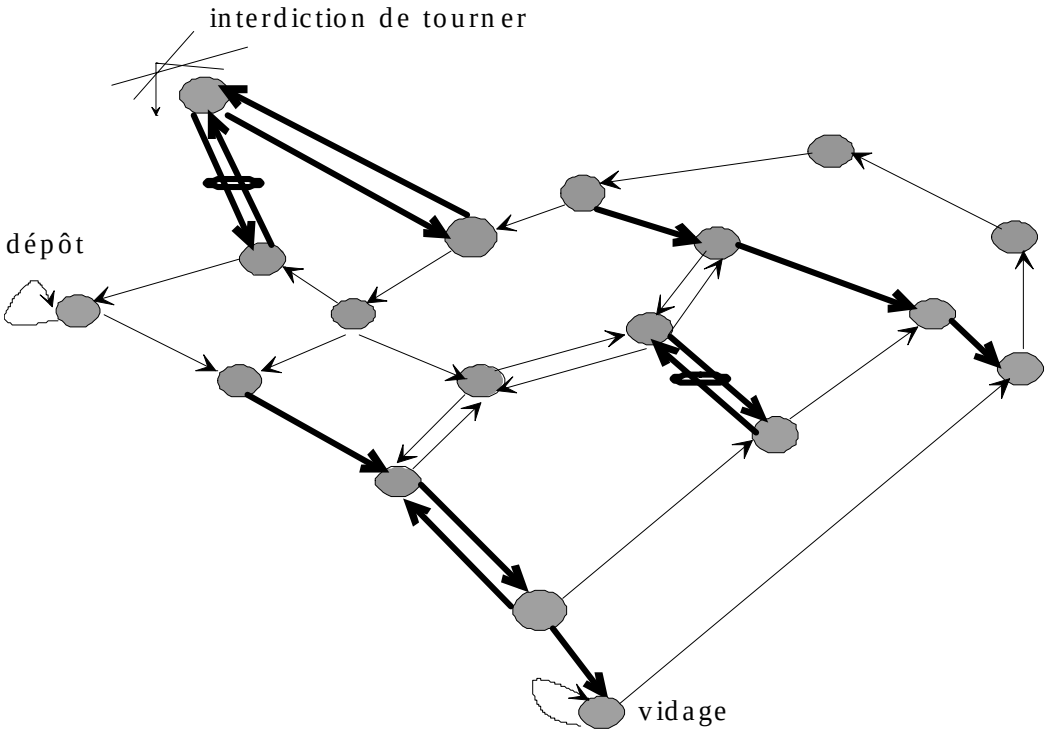


Figure 2.7 - *Graphe interne pour l'exemple de la figure* 2.1.
(*les arcs en gras sont à collecter*)

La ligne *i* de *P* stocke *n* plus courts chemins de *i* vers tout nœud, sous forme d'une anti-arborescence : $P(i, j)$ est le prédécesseur de *j* sur le plus court chemin de *i* à *j*. Ainsi, en posant $P(i, i) = i$ ou 0, le chemin de *i* à *j* peut être récupéré (à l'envers) comme suit :

```
k := j
ecrire k
while k <> i
  k := P(i,k)
endwhile
```

En cas d'interdictions de tourner, *D* et *P* deviennent $m \times m$. $d(u,v)$ est alors le coût des plus courts chemins de l'arc *u* à l'arc *v non inclus* et tenant compte des interdictions de tourner. Ainsi, dans la figure 2.8, un calcul de plus court chemin nœud à nœud entre *i* et *j* donne le chemin (i, k, j) de coût $d(i, j) = 20$ mais non réalisable. Le meilleur chemin respectant les interdictions de tourner est en fait (i, x, y, j), de coût $d(u, v) = 21$.

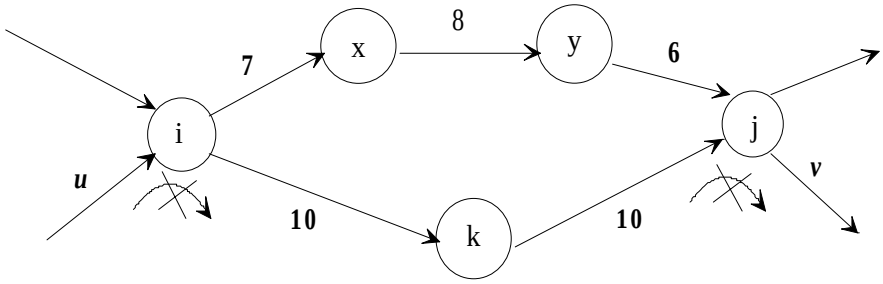


Figure 2. 8 *– Plus court chemin entre arcs avec interdictions de tourner.*

Ces distances arc à arc rendent transparentes les interdictions de tourner et permettent un calcul plus simple de la variation de coût quand on insère un arc à traiter *z* entre deux arcs *u* et *v* : $d(u,z) + c(z) + d(z,v) - d(u,v)$. Une adaptation de l'algorithme de Dijkstra pour calculer des distances arc à arc est décrite dans le chapitre 5.

2.5.2.3 Arcs à traiter

Dans le réseau réel, les véhicules effectuent un ensemble de $\tau = \varepsilon + \alpha$ *tâches*, comprenant ε arêtes à traiter (tâches-arêtes) et α arcs à traiter (tâches-arcs). Rappelons qu'en collecte de déchets chaque tâche est un traitement de rue : une tâche-arête correspond à une rue à collecter en tricot, tandis qu'une tâche-arc représente une rue à sens unique ou un coté d'une rue à double sens. Après codage dans le graphe interne, toutes ces tâches donnent lieu à un ensemble *R* de ρ arcs *requis*, avec $\rho = 2\varepsilon + \alpha$. La figure 2.9 résume cette classification.

Pour chaque arc requis *u*, on définit en plus :

- une demande $r(u) \in \mathbb{N}$, comme *request*, aussi appelée *quantité* ;
- un coût de traitement $w(u) \in \mathbb{N}$ (*working cost)*, souvent une durée.
- un pointeur *inv*(*u*) sur l'arc opposé, si ces deux arcs codent une arête.

Les deux arcs *u* et *v* codant une tâche-arête sont tels que $inv(u) = v$, $inv(v) = u$ et $r(u) = r(v)$. En général $c(u) = c(v)$ et $w(u) = w(v)$, sauf dans les cas venteux. Pour tout arc *u* ne codant pas une arête, on pose par convention $inv(u) = 0$ et, pour tout arc *u* non requis, $r(u) = w(u) = 0$. En collecte, *R* est l'ensemble des arcs de quantité non nulle, et on a $w(u) \geq c(u)$.

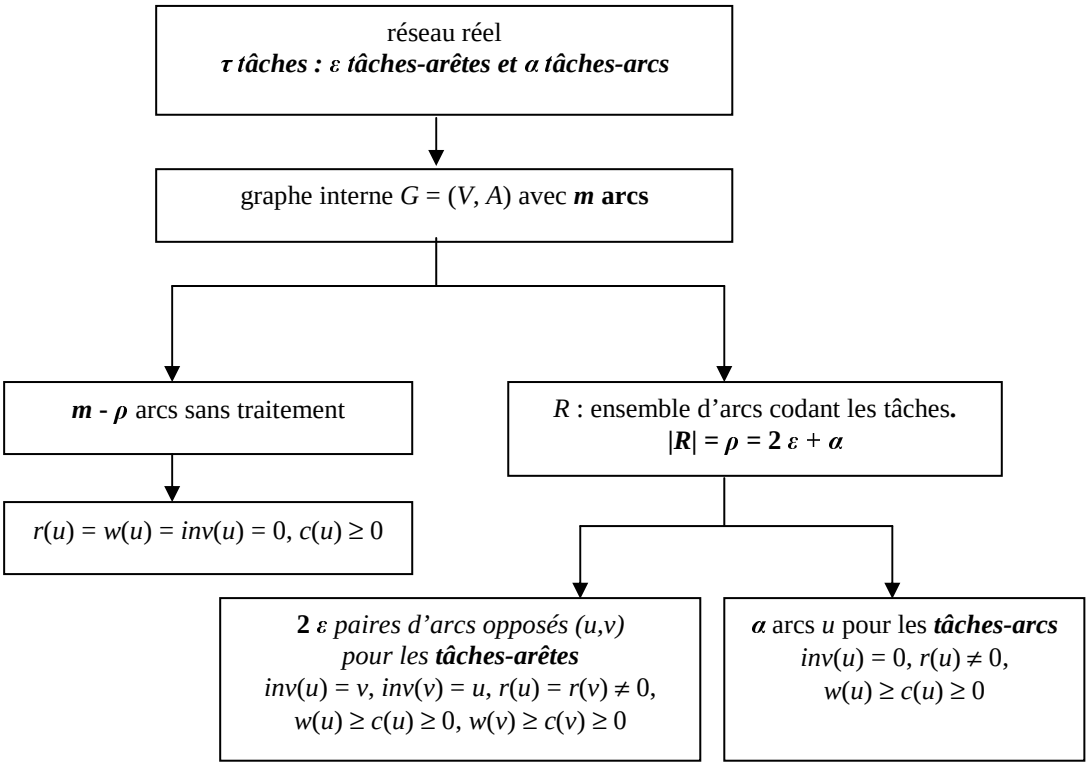


Figure 2.9 *– Partitions de l'ensemble des arcs.*

2.5.2.4 Dépôts et lieux de vidage

D'après nos restrictions (*cf.* 2.4), V ne contient qu'un nœud-dépôt, σ. Par contre, il peut y avoir un ensemble *dumpnodes* = $\{\varphi_1, \varphi_2, ..., \varphi_{ndumps}\}$de *ndumps* nœuds de vidage. Comme nous utilisons des distances arc à arc dans la thèse, ces nœuds seront peu utilisés : on ajoutera systématiquement à A une boucle fictive $s = (\sigma, \sigma)$ pour le dépôt et un ensemble *dumploops* = $\{t_1, t_2, \ldots, t_{ndumps}\}$ de *ndumps* boucles fictives pour les exutoires, avec $t_i = (\varphi_i,\varphi_i)$ pour $i = 1, 2, \ldots,$ *ndumps*. Le coût d'une opération de déchargement est noté δ. Il est identique pour tous les lieux de vidage. Si un arc u est une boucle fictive de dépôt ou de vidage, on pose par convention $r(u) = c(u) = w(u) = 0$. Le plus souvent, il n'y a qu'un exutoire. Dans ce cas, pour alléger les notations, ce nœud et sa boucle fictive seront notés respectivement φ et t.

Dans le cas de plusieurs exutoires, si un arc u est traité à la fin d'une tournée, il est clair qu'il faut aller vider au lieu de vidage *dump*(u) minimisant le coût *term*(u) nécessaire pour finir la tournée après u. Ces coûts ne dépendent pas de la tournée et peuvent être pré-calculés en $O(m.ndumps)$ au chargement des données :

```
dump(u) := argmin{d(u,t)+d(t,s) | t ∈ dumploops}        (2.1)
term(u) := d(u,dump(u)) + δ + d(dump(u),s)              (2.2)
```

2.5.2.5 Véhicules

On a au dépôt une flotte suffisante de *nva* (*number of vehicles available*) véhicules identiques de capacité Q et d'autonomie L. Le nombre de véhicules réellement utilisés *nvu* (*number of vehicules used*) est égal au nombre de tournées construites puisqu'on exclut les multitours. C'est en général une variable de décision, valant au moins $\lceil r_{tot} / Q \rceil$. Pour avoir des solutions réalisables, on suppose que chaque tâche est traitable par un seul véhicule (2.3) et que l'autonomie des véhicules permet d'aller traiter une tâche et de revenir au dépôt (2.4).

$$\forall\, u \in A : r(u) \leq Q \tag{2.3}$$
$$\forall\, u \in A : d(s,u) + w(u) + term(u) \leq L \tag{2.4}$$

2.5.2.6 Planification

Le chapitre 6 consacré au CARP multi-période (*PCARP*) nécessite des notations supplémentaires. Ce problème est défini sur un horizon H de *np périodes* ou *jours*. Chaque arc requis u doit être traitée $f(u)$ fois sur l'horizon, $1 \leq f(u) \leq np$, et au plus une fois dans chaque période. Bien entendu, si l'arc u code une tâche-arête, on a $f(u) = f(inv(u))$. Avec nos notations, le nombre total de services *ns* à assurer sur l'horizon est défini par (2.5), en évitant de compter deux fois la fréquence d'une arête :

$$ns = \sum_{\substack{u \in R \\ inv(u) < u}} f(u) \tag{2.5}$$

Dans le PCARP, les demandes et les coûts peuvent dépendre du temps. Le coût $c(u)$ et la quantité $r(u)$ d'un arc u seront notés respectivement $c(u, p)$ et $r(u, p)$ quand ils dépendent de la période p. Il peut aussi y avoir, comme en collecte des déchets, une production journalière $prod(u, p)$ qui s'accumule avec le temps. La quantité collectée en période p dépend dans ce cas du temps écoulé depuis la dernière collecte.

2.5.3 Listes

Nos algorithmes utilisent beaucoup les listes. Une liste L est une suite ordonnée d'éléments, éventuellement vide. $L(k)$ désigne l'élément de rang k, (e) une liste réduite à l'élément e et $length(L)$ ou $|L|$ ou $card(L)$ le nombre d'éléments. On suppose qu'un objet-liste est implémenté comme un descripteur (*record* par exemple), contenant le nombre d'éléments ainsi que les adresses du premier et du dernier élément. Les éléments peuvent être stockés dans un tableau ou être alloués dynamiquement, sous forme de cellules doublement chaînées entre elles.

Une liste peut contenir des listes : par exemple, une solution du CARP est une liste de tournées, qui sont elles-mêmes des listes de tâches. Dans ce cas, on note $L(i, j)$ l'élément de

rang *j* dans la liste *L*(*i*). Les opérations ensemblistes s'appliquent aux listes : appartenance d'un élément $e \in L$, union, intersection, etc. On considère aussi les primitives ci-dessous (sauf exceptions signalées, il s'agit de procédures, pas de fonctions) :

- *clear* (*L*) ou $L := \varnothing$: initialise la liste à vide (les cellules ne sont pas libérées).
- *prev(L,e)* : *fonction* renvoyant l'élément avant *e* dans *L*.
- *next*(*L*,*e*) : *fonction* renvoyant l'élément après *e* dans *L*.
- *insert* (*L*,*k*,*e*) : insère une cellule contenant *e* avant *L*(*k*).
- *enqueue* (*L*,*e*) : insère *e* à la fin de *L*, équivaut à *insert* (*L*, |*L*|+1, *e*).
- *push* (*L*,*e*) : insère *e* en tête de *L*, équivaut à *insert* (*L*, 1, *e*).
- *remove* (*L*,*k*,*e*) : détruit la cellule *L*(*k*) après avoir renvoyé son élément dans *e*.
- *first* (*L*) et *last* (*L*) : *fonctions* équivalentes à *L*(1) et *L*(|*L*|).
- *L* := *L'* : copie le descripteur de *L'* (pas les cellules) dans celui de *L*.
- *copy* (*L*, *i*, *j*) : *fonction* renvoyant une liste avec les cellules *L*(*i*) à *L*(*j*) inclus.
- *concat* (*L*,*L'*) ou *L*+*L'* : *fonction* concaténant les cellules de *L'* après celles de *L*.
- *inverse* (*L*) : *fonction* inversant l'ordre des cellules dans L.

Après un *remove*, les éléments suivant celui qui a été enlevé ont un rang diminué de 1. Les opérations *length*, *clear*, *first*, *last*, *enqueue* et *concat* sont en *O*(1), ainsi qu'un *insert* ou *remove* sur le premier ou le dernier élément. Pour un rang quelconque *k*, un accès à *L*(*k*), un *remove* ou un *insert* coûtent en théorie *O*(*k*). En pratique, ces opérations coûtent aussi *O*(1) quand on les applique à des éléments consécutifs, car passer de *L*(*k*) à *L*(*k*+1) se font en *O*(1) en suivant les chaînages. La fonction *copy* coûte *O*(*j*-*i*), *concat* est en *O*(|*L'*|), *inverse* en *O*(|*L*|).

2.5.4 Solutions et tournées

Solutions et tournées sont introduites maintenant car ce sont des listes. Une tournée θ est une liste d'index de tâches distinctes, supposées reliées par des plus courts chemins. Les notations de listes s'appliquent, ainsi θ(*k*) est la tâche de rang *k* et |θ| est le nombre de tâches. Une solution *S* est une liste de *nvu* tournées $S = (\theta_1, \theta_2, \ldots, \theta_{nvu})$. Deux *fonctions génériques* s'appliquent aux tournées et aux solutions : *cost* pour le coût total et *load* pour la charge totale. *load* s'obtient en sommant les quantités des éléments.

Par exemple, pour une solution *S*, *load*(*S*) = r_{tot} et *cost* (*S*) est la somme des coûts des tournées. Pour une tournée θ, *load*(θ) est la somme des quantités collectées par la tournée mais le coût est plus compliqué à calculer, on l'obtient par l'algorithme suivant :

```
cost := 0
last := s
for k := 1 to |θ|
  cost := cost + d(last,θ(k)) + w(θ(k))
  last := θ(k)
endfor
if |θ| > 0 then cost := cost + term(last)
```

Cet algorithme renvoie un coût nul pour une liste vide, ce qui est correct. Sinon, le calcul traduit le fait qu'une tournée non vide part de la boucle-dépôt *s*, traite la liste des tâches, puis se termine à coût minimal (*cf.* le calcul de *term* au 2.5.2.4). On suppose qu'en pratique *cost* et *load* ne sont pas calculés à la demande par une boucle, mais qu'ils sont obtenus en *O*(1) grâce à la consultation de champs conservés dans les listes et mis à jour par les opérations de listes.

Quand une solution est en construction, certains arcs ne sont pas encore traités. On les distingue grâce à un attribut supplémentaire *done*(*u*) valant *true* si et seulement si *u* est collecté, ou *trip*(*u*) donnant l'index du trip qui traite *u* (0 si *u* n'est pas encore traité). Pour éviter de détailler les initialisations dans les algorithmes, on suppose qu'un *clear* sur une solution prépare une liste-solution vide (aucune tournée) et met *done* à *false* et *trip* à 0.

2.6 Liste récapitulative des notations

2.6.1 Réseau

G	graphe interne complètement orienté, modélisant le réseau réel

2.6.2 Nœuds

n	nombre de nœuds dans G
V	ensemble des nœuds de G, indexés de 1 à n
i, j	indices typiques pour les nœuds
$\Gamma(i)$	ensemble de successeurs du nœud i
$\Gamma^-(i)$	ensemble de prédécesseurs du nœud i
$\omega^+(i)$	ensemble des arcs débutant au nœud i
σ	index du nœud dépôt
dumpnodes	ensemble des nœuds de vidage
ndumps	nombre de nœuds de vidage
φ	index du nœud de vidage s'il n'y en a qu'un

2.6.3 Arcs

m	nombre d'arcs du graphe interne G.
A	ensemble des arcs du graphe interne G, indexés de 1 à m
u, v	indices typiques pour les arcs
$b(u)$	nœud de début de l'arc u
$e(u)$	nœud de fin de l'arc u
suc(u)	listes des arcs vers lesquels on peut tourner à partir de l'arc u
pre(u)	liste des arcs à partir desquels on peut tourner vers l'arc u
pen(u,v)	pénalité pour tourner de u vers v, $v \in$ *suc*(u)
inv(u)	arc inverse si u fait partie d'une arête à traiter (sinon *inv*(u) = 0)
$r(u)$	demande de l'arc u (0 pour les arcs qui ne sont pas à traiter)
$c(u)$	coût de traversée de l'arc
$w(u)$	coût de traitement de u (0 pour les arcs qui ne sont pas à traiter)
ε	nombre de tâches-arêtes
α	nombre de tâches-arcs
τ	nombre total de tâches, $\tau = \varepsilon + \alpha$
ρ	nombre d'arcs requis codant des tâches, $\rho = 2\varepsilon + \alpha = \varepsilon + \tau$
R	sous-ensemble des arcs requis, $\rho = \|R\|$

2.6.4 Boucle de dépôts et vidages

s	index de la boucle fictive pour le dépôt, $s = (\sigma, \sigma)$
dumploops	sous-ensemble des index des boucles fictives pour les exutoires
t	index de la boucle fictive si l'exutoire est unique, $t = (\varphi, \varphi)$
term(*u*)	coût minimal pour terminer une tournée après l'arc *u*
dump(*u*)	boucle fictive de l'exutoire correspondant à *term*(*u*)
δ	coût d'une opération de vidage

2.6.5 Véhicules

nva	nombre de véhicules disponibles
nvu	nombre de véhicules réellement utilisés, égal au nombre de tournées
Q	capacité commune des véhicules
L	autonomie des véhicules (coût maximal permis pour une tournée)

2.6.6 Distancier

D	matrice *m*×*m* des "distances" entre arcs (coûts des chemins optimaux)
d(*u*,*v*)	distance de l'arc *u* à l'arc *v* (*u* et *v* non inclus)
P	matrice *m*×*m* stockant un des chemins optimaux pour tout couple d'arcs
P(*u*,*v*)	arc prédécesseur de *v* sur le chemin optimal de *u* à *v*

2.6.7 Planification

H	horizon de planification
np	nombre de périodes ("jours") de l'horizon
p, *q*	indices habituels pour les jours
f(*u*)	fréquence de l'arc *u* (nombre de traitement sur l'horizon)
ns	nombres total de services sur l'horizon, somme des fréquences
prod(u,p)	production de l'arc *u* à la période *p* avec accumulation d'une période à l'autre
r(u,p)	demande de l'arc *u* pour une période *p*
c(u,p)	coût de l'arc *u* pour une période *p*

2.6.8 Solution et tournées

S	solution du CARP composé de *nvu* tournées, $S = (\theta_1, \theta_2, \ldots, \theta_{nvu})$.
θ	une tournée de la solution *S*
load	charge totale, fonction générique applicable à une tournée ou à une solution
cost	coût total, fonction générique applicable à une tournée ou à une solution

2.6.9 Traitement d'une tâche

done(*u*)	variable booléenne égale à vrai ssi l'arc requis *u* est traité
trip(*u*)	index de la tournée qui traite *u*, nul si *u* n'est pas encore traité

2.7 Conclusion

Nous avons présenté dans ce chapitre de transition la problématique de la thèse, le codage utilisé pour le réseau, et des notations. Le domaine d'application qui nous a motivée, la collecte des déchets ménagers, a été décrit dans ses grandes lignes. Les problèmes théoriques de la thèse (CARP, ECARP, SCARP, PCARP) ont été brièvement introduits, ils seront traités en détail dans les chapitres suivants. Le codage et les notations peuvent sembler indigestes à première vue, mais ils vont permettre ensuite une présentation homogène de tous les problèmes et algorithmes de la thèse.

Nous avons vu que la revue de la littérature sur le CARP a mis en évidence l'absence d'algorithmes évolutionnistes. Ce constat nous a motivé pour développer un algorithme génétique pour le CARP de base et pour l'étendre ensuite aux ECARP, PCARP et SCARP. Nous débutons à partir du prochain chapitre, consacré à notre algorithme génétique pour le CARP de base, la description de nos contributions.

CHAPITRE 3
Algorithmes génétiques pour le CARP

3.1 Introduction

Jusqu'à présent, les métaheuristiques publiées pour le CARP de base sont des méthodes tabou (Eglese et Li, 1996 ; Belenguer *et al*., 1997 ; Hertz et *al*., 2000), des recherches à voisinage variable (Mittaz, 1999) ou des méthodes de recuit simulé (Li, 1992 ; Eglese, 1994). L'algorithme le plus efficace est la méthode tabou *Carpet* de Hertz *et al*. Après avoir constaté l'absence de méthodes évolutionnistes, nous proposons dans ce chapitre les premiers algorithmes génétiques (GA) pour le CARP de base. Nos méthodes de résolution pour des extensions du CARP font l'objet des chapitres suivants.

Rappelons que le CARP de base est défini sur un réseau non orienté, avec un seul nœud-dépôt σ servant aussi d'exutoire. Avec nos conventions, ce réseau est codé comme un graphe orienté symétrique $G = (V,A)$, avec indexation des nœuds de 1 à n et des arcs de 1 à m. Chaque arête est codée par deux arcs opposés liés par un pointeur inv : $inv(u) = v$ et $u = inv(v)$. De plus, il n'y a aucune différence entre coûts de traversée et coûts de traitement. Sauf exceptions dûment spécifiés, le nombre de véhicules est une variable de décision.

Pour que ce chapitre soit auto-suffisant, nous décrivons d'abord le calcul du distancier et trois heuristiques constructives utilisées dans la population initiale. Nous rappelons ensuite les principes des algorithmes génétiques avant de présenter plusieurs composants possibles pour des GA adaptés au CARP. La démarche consiste à partir de composants de base et d'ajouter des raffinements progressifs pour améliorer la puissance des algorithmes. Nous expliquons ensuite le choix des composants et le réglage des paramètres ayant conduit à notre meilleur GA. Il s'agit en fait d'un algorithme dit *mémétique* (Moscato, 1999), c'est-à-dire qu'il est hybridé avec une recherche locale. Le chapitre se termine par une comparaison entre ce GA et Carpet sur trois bibliothèques de jeux d'essai de la littérature.

3.2 Calcul du distancier

Le calcul d'un distancier n'est pas si évident et mérite quelques explications. Dans le chapitre 2, nous avons vu que nous utilisons des distances arc à arc pour gérer des interdictions de tourner. Rappelons que $d(u,v)$ désigne le coût du plus court chemin (distance) de l'arc u à l'arc v (non inclus) et que $p(u,v)$ est l'arc précédant v sur ce plus court chemin.

Pour le CARP de base, sans interdictions de tourner, l'algorithme pour le distancier peut se contenter de calculer des chemins entre nœuds dans deux matrices *dist* et *back*, $n \times n$: *dist*(*i*,*j*) est la distance entre les nœuds *i* et *j* tandis que *back*(*i*,*j*) est le prédécesseur de *j* sur le chemin correspondant. Avec les notations du chapitre 2, on a donc $d(u,v) = dist(e(u),b(v))$ et $p(u,v) = (back(e(u),b(v)),b(v))$.

Les coûts de traversée positifs sur les arcs suggèrent d'utiliser l'algorithme 3.1 conçu par Dijkstra (Cormen *et al.*, 1990). Il calcule pour un nœud de départ *i* un plus court chemin de *i* vers tout autre nœud *j*, c'est-à-dire la ligne *i* du distancier. L'algorithme initialise d'abord les labels à l'infini, sauf *dist*(*i*,*i*) = 0. Un tableau de booléens *fix* indique pour chaque nœud si son label est fixé (définitif). Chaque itération cherche le nœud *j* non fixé de label minimal, le fixe, puis met à jour les labels des successeurs. Nous donnons une version valable pour un graphe fortement connexe, ce qui est le cas par hypothèse.

```
dist(i,i) := 0
for j := 1 to n do dist(i,j) := ∞; fix(j) := false endfor
for count := 1 to n do
   j      := argmin{dist(i,k): k∈V and fix(k)=false}
   fix(j) := true
   for all k in Γ(j) with dist(i,j) + c(j,k) < dist(i,k) do
      dist(i,k) := dist(i,j) + c (j,k)
      back(i,k) := j
   endfor
endfor
```

Algorithme 3.1 - *Algorithme de Dijkstra pour un graphe fortement connexe*

L'algorithme est en $O(n^2)$. On peut l'améliorer avec une structure de données appelée *tas* (Cormen *et al*., 1990). On construit en fait en $O(n)$ un tas de nœuds non fixés, avec celui de label minimal à la racine. *j* est extrait du tas en $O(\log n)$, mais la mise à jour de chaque successeur nécessite un ajustement du tas, en $O(\log n)$ également. Comme on met à jour $O(m)$ successeurs pendant l'algorithme, on obtient une complexité en $O(m.\log n)$, meilleure que $O(n^2)$ si *G* est peu dense. En particulier, on a $O(n.\log n)$ pour un réseau routier avec $m \approx 4n$.

L'algorithme doit être appelé pour *i* variant de 1 à *n* pour avoir le distancier complet, d'où une complexité totale en $O(n^3)$ pour la version sans tas, en $O(m.n.\log n)$ pour celle avec tas et $O(n^2.\log n)$ pour celle avec tas sur un réseau routier. Si *G* est peu dense, on constate donc qu'il n'y a aucun intérêt à utiliser l'algorithme de Floyd proposé dans la littérature pour les distanciers (Cormen *et al*., 1990) : sa complexité est en effet $O(n^3)$.

3.3 Heuristiques constructives

Pour initialiser la population dans nos algorithmes génétiques, nous avons sélectionné les heuristiques Path-Scanning (*PS*) et Augment-Merge (*AM*), proposées dans Golden et *al*. en 1983, ainsi que l'heuristique d'Ulusoy (*UL*) développée en 1985. Ces heuristiques sont en effet assez efficaces tout en étant facilement extensibles pour traiter des contraintes supplémentaires. Elles ont été très brièvement introduites dans la section 1.8.5 du chapitre 1. Nous

proposons ici des implémentations détaillées et non ambigües, ainsi que des analyses de complexité, tâches omises par les auteurs de ces heuristiques dans les articles de référence.

3.3.1 Path-Scanning

3.3.1.1 Principe

Path-Scanning de Golden et Wong est décrite de façon littéraire par ses auteurs, sans vrai algorithme ni mention de la complexité. On construit les tournées une par une. La tournée courante θ part du dépôt σ et est prolongée à chaque itération vers une arête à traiter compatible avec la capacité résiduelle du véhicule. Pour une tournée parvenue en un nœud *i*, on donne la priorité aux arêtes non encore traitées partant de *i*. S'il n'y en a plus, on va au plus proche nœud ayant des arêtes non traitées. Les arêtes possibles [*i*,*j*] sont départagées par un critère *F*. Les auteurs en proposent 5, expliqués plus loin. L'algorithme est exécuté 5 fois (une fois par critère) et on sort à la fin la meilleure solution.

3.3.1.2 Algorithme

Pour un critère *F* donné, nous donnons une description concise de Path-Scanning dans l'algorithme 3.2, avec les notations du chapitre 2 que nous rappelons pour la dernière fois. Les variables propres à l'algorithme sont la solution *S*, le nombre d'arcs traités *ndone* et *rmin*(*i*), la quantité minimale sur les arcs à traiter partant de *i*. Par convention, *rmin*(*i*) = ∞ si *i* n'a pas d'arcs à traiter.

Rappelons qu'une solution *S* est une liste de tournées, qui sont elles-mêmes des listes d'arcs requis reliés implicitement par des plus courts chemins. *S* cache aussi des indicateurs *done*(*u*) indiquant les arcs déjà traités, un coût *cost*(*S*) et une charge *load*(*S*). Coût et charge sont automatiquement mis à jour quand on ajoute ou on enlève des arcs.

L'heuristique commence par un *clear*(*S*) qui prépare une solution vide *S* et initialise le statut des arcs à « non traité » (*done*(*u*) = *false*). Elle calcule les *rmin*(*i*) et met *ndone* à 0. Puis, chaque itération du *repeat* construit une tournée courante θ qui est concaténée à *S*. Partant d'une tournée vide θ au nœud-dépôt, chaque itération du *loop* prolonge θ d'un arc.

On calcule d'abord le nœud *j* le plus proche de *i* ayant encore des arcs à traiter compatibles avec la charge du camion (si *rmin*(*j*) ne rentre pas dans le camion, aucun arc partant de *j* ne convient). Notez que l'algorithme trouve *j* = *i* si *i* a encore des arcs à traiter, puisque dans ce cas *dist*(*i*,*j*) = 0. Si *j* n'est pas trouvé, la tournée est terminée. Sinon, on cherche un arc *u* non encore traité partant de *j*, compatible avec la charge du camion, et minimisant le critère *F*. Cet arc existe d'après notre définition des *rmin*. On l'ajoute en fin de tournée et on note qu'il est traité, ainsi que son inverse. On met à jour *rmin*(*i*) car on peut repasser plus tard en *i*. Le nœud courant devient l'extrémité de *u*, *e*(*u*) avec nos notations.

Quand la tournée courante θ est finie, on la concatène à la solution avec la procédure *enqueue* et on compte dans *ndone* son nombre de tâches. L'algorithme se termine quand toutes les τ tâches sont traitées.

```
clear(S)
for all i in V do rmin(i) := min{r(u): u∈δR+(i)} endfor
ndone := 0
repeat
   θ := ∅
   i := σ
   loop
      j := argmin{dist(i,k): k∈V and rmin(k) ≤ (Q-load(θ))}
      exit when j = 0
      u := argmin{F(v): v∈δR+(j) and done(v)=false}
      done(u),done(inv(u)) := true
      enqueue(θ,u)
      rmin(j) := min{r(v): v∈δR+(j) and done(v)=false}
      i := e(u)
   endloop
   enqueue(S,θ)
   ndone := ndone + |θ|
until ndone = τ
```

Algorithme 3.2 – *Heuristique Path-Scanning*

Cet algorithme doit être exécuté en instanciant *F* par les fonctions simples suivantes :

$F_1(v) = -d(e(v),\sigma)$, maximisation de la distance au dépôt,
$F_2(v) = d(e(v),\sigma)$, minimisation de la distance au dépôt,
$F_3(v) = -r(v)/w(v)$, maximisation du rendement (quantité / coût de traitement),
$F_4(v) = r(v)/w(v)$, minimisation du rendement,
$F_5(v) =$ **if** $load(\theta) \leq Q/2$ **then** $F_5(v) := F_1(v)$ **else** $F_5(v) := F_2(v)$ **endif**

3.3.1.3 Complexité de la version de base

La fonction *clear* qui met *done* à *false* coûte $O(n)$. Le calcul de tous les *rmin* est possible en $O(\tau)$. La boucle *loop* range les tâches dans les tournées et fait donc en tout $O(\tau)$ itérations. Les calculs de *j* est en $O(n)$. Celui de *u* et la mise à jour des $rmin(i)$ sont aussi en $O(n)$ car pour tout nœud *j* on a $|\delta_R^+(j)| \leq n$. Le corps du *loop* est donc en $O(n)$, d'où un algorithme en $O(\tau.n)$. Ces complexités deviennent respectivement $O(m.n)$ si tous les arcs sont requis, $O(n^2)$ si *G* est de type routier et $O(n^3)$ si *G* est complet.

3.3.2 Augment-Merge

3.3.2.1 Principe

Cette heuristique ressemble à la fameuse méthode de la marguerite proposée par Clarke et Wright (1964) pour le VRP. Il s'agit encore d'un algorithme de Golden et Wong, pour lequel les auteurs donnent une description laborieuse et sans complexité. Voici son principe pour le CARP de base, non orienté (voir aussi la figure 3.1). Elle part d'une solution triviale de τ tournées réduites chacune à une tâche.

Dans la phase dite *Augment*, on cherche à éliminer les petites tournées dont l'unique tâche est traversée par une tournée plus grande : si la capacité résiduelle du camion le permet, la grande tournée peut collecter cette tâche en passant, absorbant ainsi la petite tournée. Pour de meilleurs résultats, Golden et Wong préconisent de trier les tournées initiales par coûts décroissants puis, pour toute tournée S_i dans l'ordre du tri, de tenter d'absorber S_{i+1}, S_{i+2} …S_τ, si S_i passe sur l'unique tâche-arête de ces tournées et si *load*(S_i) le permet.

Ces absorptions ne changent pas le coût de S_i puisque coûts de traitement et coûts de traversée sont égaux. Notons que pour une tournée en cours d'augmentation S_i, les tournées S_1 à S_i peuvent avoir plusieurs tâches, tandis que celles à partir de S_{i+1} n'en ont toujours qu'une.

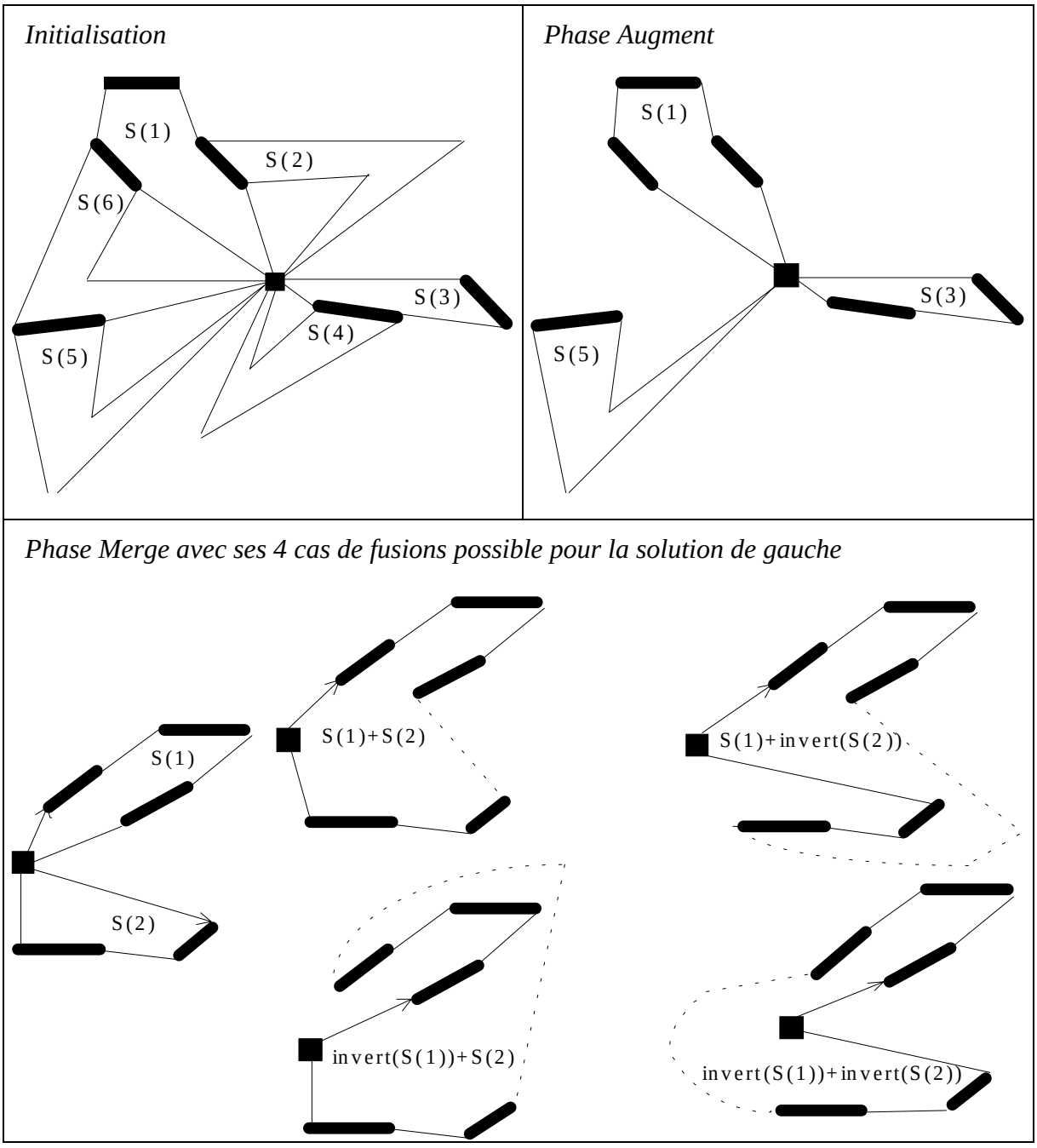


Figure 3.1 *– Heuristique Augment-Merge*

3.3.2.2 Algorithme

Nous proposons une version conceptuelle de l'heuristique dans l'algorithme 3.3. Nous rappelons qu'avec nos notations de listes, $S(i,j)$ désigne la tâche de rang j dans la tournée i de la solution S. Pour écrire simplement les coûts des fusions, on utilise un distancier arc à arc. Les tournées gardent leurs rangs pendant l'algorithme : quand une tournée $S(k)$ est supprimée dans la solution, on la met donc à vide plutôt que de faire un *remove* qui décrémenterait le rang des tournées suivantes. Cette façon de faire est importante pour une bonne complexité. Le 1er bloc (*initialisation*) construit la marguerite avec une tournée par arête puis trie les tournées par coûts décroissants. Notons que c'est la liste-tournée (u) et non pas l'arc u qui est ajouté à S. Le 2ème bloc (phase *Augment*) considère chaque tournée $S(i)$ en partant de la plus grande et tente d'absorber les tournées suivantes.

```
// Initialisation
clear(S)
for u := 1 to m with done(u) = false do
  enqueue(S,(u))
  done(u),done(inv(u)) := true
endfor
trier S en ordre décroissant des coûts des tournées

// Phase Augment
for i := 1 to |S| with S(i)≠∅ do
  for j := i+1 to |S| with S(j)≠∅ and load(S(i))+load(S(j)) ≤ Q do
    if S(j,1) est insérable à coût nul dans S(i) à un certain rang k then
       insert(S(i),k,S(j,1))
       S(j) := ∅
    endif
  endfor
endfor

// Phase Merge
loop
  vmin := 0
  for i := 1 to |S| with S(i)≠∅ do
    for j := i+1 to |S| with S(j)≠∅ and load(S(i))+load(S(j)) ≤ Q do
      bi := S(i,1); ei := last(S(i));
      bj := S(j,1); ej := last(S(j))
      var := d(ei,bj) - d(ei,s) - d(s,bj)
      if v < vmin then vmin := v; kind := (+i,+j) endif
      var := d(ei,inv(ej)) - d(ei,s) - d(ej,s)
      if v < vmin then vmin := v; kind := (+i,-j) endif
      var := d(inv(bi),bj) - d(s,bi) - d(s,bj)
      if v < vmin then vmin := v; kind := (-i,+j) endif
      var := d(inv(bi),inv(ej)) - d(s,bi) - d(ej,s)
      if v < vmin then vmin := v; kind := (-i,-j) endif
    endfor
  endfor
  exit when vmin = 0
  i := abs(kind(1)); j := abs(kind(2))
  if kind(1) < 0 then S(i) := inverse(S(i)) endif
  if kind(2) < 0 then S(j) := inverse(S(j)) endif
  S(i) := S(i) + S(j)
  S(j) := ∅
endloop
```

Algorithme 3.3 – *Heuristique Augment-Merge*

Chaque itération du 3ème bloc (*Merge*) considère les couples (*i*,*j*) de tournées distinctes et évalue dans *v* les variations de coût des fusions possibles. On peut concaténer *S*(*i*) puis *S*(*j*) en inversant ou pas chaque tournée, d'où 4 cas. Il suffit de traiter les couple (*i*,*j*) avec $i < j$ car les coûts d'une tournée et de son inverse sont égaux : ainsi, concaténer *S*(*i*) et *S*(*j*) équivaut à concaténer l'inverse de *S*(*j*) et l'inverse de *S*(*i*). Les calculs de variations sont plus lisibles en introduisant les arcs *bi* et *ei* (*resp*. *bj* et *ej*) débutant et terminant la tournée *i* (*resp*. *j*).

La meilleure fusion de variation négative est stockée avec *vmin* et le couple *kind* des indices des deux tournées, multipliés par –1 quand elles doivent être inversées : par exemple, *kind* = (*i*,-*j*) signifie une fusion concaténant la tournée *i* puis l'inverse de la tournée *j*. Après évaluation de toutes les paires, chaque itération de *Merge* effectue la meilleure fusion trouvée, en réutilisant l'indice *i* pour stocker la nouvelle tournée. L'algorithme se termine si aucune fusion n'est possible à cause des charges ou si les fusions possibles ne procurent plus de gains.

3.3.2.3 Analyse de complexité

a) Version de l'algorithme 3.3

Golden et Wong ne mentionnent pas la complexité. Pearn (1989) prétend sans aucune justification qu'elle est en $O(n^3)$. En fait, seuls τ arcs parmi *m* (un par tâche) forment les tournées initiales. L'initialisation (tri) peut se faire en $O(\tau.\log \tau)$. Dans *Augment*, pour *S*(*i*) et *S*(*j*) fixées, la tournée *S*(*i*) peut vite contenir plusieurs tâches et il faut tester $|S(i)|+1 = O(\tau)$ positions d'insertion. La phase *Augment* balaie $O(\tau^2)$ paires de tournées et coûte donc $O(\tau^3)$. La phase *Merge* effectue au pire τ-1 fusions et calcule pour chacune les variations de coût pour $O(\tau^2)$ paires de tournées, elle est aussi en $O(\tau^3)$. La complexité totale est donc $O(\tau^3)$, c'est-à-dire $O(m^3)$ si tous les arcs sont requis, $O(n^6)$ si *G* est complet et $O(n^3)$ si *G* est routier.

b) Version accélérée

Pour améliorer *Augment*, avant de tenter d'absorber dans *S*(*i*) les tournées suivantes, on convertit *S*(*i*) en une liste complète d'arcs (tâche de la tournée + arcs des chemins en haut-le-pied) et on construit en même temps une table *trav*, avec *trav*(*u*) = la première position dans la liste de l'arc *u* s'il est traversé par la tournée, 0 sinon. Cette préparation se fait une seule fois pour tout *S*(*i*) et coûte *O*(*m*) car il n'y a que deux chemins HLP, élémentaires. Ensuite, pour chaque tournée *S*(*j*), l'absorption est possible si $k = trav(S(j,1)) \neq 0$, et il suffit d'insérer en position *k*. Quand *S*(*i*) ne peut plus rien absorber, on élimine en *O*(*m*) les arcs restés en HLP. L'introduction temporaire des arcs HLP est nécessaire pour avoir des positions d'insertion fixes pendant le balayage des *S*(*j*). On obtient une phase *Augment* en $O(\tau.(m+\tau)) = O(m.\tau)$.

Une remarque-clé pour améliorer *Merge* est que les variations de coût sont recalculées pour toutes les paires de tournées avant chaque fusion, alors qu'une seule fusion va être effectuée. En fait, on peut calculer au début une liste des $O(\tau^2)$ fusions possibles avec leurs variations de coûts et la trier en $O(\tau^2.\log \tau^2) = O(\tau^2.\log \tau)$. Ensuite, il « suffit » d'effectuer en $O(1)$ chaque fusion dans l'ordre de la liste, en ignorant les fusions devenues impossibles. Pour cela, on stocke dans la liste chaque fusion avec les arêtes *u* et *v* concernées aux extrémités des tournées, et on définit pour tout arc *u* le numéro *trip*(*u*) de sa tournée actuelle. Ce travail peut être fait avant le tri, en $O(\tau)$ pour l'ensemble des tâches. On sait alors qu'une fusion (*u*,*v*) sortie de la liste est toujours possible si (rappelons que *trip*(*u*) est l'index de la tournée qui traite *u*) :

- $load(S(trip(u)))+load(S(trip(v))) \leq Q$
- u et v sont toujours aux extrémités de $S(trip(u))$ et $S(trip(v))$

Pour faire ces tests en $O(1)$, il faut accéder en $O(1)$ à une tournée de rang donné, ce qui est possible car nous gérons la solution S comme un tableau de τ tournées : toutes les tournées gardent donc leur rang. $O(\tau)$ fusions vont être effectuées sur les $O(\tau^2)$ possibles. Pour chacune d'elles, on réutilise la tournée $S(trip(u))$. Il faut faire $trip(z) := trip(u)$ pour toute tâche z de $S(trip(v))$, ce qui coûte $O(\tau)$, puis inverser éventuellement $S(trip(u))$ ou $S(trip(v))$ en $O(\tau)$, et enfin les concaténer dans $S(trip(u))$ en $O(1)$. On obtient ainsi une version de *Augment-Merge* en $O(\tau^2.\log \tau)$ à cause du tri des fusions possibles. Cette complexité devient $O(m^2.\log n)$ si tous les arcs sont requis, $O(n^4.\log n)$ si G est complet, et $O(n^2.\log n)$ pour un graphe routier, ce qui est un peu moins bon que le $O(n^2)$ de *Path-Scanning*).

3.3.3 Algorithme d'Ulusoy

3.3.3.1 Principe

Comme Golden et Wong, Ulusoy donne une description littéraire et sans complexité de son algorithme, avec une évaluation numérique réduite à quelques exemples. Nous proposons donc notre propre analyse de cette heuristique qui, nous le verrons, est *très* efficace. L'heuristique d'Ulusoy nécessite une séquence θ contenant les τ tâches, qui peut être vue comme un « tour géant » effectué par un véhicule de capacité infinie. Ce tour géant est ensuite découpé optimalement en tournées réalisables pour le CARP.

Si toutes les arêtes sont à traiter, le calcul du tour géant revient à résoudre un *problème de postier chinois non orienté* (UCPP, *cf.* chapitre 1), polynomial. Sinon, on a affaire à un *problème de postier rural non orienté* (URPP), NP-difficile. Comme une solution optimale du URPP ne donne pas nécessairement une solution optimale du CARP, une résolution avec n'importe quelle heuristique pour le URPP suffit. Nous appelons en fait Path-Scanning avec une capacité infinie et ses 5 critères pour construire 5 séquences différentes. Nous découpons ensuite ces 5 séquences avec la méthode d'Ulusoy et retournons la meilleure solution. Cette technique donne de très bons résultats.

La méthode d'Ulusoy découpe optimalement θ en calculant un plus court chemin dans un graphe auxiliaire valué $H = (X,U,Z)$. X comprend $\tau + 1$ nœuds indexés de 0 à τ. Le nœud 0 est un nœud de départ. Tout arc (i,j) de U représente une tournée réalisable traitant les tâches $\theta(i+1)$ à $\theta(j)$ incluses, et la valeur $z(i,j)$ sur l'arc est le coût d'une telle tournée. Un plus court chemin entre les nœuds 0 et τ dans H fournit un découpage de θ en tournées réalisables minimisant le coût total. On peut détailler les tournées en remontant le chemin et en remplaçant chaque arc par la sous-séquence correspondante de θ.

La figure 3.2 donne un exemple avec un tour géant initial θ de 5 tâches et des quantités entre parenthèses, en supposant une capacité $Q = 9$. Le plus court chemin dans le graphe auxiliaire est calculé avec l'algorithme de Bellman. Il donne un découpage optimal (sous contrainte de la séquence imposée par θ) en trois tournées et un coût total de 141.

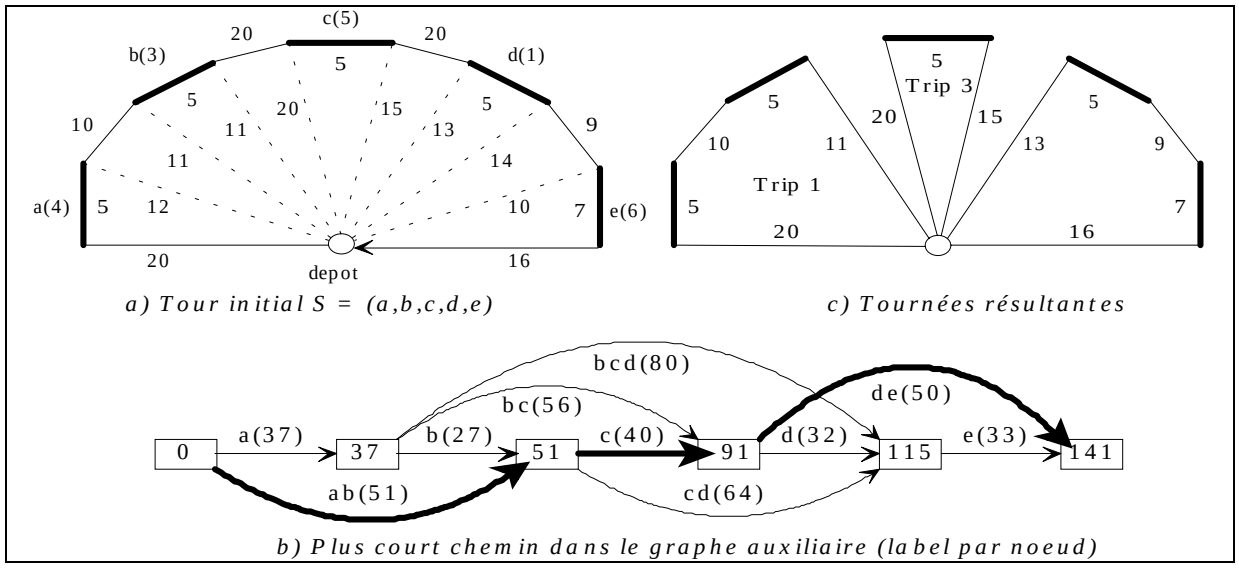


Figure 3.2 - *Principe de l'heuristique d'Ulusoy*.

3.3.3.2 Algorithme

Notre algorithme 3.4 donne une procédure *Split* qui évite une génération explicite du graphe auxiliaire *H* et minimise en plus le nombre de véhicules en critère secondaire. Trois labels sont utilisés pour chaque nœud *j* de *H* : V_j, le coût du plus court chemin de 0 à *j* dans *H*, N_j, le nombre d'arcs de ce chemin (nombre de tournées ou de véhicules) et B_j, le prédécessur de *j* sur ce chemin. Pour un tour géant θ, l'algorithme énumère toutes les tournées faisables (θ_i, …, θ_j) et calcule leurs charges et leurs coûts comme expliqué en 2.5.4. Au lieu de créer un arc (*i*-1, *j*) pour chaque tournée comme dans 3.3.3.1, les labels de *j* sont directement mis à jour. A la fin, le coût total et le nombre de véhicules utilisés *nvu* sont définis par V_τ *et* N_τ.

```
V(0),N(0) := 0
for i := 1 to τ do V(i) := ∞ endfor
for i := 1 to τ do
   load, cost := 0; j := i
   repeat
      load := load + r(θ(j))
      if i = j then
         cost := D(s,θ(i)) + w(θ(i)) + D(θ(i),s)
      else
         cost := cost - D(θ(j-1),s) + D(θ(j-1),θ(j)) + w(θ(j)) + D(θ(j),s)
      endif
      if load ≤ Q then
         VNew := V(i-1) + cost
         if (VNew < V(j)) or ((VNew = V(j)) and (N(i-1) + 1 < N(j)) then
            V(j) := VNew
            N(j) := N(i-1) + 1
            B(j) := i-1
         endif
         j := j + 1
      endif
   until (j > τ) or (load > Q)
endfor
```

Algorithme 3.4 *– Procédure de découpage Split pour l'heuristique d'Ulusoy*.

Une solution *S* pour le CARP peut être obtenue avec l'algorithme 3.5 qui remonte les arcs du plus court chemin grâce aux labels B_j. La tournée ξ associée à l'arc courant est extraite de θ puis insérée *en tête* de *S* pour respecter l'ordre du tour géant θ (procédure *push*). Remarquons la grande compacité du processus grâce à nos notations de listes.

```
clear(S)
j  := τ
repeat
   i := B(j)
   ξ := copy(θ,i+1,j)
   j := i
   push (S,ξ)
until i = 0
```

Algorithme 3.5 – *Heuristique d'Ulusoy : extraction de la solution S.*

3.3.3.3 Complexité

La complexité du calcul du tour géant dépend de l'heuristique utilisée. Nous analysons donc seulement la phase de découpage. Pour un tour géant θ de τ tâches, le graphe *H* a $O(\tau^2)$ arcs. L'algorithme de Bellman a pour complexité le nombre d'arcs du graphe, il est donc aussi en $O(\tau^2)$. L'extraction de la solution coûte $O(\tau)$. L'algorithme global est donc en $O(\tau^2)$, c'est à dire $O(m^2)$ si tous les arcs sont requis, $O(n^4)$ si *G* est complet, et $O(n^2)$ si *G* est de type routier.

3.4 Rappels sur les algorithmes génétiques

3.4.1 Introduction

Les premiers travaux sur les algorithmes génétiques (GA) ont commencé dans les années 50 lorsque plusieurs biologistes américains ont simulé des structures biologiques sur ordinateur. Puis, entre 1960 et 1970, Holland (1975) développa sur la base de ces travaux les principes fondamentaux des algorithmes génétiques dans le cadre de l'optimisation mathématique. Malheureusement, les ordinateurs de l'époque n'étaient pas assez puissants pour utiliser ces algorithmes sur des problèmes réels de grande taille.

L'ouvrage de Goldberg (1989b), en montrant comment appliquer les GA à des problèmes concrets, a permis de les populariser et a impulsé de nouveaux travaux. Les GA ont été depuis utilisés avec un succès croissant en optimisation combinatoire, voir par exemple la synthèse de Reeves (1996) et celle en français de Fleurent et Ferland (1996). Parmi d'autres articles de référence, citons par exemple Davis (1991), Dejong et Spears (1989), Grefenstette *et al*.(1985) et Michalewicz (1992).

Sur le plan théorique, les résultats sont apparus très tardivement et il faudra attendre 1993 pour qu'une démonstration complète et rigoureuse de convergence stochastique soit établie par Cerf (1994). Néanmoins, ces résultats théoriques sont difficilement exploitables dans la pratique. La performance pratique des algorithmes génétiques repose le plus souvent sur une approche empirique du problème et fait beaucoup appel au savoir-faire du concepteur.

3.4.2 Principe

Les algorithmes génétiques tentent de simuler le processus d'évolution des êtres vivants. Ils utilisent un vocabulaire similaire à celui de la génétique naturelle, bien que les processus naturels auxquels ils font référence soient beaucoup plus complexes. On parlera ainsi d'*individus* dans une *population*. Pour un problème donné, un individu représente une solution, à laquelle on associe une mesure d'adaptation (*fitness*). Un individu est composé d'un ou plusieurs *chromosomes*, eux-mêmes constitués de *gènes* qui contiennent les caractères héréditaires de l'individu. Pour un problème d'optimisation combinatoire, un individu est simplement une solution et son fitness est le coût de cette solution (critère à optimiser).

Les principes de *sélection*, *croisement* et *mutation* s'appuient sur les processus naturels de même nom. La population est codée en général comme un tableau d'individus. La sélection a pour but de choisir deux *individus-parents* dans la population, en favorisant les plus adaptés. Le croisement ou *crossover* combine les individus choisis pour former un ou deux nouveaux individus appelés *enfants*. C'est lui qui assure l'*exploration* de l'espace des solutions. La mutation modifie aléatoirement et avec une faible probabilité les enfants. Elle apporte une certaine *diversification* visant à éviter une convergence prématurée.

Ces opérations peuvent être agencées selon deux modes de gestion de la population. Dans le mode *générationnel*, chaque itération reproduit massivement tous les individus et range les enfants obtenus dans un autre tableau, qui devient la population courante pour la génération suivante. Dans le mode *incrémental*, deux parents seulement se reproduisent et leurs enfants remplacent aussitôt des individus de la population courante. Les deux modes stoppent quand certains critères d'arrêt sont respectés, par exemple un nombre maximal d'itérations. L'algorithme 3.6 donne la structure générale typique d'un GA incrémental. Le même processus est représenté graphiquement dans la figure 3.3.

```
Générer la population initiale (exemple 50 solutions)
repeat
   Sélection :
   Tirer au sort 2 parents en favorisant les meilleurs (les moins coûteux)
   Reproduction :
   Appliquer une procédure de croisement pour obtenir des solutions-enfants
   Mutation (avec une faible probabilité) :
   Modifier aléatoirement les enfants
   Remplacement :
   Remplacer des solutions dans la population par les enfants
   (en préservant la meilleure solution !)
until un certain critère d'arrêt.
```

Algorithme 3.6 - *Structure générale d'un algorithme génétique incrémental.*

Pour concevoir un algorithme génétique sur un problème d'optimisation particulier, on doit donc disposer des cinq éléments suivants :

- un codage de chaque solution sous forme de chromosome. Cette étape est cruciale car elle va permettre des opérateurs de croisement et de mutation plus ou moins efficaces.

- un mécanisme de génération de la population initiale, capable de produire une population d'individus non homogène qui servira de base pour les générations futures. Le choix de la population initiale est important car il peut rendre plus ou moins rapide la convergence vers l'optimum global. Dans le cas où on ne connaît rien du problème à résoudre, il est essentiel que la population initiale soit bien dispersée dans l'espace de recherche.

- une mesure du fitness de chaque individu, en général le coût de la solution associée.

- des opérateurs de croisement et de mutation. L'opérateur de croisement brasse des gènes existant déjà dans la population, L'opérateur de mutation a par contre pour but de générer de nouveaux gènes chez les individus de la population (apport de « sang neuf »).

- des paramètres de réglage, comme la taille de la population, les critères d'arrêt, les probabilités d'application des opérateurs de croisement et de mutation.

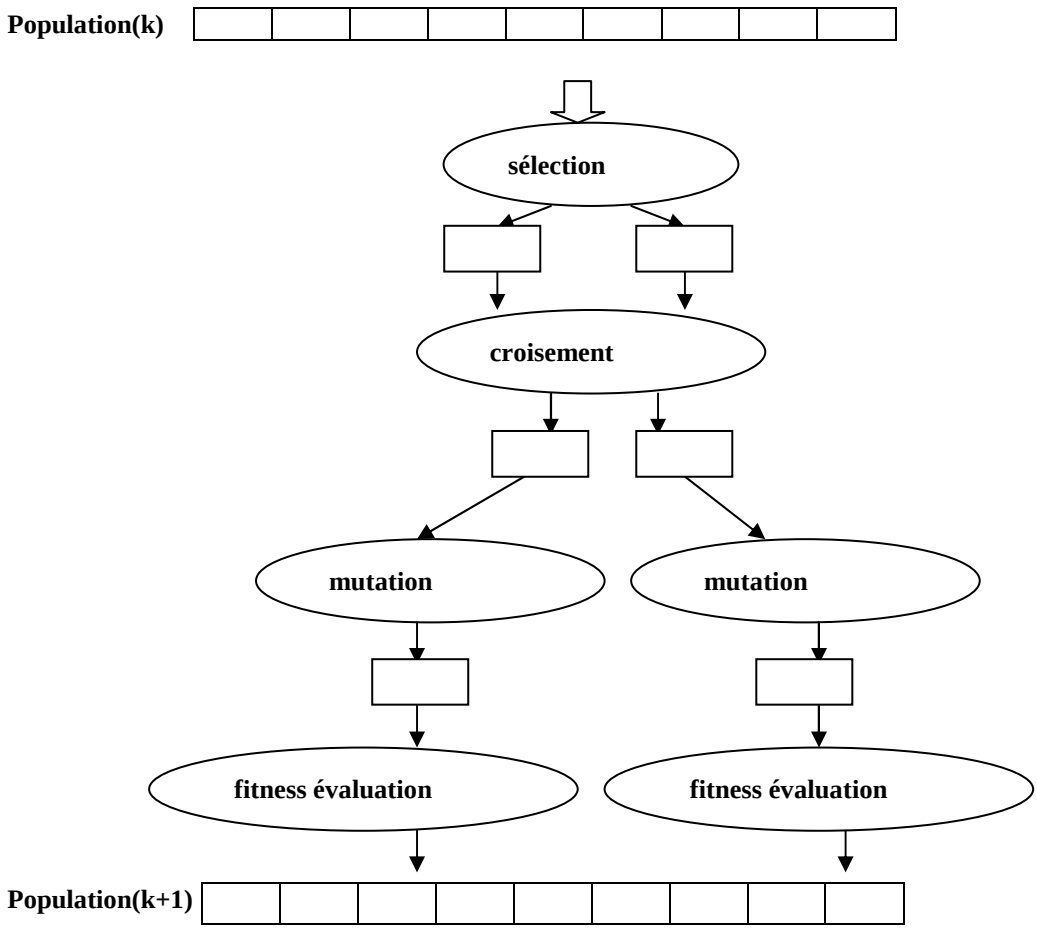


Figure 3.3 *– Itération de base d'un algorithmes génétique incrémental.*

3.4.3 Résultats théoriques

Les algorithmes génétiques ont montré leur efficacité pratique bien avant que des résultats de convergence théorique soient établis. Aujourd'hui, on connaît trois approches théoriques différentes pour mieux comprendre le fonctionnement asymptotique des GA :

- la théorie des schémas de Goldberg (1989a), qui concerne des chromosomes codés par des chaînes de bits. Elle étudie le comportement asymptotique de l'algorithme et l'effet des différents opérateurs sur des sous-chaînes appelées schémas.

- des techniques développées à l'origine pour le recuit simulé (Van Laarhoven et Aarts, 1987). Sous certaines hypothèses, on a montré que l'algorithme génétique converge en un temps infini, grâce à l'opérateur de mutation. Le rôle de l'opérateur de croisement reste ignoré, ce qui n'est pas satisfaisant puisqu'on réduit le GA à une sorte de marche aléatoire.

- l'approche théorique la plus récente est proposée dans les travaux de Cerf (1994), qui utilise la théorie de Freidlin et Wentzell (1983) pour étudier les états stables de l'algorithme génétique, processus Markovien. Cerf met notamment en évidence l'intérêt de l'opérateur de croisement au travers d'un algorithme génétique utilisant une méthode de sélection particulière qui assure la diversité de la population.

3.4.4 Algorithmes génétiques publiés pour des problèmes de tournées

Concernant les problèmes de tournées de véhicules, des algorithmes génétiques ont été développés pour les tournées sur nœuds mais pas pour les tournées sur arcs. Les premiers travaux ont concerné le problème mono-véhicule du voyageur de commerce : Goldberg (1989b), Grefenstette et *al*.(1985), Muhlenbein et *al* (1988) et Starkweather et *al* (1991). Concernant les problèmes multi-véhicules, aucun GA ne semble compétitif avec les meilleures méthodes tabou publiées pour le VRP. Par contre, il existe des algorithmes génétiques très efficaces pour des extensions comme le VRP avec fenêtres de temps (VRPTW) : Potvin et Bengio (1993), Blanton et WainWright (1993), Thangiah (1993). Plus récemment, Filipec et *al*. (2000) a traité un VRP multi-dépôt.

Ayant constaté le manque d'algorithmes évolutionnistes pour les tournées sur arcs, notre but a été naturellement de concevoir le chaînon manquant sous forme d'algorithmes génétiques pour le CARP, suffisamment flexibles pour supporter des généralisations progressives vers des problèmes plus réalistes.

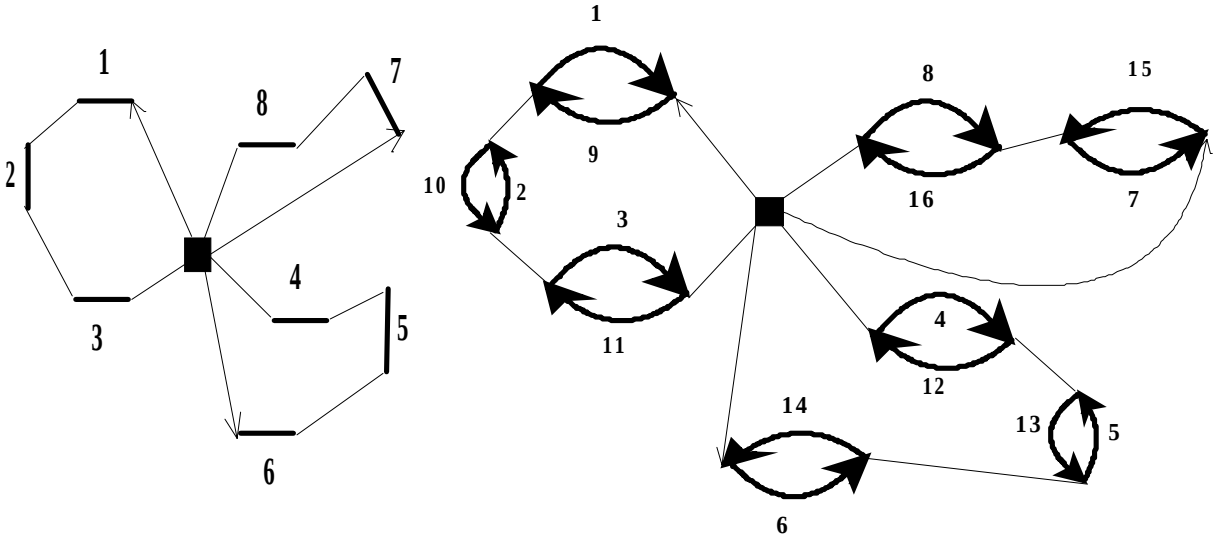
Dans ce qui suit, nous présentons les caractéristiques essentielles de nos GA : les chromosomes et leurs évaluations, les croisements et mutations étudiés, etc. Ces caractéristiques peuvent se décliner en différent GA selon l'implémentation de la population initiale, la gestion des générations, et les critères d'arrêt.

3.5 Composants pour des GA pour le CARP

3.5.1 Chromosomes : représentations et évaluations

La conception d'algorithmes génétiques pour le CARP nécessite en premier lieu un bon codage des chromosomes. Le GA initial de Holland utilise des chromosomes *binaires*. Comme dans beaucoup de GA pour des problèmes de séquencement, nous préférons utiliser une représentation plus adaptée, sous forme de *listes de tâches*. En fait, nos chromosomes sont quasiment des *permutations* des τ tâches du réseau, sauf qu'une tâche-arête peut y figurer comme un de ses deux arcs opposés. Deux tâches successives sont implicitement reliées par des plus courts chemins. Il s'agit donc d'un codage très compact, de taille $O(\tau)$.

Toute solution réalisable du CARP déjà calculée par un algorithme quelconque peut être facilement convertie en chromosome, en concaténant les listes de tâches des tournées. Nous proposons en fait deux codages différents : un qui contient des délimiteurs de tournées et un autre sans délimiteurs de tournées. Ces deux formats de chromosomes vont nécessiter deux méthodes d'évaluation complètement différentes. Dans ce qui suit, un chromosome *S* est traité comme une liste d'index de tâches et *F*(*S*) désigne son fitness.



Chromosome avec délimiteurs de tournées : 9, 10, 3 - 6, 5, 12 - 15, 16
Chromosome sans délimiteurs de tournées : 9, 10, 3, 6, 5, 12, 15, 16

Figure 3.4 *– Les deux types de chromosomes pour le CARP*

La figure 3.4 montre les deux types de chromosomes pour un CARP de 8 tâches-arêtes, en trait épais. Les demandes valent toutes 1 et $Q = 3$. Les traits fins sont des plus courts chemins dans le graphe complet, non détaillé. Le dépôt *s* est le point de jonction central. Les flèches indiquent le sens de parcours des tournées. Chaque arête est codée par deux arcs internes. On suppose que les traversées d'arêtes de gauche à droite ou de bas en haut correspondent aux arcs 1 à 8 et les autres sens aux arcs 9 à 16. De plus, pour tout *u* entre 1 et 8, on a supposé que $inv(u) = u + 8$: Par exemple, l'arête la plus haute est codée par les arcs internes 7 et 15. Les tirets dans les chromosomes indiquent les délimiteurs de tournées.

3.5.1.1 Chromosome avec délimiteurs de tournées

C'est une idée très naturelle, qui consiste à concaténer des sous-chromosomes (un par tournée), séparés par des symboles spéciaux appelés *délimiteurs de tournées*. Cette technique a déjà été utilisée par Potvin et Bengio (1996) pour le VRP avec contrainte de fenêtres de temps. Elle est connue pour nécessiter des *procédures de réparation* quand les croisements tentent de conserver des délimiteurs provenant des parents : des sous-chromosomes violant la capacité des véhicules peuvent apparaître.

De tels chromosomes peuvent être obtenus à partir d'une solution déjà calculée, par génération aléatoire (dans la population initiale), par croisement ou par mutation. Dans tous les cas, on génère en fait d'abord des séquences de tâches sans délimiteurs. Ainsi, pour une solution existante, on concatène les tournées sans inclure de délimiteurs. Pour une génération aléatoire, on tire au sort une permutation des τ tâches-arêtes puis on tire à pile ou face pour chaque arête un des deux arcs qui la codent. Quant aux croisements et mutations proposés en 3.5.2 et 3.5.3, ils sont conçus pour ignorer purement et simplement les délimiteurs présents dans les parents et ils produisent aussi des séquences sans délimiteurs.

Les délimiteurs sont ensuite insérés par la *procédure d'évaluation séquentielle* très simple en $O(\tau)$ de l'algorithme 3.7. Partant d'une séquence de tâches S sans délimiteurs, elle construit une version T avec délimiteurs et calcule son fitness $F(T)$. S est simplement balayée de gauche à droite et on change de tournée à chaque fois que la capacité du véhicule est atteinte. L'identificateur *delim* désigne le symbole-délimiteur.

```
i := 1
j := 1
F := 0
T := ∅
repeat
   while (j ≤ |S|) and (load(copy(S,i,j)) ≤ Q) do j := j + 1 endwhile
   F := F + cost(copy(S,i,j-1))
   T := T + copy(S,i,j-1) + delim
   i := j
until i > |S|.
```

Algorithme 3.7 – *Procédure d'évaluation séquentielle des chromosomes.*

La pose des délimiteurs par la procédure d'évaluation offre deux avantages. Premièrement, on obtient toujours des chromosomes réalisables, ce qui évite le recours à des procédures de réparation comme dans Potvin et Bengio (1996). Deuxièmement, on a une valeur unique du fitness quelle que soit l'origine du chromosome.

3.5.1.2 Chromosome sans délimiteurs de tournées et évaluation

Le deuxième format de chromosome ne contient aucun délimiteur. Clairement, un tel chromosome ne représente pas directement une solution du CARP mais peut être vu comme un tour géant ignorant la capacité des véhicules. Nous utilisons en fait la procédure de découpage *Split* de l'heuristique d'Ulusoy, vue dans l'algorithme 3.4, pour calculer le fitness.

Rappelons que *Split* calcule la meilleure solution possible du CARP, sous contrainte d'un ordre imposé des τ tâches. Cette procédure d'évaluation est plus coûteuse que dans le cas avec délimiteur : $O(\tau^2)$ au lieu de $O(\tau)$, *cf.* 3.3.3. Mais elle a les deux précieuses propriétés suivantes, très rarement respectées dans les algorithmes génétiques publiés :

Propriété 3.1. Tout chromosome est *évalué optimalement*, sous contrainte de sa séquence.

Propriété 3.2. Il existe au moins *un chromosome optimal*, c'est-à-dire fournissant après évaluation une solution optimale du CARP. Pour s'en convaincre, considérons une solution optimale du CARP et concaténons ses tournées : par définition, *Split* va être capable de re-découper optimalement le chromosome obtenu.

Grâce à ces deux propriétés et à son parallélisme intrinsèque, on peut donc espérer qu'un GA va être capable de trouver une solution optimale.

3.5.1.3 Illustation de la flexibilité de Split : minimisation du makespan

La technique de découpage de *Split* est basée sur le calcul d'un plus court chemin dans un graphe auxiliaire, *cf.* 3.3.3.1. Elle peut être adaptée à d'autres fonctions-objectifs, comme dans l'algorithme 3.8 qui concerne un CARP très réaliste que nous avons mis en évidence lors d'interviews avec des industriels de la collecte des déchets. La flotte est cette fois limitée, avec *nva* véhicules. Le nombre véhicules utilisés (et donc de tournées) *nvu* est non fixé mais ne doit pas dépasser *nva*. Les coûts sont des temps et l'objectif est de minimiser le *makespan* ou durée de la plus longue tournée.

Ce problème se rencontre quand la collecte doit se terminer le plus tôt possible pour affecter ensuite les équipages à d'autres tâches, comme un tri manuel des déchets collectés. Notons que le problème a une solution triviale si $nva \geq \tau$, en traitant chaque tâche avec une tournée séparée. L'algorithme utilise des labels V_j comme dans l'algorithme 3.4 du 3.3.3.2 mais nécessite un graphe auxiliaire *H* généré explicitement (voir 3.3.3). Il calcule un chemin min-max du nœud 0 au nœud τ dans *H*. Z_{ij} est le poids de l'arc (*i*,*j*) dans *H*.

```
K := 0
V(0),V2(0),N(0),N2(0) := 0
for i := 1 to τ do V(i),V2(i) := ∞ endfor

repeat
   K      := K+1
   stable := true
   for i := 0 to τ-1 do
      for each successor j of i in H with max(V(i),Z(i,j)) < V2(j) do
         V2(j)  := max(V(i),Z(i,j))
         N2(j)  := N(i)+1
         stable := false
      endfor
   endfor
   V := V2
   N := N2
until stable or (K = nva)
```

Algorithme 3.8 – *Version de Split pour la minimisation du makespan et une flotte limitée.*

Chaque itération de la boucle *repeat* calcule en $O(\tau^2)$ dans H les chemins max-min partant du nœud 0 et ayant au plus K arcs. Ils correspondent à des solutions du CARP avec au plus K tournées. Les labels qui sont améliorés à partir des labels courants de V et N sont rangés dans V_2 et N_2. Ces derniers tableaux sont copiés dans V et N à la fin de l'itération. L'algorithme s'arrête quand tous les labels sont stables (test du booléen *stable*) ou quand $K = nva$. Si à ce moment $V_\tau = \infty$, le chromosome est infaisable. Sinon, le makespan minimal pour S et le nombre de véhicules utilisés sont données respectivement par V_τ et N_τ.

Chaque itération du *repeat* revient à énumérer tous les arcs de H, en nombre $O(\tau^2)$ d'après 3.3.3.3. Un plus court chemin du nœud 0 au nœud τ dans H a au plus $\min(\tau,nva)$ arcs. L'algorithme est donc en $O(\min(\tau,nva).\tau^2)$. En remplaçant $\max(V_i,Z_{ij})$ par V_i+Z_{ij}, il minimise le coût total comme l'algorithme 3.4, mais sous la contrainte d'une taille de flotte limitée.

3.5.2 Croisements étudiés

Le croisement a pour but de découvrir de nouvelles solutions en combinant des chromosomes. Classiquement, les croisements sont conçus pour deux parents P1 et P2 et génèrent deux enfants C1 et C2. Le GA de base de Holland avec chromosomes binaires utilise un croisement très simple, qui tire au sort un point de coupure puis combine le début d'un parent (avant la coupure) avec la fin de l'autre (après la coupure).

La figure 3.5 illustre ce croisement (a) et un opérateur de mutation possible (b) sur des chromosomes binaires de 9 gènes (bits). L'application du croisement à des chromosomes codés comme des permutations (cas du CARP) cause des duplications (c). Le croisement X1 (d) évite les duplications : au lieu de copier la partie de P2 située après la coupure, il balaie entièrement P2 pour compléter C1 avec les éléments manquants.

<table>
<tr>
<td>point de coupure
↓
P1 : 0 1 1 0|1 1 0 0 1
P2 : 1 1 0 1|0 0 1 0 1

C1 : 0 1 1 0|0 0 1 0 1 (début P1+fin P2)
C2 : 1 1 0 1|1 1 0 0 1 (début P2+fin P1)

a) croisement pour chromosomes binaires</td>
<td>inversion
↓
Avant: 0 1 1 0 0 0 1 0 1
Après: 0 1 1 0 1 0 1 0 1

b) une mutation possible</td>
</tr>
<tr>
<td>P1 : 1 3 9 2|7 4 5 8 6
P2 : 8 3 4 7|6 1 9 5 2
C1 : 1 3 9 2|6 1 9 5 2 -> 1,9,2 dupliqués

c) duplications si on applique a) à
des chromosomes de permutation</td>
<td>P1 : 1 3 9 2|7 4 5 8 6
P2 : 8 3 4 7|6 1 9 5 2
C1 : 1 3 9 2 - - - - -
C2 : 1 3 9 2 8 4 7 6 5

d) croisement adapté X1</td>
</tr>
</table>

Figure 3.5 – *Opérations sur chromosomes binaires et de permutation.*

Nous avons testé sur le CARP trois croisements opérant sur des chromosomes de permutation et conçus à l'origine pour des problèmes de séquencement : X1 (le croisement à un point de coupure de la figure 3.5), LOX (*Linear Order Crossover*) et OX (*Order Crossover*). Ces croisements sont conçus pour générer des permutations valides (pas de gènes dupliqués) et pour préserver dans une certaine mesure l'ordre transmis par les parents.

LOX est destiné à des chromosomes *linéaires*, c'est-à-dire codant des objets qui ont clairement un début et une fin, comme un ordonnancement sur une machine. OX concerne des permutations *circulaires*, comme des tournées de voyageur de commerce. Intuitivement, le meilleur choix (confirmé dans les expérimentations) est OX parce qu'il n'y a aucune raison pour distinguer une « première » et une « dernière » tournée dans une solution du CARP.

Pour deux parents P1 et P2 de longueur τ, LOX et OX commencent par tirer au sort deux positions p et q avec $1 \leq p \leq q \leq \tau$. Pour construire l'enfant C1, la portion de P1 entre p et q inclus est copiée dans C1, aux mêmes positions. Dans LOX, P2 est ensuite balayé de 1 à τ et les éléments non déjà présents dans l'enfant remplissent de gauche à droite les positions libres de l'enfant. Dans OX, la seule différence est que le balayage de P2 et le rangement dans l'enfant s'effectuent de façon circulaire, en commençant à l'index $q + 1 \pmod{\tau}$. Pour tous ces croisements, C2 s'obtient de la même manière, en permutant les rôles de P1 et P2. Notons que X1 peut être vu comme un cas particulier de LOX avec $p = 1$.

Les algorithmes 3.9 et 3.10 détaillent la construction de C1 dans le cas du CARP. Rappelons que nos chromosomes contiennent les τ tâches du problème, mais que chaque tâche peut apparaître comme un des deux arcs qui la codent. Par rapport aux croisements LOX et OX standards, il faut donc vérifier, en copiant un arc u de P2, que ni u ni $inv(u)$ ne sont déjà dans C1. Nos versions réalisent les croisements en $O(m)$, grâce à un tableau de booléens *free* indiquant les arcs non encore copiés dans l'enfant. Ils évitent d'avoir en même temps $p = 1$ et $q = \tau$, pour engendrer des enfants différents des parents.

```
for u := 1 to m do free(u) := true endfor
draw p in [1,τ]
if p = 1 then draw q in [1,τ-1] else draw q in [p,τ] endif
for i := p to q do
   C1(i) := P1(i)
   free(P1(i)),free(inv(P1(i))) := false
endfor
j := 0
for i := 1 to τ with free(P2(i))=true do
   j := j + 1
   if j = p then j := q + 1 endif
   C1(j) := P2(i)
   free(P2(i)),free(inv(P2(i))) := false
endfor
```

Algorithme 3.9 – *Croisement LOX en O*(*m*) *pour le CARP*

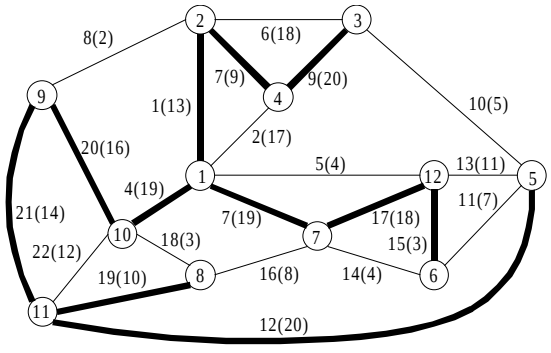
Dans le cas de chromosomes avec délimiteurs de tournées (3.5.1.1), les croisements ignorent les délimiteurs présents dans les parents et l'enfant C1 est immédiatement évalué avec la procédure d'évaluation séquentielle pour déterminer son fitness $F(C1)$ et insérer les délimiteurs. Pour des chromosomes sans délimiteurs (3.5.1.2), C1 est évalué par *Split*.

```
for u := 1 to m do free(u) := true endfor
draw p in [1,τ]
if p = 1 then draw q in [1,τ] else draw q in [p,τ] endif
for i := p to q do
   C1(i) := P1(i)
   free(P1(i)),free(inv(P1(i))) := false
endfor
i,j := q
for i := (i mod τ) + 1 to q with free(P2(i))=true do
   j := (j mod τ) +1
   C1(j) := P2(i)
   free(P2(i)),free(inv(P2(i))) := false
endfor
```

Algorithme 3.10 – *Croisement OX en O*(*m*) *pour le CARP*

La figure 3.6 décrit un exemple complet de croisement pour un réseau avec $n = 11$ nœuds, une capacité $Q = 5$ et 22 arêtes dont $\tau = 11$ tâches-arêtes (traits épais), ayant toutes des demandes égales à 1. Pour alléger, le réseau initial est dessiné au lieu du graphe interne orienté G qui a $m = 44$ arcs. Les valeurs sur chaque arête [i, j] sont l'index de l'arc (i,j) tel que $i < j$ (par exemple 7 pour l'arc (2,4)) et le coût de traversée. L'index de l'arc opposé (j,i) n'est pas mentionné mais vaut par convention 22 + u, par exemple 29 pour (4,2).

Deux exemples de parents P1 et P2 sont donnés, avec un enfant C1 généré par LOX. Les trois dernières lignes donnent les découpages en tournées et les coûts des solutions calculés par notre implémentation de *Split* (algorithme 3.4). Les graphes auxiliaires considérés par *Split* sont trop volumineux pour être dessinés ici. On peut remarquer que certaines arêtes sont traitées dans un sens différent par P1 et P2 : par exemple, l'arête [3,4] est traitée dans le sens (3,4) dans P2 (arc d'index 9) mais dans le sens (4,3) dans P1 (index 31).




```
Rang        :   1    2    3    4    5    6    7    8    9   10   11 = τ
Coupure     :                          de p=6     à q=8  (inclus)
                                          ↓         ↓
Parent P1   :  31   21   20   17   15 | 07   03   12 | 23   19   26
Parent P2   :  34   09   29   20   41   26   43   25   15   39   23
Enfant C1   :  09   20   41   26   43 | 07   03   12 | 15   39   23

Split sur P1: (31,  21,  20,  17,  15),(07),(03,  12,  23,  19,  26), F(P1) = 318
Split sur P2: (34,  09,  29),(20,  41,  26),(43,  25,  15,  39,  23), F(P2) = 324
Split sur C1: (09,  20,  41,  26),(43,  07),(03,  12,  15,  39,  23), F(C1) = 311
```

Figure 3.6 - *Exemple de parents avec un enfant obtenu par le croisement LOX.*

3.5.3 Mutations simples étudiées

En théorie, l'opérateur de mutation apporte aux algorithmes génétiques la propriété d'ergodicité de parcours d'espace. Cette propriété indique que l'algorithme sera susceptible d'atteindre toutes les solutions, sans pour autant les parcourir tous dans le processus d'exploration. Ainsi, en toute rigueur, l'algorithme génétique peut converger sans croisement, et certaines versions fonctionnent de cette manière (Fogel et *al.*, 1966). Les propriétés de convergence des algorithmes génétiques sont donc fortement dépendantes de cet opérateur.

En pratique, la mutation est un simple ingrédient de diversification, appliqué avec un faible taux p_m aux enfants pour empêcher une convergence prématurée du GA. Nous avons testé deux opérateurs nommés *Move* et *Swap*. Pour un enfant C, *Move* tire au sort deux positions p et q avec $1 \leq p \leq \tau$, $0 \leq q \leq \tau$, $p \neq q$ et $p \neq q$-1. La tâche u = C(p) est déplacée après celle de rang q, q = 0 signifiant une insertion en tête. Enfin, on inverse avec une probabilité de 0.5 le sens de traitement de u (remplacement par $inv(u)$). *Swap* tire au sort deux positions distinctes p et q entre 1 et τ inclus, permute les tâches situées à ces positions, et inverse chacune d'elles avec une probabilité de 0.5.

Les deux mutations peuvent être implémentées en $O(\tau)$. Dans le cas de chromosomes avec délimiteurs de tournées, *Move* et *Swap* ignorent les délimiteurs de C. L'enfant muté est aussitôt évalué avec la procédure d'évaluation séquentielle qui détermine son fitness F(C) et insère les délimiteurs. Dans le cas de chromosomes sans délimiteurs, C est évalué par *Split*.

3.5.4 Remplacement de la mutation par une recherche locale

Nous avons essayé de remplacer les opérateurs précédents par une procédure de recherche locale *LS* (*local search*) qui améliore les enfants avant de les injecter dans la population. En effet, les supporteurs du *GA* initial de Holland (qui ne prévoit pas cette technique) affirment que la recherche locale ne respecte pas l'esprit de la méthode : le croisement paraît réduit à une astuce nécessaire pour échapper aux minima locaux atteints par la recherche locale. On peut cependant interpréter cette phase d'amélioration individuelle comme un apprentissage ou une évolution culturelle (appelée en anglais *meme* par Dawkins, 1976) permettant à l'individu de mieux s'adapter à l'environnement.

En tout cas, il est clair aujourd'hui que les GA avec recherches locales sont bien meilleurs que les GA classiques qu'ils peuvent même surpasser les autres métaheuristiques comme le recuit simulé et les méthodes tabou. Par exemple, un GA de ce type proposé par Prins (2000) pour le problème d'ordonnancement open-shop surpasse l'algorithme de recuit simulé et les trois méthodes tabou publiées pour ce problème . De tels GA sont appelés *GA hybrides*, *recherches locales génétiques* ou *algorithmes « mémétiques »*, en anglais *hybrid GA* (*HGA*), *genetic local search* (*GLS*) ou *memetic algorithm* (*MA*).

Notre recherche locale *LS* opère sur des solutions du CARP, pas sur des chromosomes. Elle commence donc par convertir l'enfant *C* à améliorer en une solution *S*. Elle effectue ensuite des passes successives en $O(\tau^2)$ qui énumèrent toutes les paires de tâches (u,v). En fait, les voisinages suivants sont explorés (x est la tâche traitée après u et y celle après v) :

- N_1: inverser *u* dans sa tournée, c'est-à-dire remplacer *u* par *inv*(*u*) ;
- N_2: déplacer *u* après *v*, ou avant *v* si *v* est la première tâche de sa tournée ;
- N_3: déplacer 2 tâches adjacentes *u* et *x* après *v*, ou avant si *v* est 1ère dans sa tournée ;
- N_4: permuter *u* et *v* ;
- N_5: transformations 2-opt (voir figure 3.7).

Chaque mouvement sauf N_1 peut impliquer une ou deux tournées. De plus, dans N_2 à N_4, on teste les deux directions possibles quand on insère une arête à sa nouvelle position. Par exemple, l'implémentation de N_4 comprend en fait 4 cas de permutations : *u* et *v* peuvent être remplacés par *v* et *u*, *inv*(*v*) et *u*, *v* et *inv*(*u*) ou *inv*(*v*) et *inv*(*u*). Dans N_5, notons que certains mouvements nécessitent l'inversion d'une chaîne de tâches (*cf.* figure 3.7).

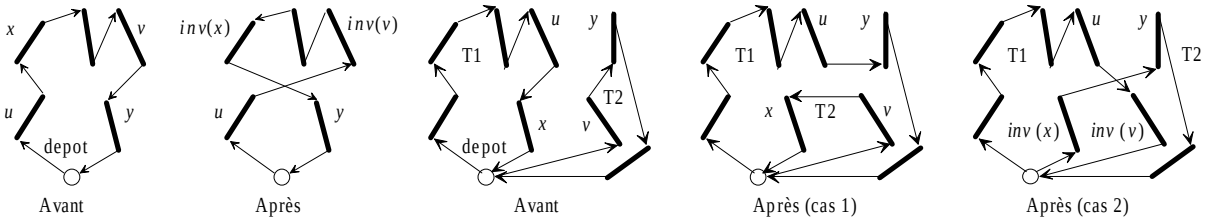


Figure 3.7 *– Mouvements 2-opt de N5 sur une tournée (à gauche) ou deux (à droite). Les tâches sont en traits épais, les traits fins désignent des plus courts chemins non détaillés.*

Chaque passe sur les voisinages stoppe à la première amélioration trouvée. La transformation correspondante est effectuée. Le processus est répété jusqu'à ce qu'on ne trouve plus d'amélioration. La solution obtenue est reconvertie en chromosome. Dans le cas de chromosomes sans délimiteurs, le chromosome résultant est systématiquement ré-évalué par *Split* car, pour la même séquence, *Split* trouve parfois un découpage de coût inférieur.

Notons que les déplacements de tâches peuvent vider complètement une tournée et économiser ainsi un véhicule. Cependant, une tournée peut se vider temporairement durant une passe, puis recevoir de nouvelles tâches lors des passes suivantes. C'est pourquoi de telles tournées vides sont détruites uniquement à la fin de la recherche locale.

3.5.5 Structure de la population et initialisation

Le choix de la population initiale d'individus conditionne fortement l'efficacité d'un GA. Si les positions des optima dans l'espace des solutions sont totalement inconnues, il est naturel de générer aléatoirement des individus bien dispersés. Bien entendu, les opérateurs de croisement et de mutation devront ensuite veiller à préserver cette diversité, afin de parcourir le plus largement possible l'espace des solutions.

Nous utilisons une population contenant un nombre fixe *nc* de chromosomes, rangés dans un tableau Π. Les chromosomes sont stockés avec leur évaluation et la population est maintenue triée par ordre croissant des coûts pour faciliter les opérations de sélection vues

plus loin. Pour chaque chromosome initial, on génère une permutation aléatoire des τ tâches et on tire à pile ou face l'arc qui va représenter chaque arête.

Le chromosome obtenu est aussitôt évalué par la procédure *ad hoc* (procédure séquentielle ou *Split*). De telles solutions aléatoires sont en moyenne très mauvaises, c'est pourquoi nous avons aussi testé des GA dans lesquels Π inclut trois bonnes solutions obtenues par les heuristiques décrites en 3.3 : (Path-Scanning), AM (Augment-Merge) et UH (heuristique d'Ulusoy).

Dans les algorithmes génétiques classiques, des solutions identiques (*clones*) peuvent envahir la population, causant ainsi une convergence prématurée du GA. Ce phénomène est encore pire dans notre cas à cause de la recherche locale LS, qui compresse rapidement les solutions dans un intervalle réduit de coûts. On peut l'éviter en interdisant les clones. La détection exacte des clones est possible mais alourdit les calculs (utilisation de techniques de *hashing*, *cf*. Cormen et *al*., 1990).

Nous nous contentons d'interdire des individus de coûts identiques. Pour cela, on génère au début un à un les individus Π_1, Π_2 … en s'assurant qu'ils ont bien des coûts différents. En cas de conflit, on tente un nombre maximum de fois (50 par exemple) de générer l'individu courant Π_k. En cas d'échec, la population est tronquée à k-1 individus, ce qui ne se produit que quand on se fixe un effectif *nc* trop élevé.

Ensuite, pendant les itérations du GA, on peut tester en $O(1)$ si un enfant est un clone avec un tableau de booléens *used* indicé par les valeurs des coûts : *used*(*cost*) = *true* ssi la population Π contient un individu de coût *cost*. Ce tableau doit être indicé entre 0 et *UB*, un majorant suffisant des valeurs des coûts possible. Un croisement est dit *improductif* si les enfants générés ne peuvent pas être ajoutés à la population à cause d'un coût dupliqué. Un tel croisement sera purement et simplement ignoré par le GA.

3.5.6 Sélection et remplacement

Tout GA sélectionne des parents et remplace des solutions existantes par les enfants obtenus. Ce processus tend à favoriser les meilleurs individus de la population et éliminer les mauvais. En général, il est stoppé après un nombre maximum de croisements productifs, après un nombre maximum de croisements sans amélioration de la meilleure solution Π_1 ou quand une borne inférieure *LB* du coût minimal est atteinte. Dans ce dernier cas, Π_1 est évidemment optimale. Nous avons testé les deux grands types de GA, qui agencent de manière différente les étapes de sélection et de remplacement.

3.5.6.1 Algorithme génétique incrémental

L'itération de base d'un algorithme génétique *incrémental* sélectionne deux parents, applique les opérateurs de croisement et de mutation, puis remplace un ou deux chromosomes de la population par les enfants obtenus. Nous avons testé deux techniques de sélection :

- la *méthode de Reeves* (1995). Rappelons que la Π est triée par coûts croissants. Le rang i du 1er parent est choisi avec une probabilité $2(nc-i+1) / (nc(nc+1))$. Ainsi, la probabilité de tirer un individu de coût médian est d'environ $1 / nc$, celle de tirer le meilleur Π_1 est

quasiment le double (2 / (*nc*+1)), tandis que la probabilité de tirer le pire Π(*nc*) est 2 / (*nc*(*nc*+1). Le rang du second parent est choisi avec une probabilité 1 / *nc*.

- la *méthode du tournoi binaire* : deux chromosomes sont sélectionnés aléatoirement. Le meilleur donne le premier parent. Le processus est répété pour définir le second parent.

Le croisement choisi au départ du GA (X1, LOX ou OX) est ensuite appliqué. On ne conserve qu'un des enfants, par tirage au sort. Cet enfant subit une mutation avec une probabilité fixée p_m. Il écrase ensuite un chromosome choisi uniformément sous la médiane, c'est-à-dire avec un rang *nc* / 2 à *nc*.

3.5.6.2 Algorithme génétique générationnel

Nous avons testé un schéma générationnel proposé par Hartmann en 1998. Chaque génération partitionne aléatoirement Π en paires de chromosomes. Chaque paire subit un croisement et les enfants obtenus sont ajoutés à la population, dont l'effectif passe ainsi à 2.*nc*. La population augmentée est finalement ramenée à sa taille normale en la triant par coût croissant et en conservant les *nc* meilleures solutions.

3.5.7 Redémarrages

Pour « décoincer » un GA qui n'améliore plus sa meilleure solution, nous avons testé une méthode de remplacement partiel (*restart*), proposée récemment dans un GA pour un problème de localisation de plates-formes pétrolières (Cheung et *al*., 2001). L'algorithme 3.11 décrit l'adaptation de cette technique à notre population avec des coûts distincts. Les données sont la population Π de *nc* chromosomes triés par ordre croissant des coûts et le nombre de chromosomes à remplacer *nrep*, par exemple *nc*/4.

```
done := 0 //compteur des solutions déjà remplacées
repeat
   générer au hasard une population Ω avec
   nrep coûts distincts non présents dans Π
   trier Ω par coûts croissants
   k := 0
   repeat
      k := k + 1
      if F(Ω(k)) < F(Π(nc)) then
         Π(nc) := Ω(k); done := done + 1; rétablir le tri de Π
      else
         croiser Ω(k) avec chaque individu de Π∪Ω
         C := meilleur enfant obtenu avec un coût non présent dans Π
         if F(C) < F(Π(nc)) then
            Π(nc) := C; done := done + 1; rétablir le tri de Π
         endif
      endif
   until (done = nrep) or (k = nrep)
until done = nrep
```

Algorithme 3.11 – *Procédure de remplacement partiel utilisée dans les restarts*

3.6 Sélection des composants et des paramètres

3.6.1 Implémentation et jeux de tests

Les GA ont été programmés en Delphi 5 (successeur 32 bits de Turbo Pascal) et exécutés sur un PC sous Windows 98 équipé d'un microprocesseur Pentium III cadencé à 1 GHz. L'évaluation numérique utilise trois ensembles de problèmes-tests pour le CARP, qui peuvent être téléchargés à l'adresse web http://www.uv.es/~belengue/carp.html.

Le premier ensemble (fichiers *gdb*) contient les 25 instances de la thèse de DeArmon (1981). Les instances 8 et 9 sont omises par tous les auteurs car elles contiennent des erreurs (graphes non connexes). Les 23 problèmes restants ont de 7 à 27 nœuds et 11 à 55 arêtes. Toutes les arêtes sont à traiter : il s'agit donc de postiers chinois avec contrainte de capacité (CCPP, voir chapitre 1). Le deuxième ensemble (fichiers *val*) comprend 34 CCPP plus durs construits par Belenguer et Benavent (2002), avec 24 à 50 nœuds et 34 à 97 arêtes.

Le troisième et dernier ensemble, dit « d'Eglese » (fichiers *egl*) contient 24 instances de grande taille conçues en fait par Belenguer et Benavent (2002), avec 77 à 140 nœuds et 98 à 190 arrêtes. Elles sont basées sur des portions du réseau routier rural du comté de Lancaster (Royaume-Uni), étudié par Li et Eglese (1996) pour la planification des opérations de sablage en cas de verglas. Belenguer et Benavent en ont tiré 24 fichiers en faisant varier la capacité des véhicules et le pourcentage d'arêtes à traiter. Certaines de ces instances sont de vrais CARP, dans lesquels certaines arêtes ne sont pas à traiter.

3.6.2 Choix des composants et des paramètres de l'algorithme

Nous avons testé plusieurs GA en commençant par une version simple comprenant : une population de 50 chromosomes aléatoires avec délimiteurs de tournées (sans clones et sans bonnes solutions heuristiques initiales), la méthode de sélection de Reeves, le croisement LOX, une mutation *Move* avec une probabilité de 0.02, le remplacement de deux chromosomes choisis sous la médiane par les deux enfants et 5000 itérations. Puis nous avons mené le plan d'expérience de la table 3.1 pour améliorer la combinaison de composants.

N°	*Tests*	*Impact sur le coût de Π(1)*	*Decision*
1	chromosomes sans délimiteurs	diminution	chromosome sans délimiteurs
2	pas de mutation	augmentation	maintenir la mutation
3	mutations *Swap* et *LS*	meilleurs résultats avec *LS*	appliquer la recherche locale *LS*
4	clones autorisés	augmentation	clones interdits
5	test de combinaisons (nc,p_m)	meilleur si nc = 30 et p_m = 0.1	utiliser nc = 30, p_m = 0.1
6	gestion générationnelle	petite augmentation	maintenir gestion incrémentale
7	sélection par tournoi binaire	petite diminution	appliquer tournoi binaire
8	croisement OX au lieu de LOX	petite diminution	utilisation de OX
9	garder un seul enfant	petite diminution	garder un enfant au lieu de deux
10	remplacer le pire individu	augmentation	remplacement sous la médiane
11	heuristiques PS, AM et UL dans la population initiale	petite diminution mais convergence accélérée	ajout de ces solutions à Π
12	appliquer *LS* à la population initiale	augmentation	pas de *LS* sur population initiale
13	procédure de remplacement partiel	diminution	*restarts* ajoutés

Table 3.1 - *Sélection des meilleurs composants.*

Les différents composants sont interdépendants. Par exemple nous avons choisi une population sans clones car elle assure une meilleure dispersion des solutions, ce qui favorise une bonne exploration de l'espace de recherche tout en évitant une convergence prématurée. Du coup, on peut se permettre d'appliquer comme mutation la recherche locale *LS* avec des taux p_m assez élevés (10 à 30 %), sans que la convergence du GA soit trop rapide. Cependant, la probabilité qu'un enfant amélioré par *LS* soit rejeté à cause d'un coût déjà présent dans Π augmente rapidement avec p_m et le GA perd trop de temps dans les itérations improductives. Ce constat nous a conduit à prendre des petites populations : ainsi, pour $nc \leq 40$, le taux moyen de rejet est raisonnable, de l'ordre de 5 à 10% selon les instances.

Concernant les *paramètres* comme le nombre maximum d'itérations, trop d'articles sur les métaheuristiques montrent des résultats avantageux, obtenus avec des réglages différents selon les instances. Le meilleur réglage pour chaque instance est en plus rarement mentionné, ce qui fait qu'un lecteur reprogrammant la méthode avec un réglage fixe a peu de chances de retrouver d'aussi bons résultats. Comme mentionné par Barr *et al*. (1995), il est nécessaire de donner des *résultats standard*, obtenus en appliquant à toutes les instances le même réglage.

Avoir des résultats standard est important pour toute comparaison fiable avec d'autres méthodes. Ces résultats donnent aussi une idée de la qualité des solutions qu'on obtiendrait dans des conditions réelles d'utilisation, quand on n'a pas le temps d'essayer divers réglages. La table 3.2 liste les valeurs standards de nos paramètres, déterminés après la phase préliminaire de sélection des composants. Il s'agit des valeurs donnant les meilleurs résultats moyens sur l'ensemble des instances. La taille surprenante du tableau *used*, introduit en 3.5.5 pour la détection des clones, correspond au coût maximal des solutions aléatoires présentes dans la population initiale, pour les plus grandes instances (*egl*).

Nom	*Rôle du paramètre*	*Valeur*
nc	taille de la population	30
mnt	nombre max. d'essais de génération aléatoire d'un individu dans la population initiale	50
UB	borne supérieure des coûts (dimension du vecteur *used*, *cf.* 3.5.5)	50000
p_m	taux de recherche locale dans la phase principale du MA	0.1
mnpi	nombre max. d'itérations productives dans la phase principale du MA (avant restarts)	20000
mnwi	nombre max. d'itérations productives sans amélioration de $\Pi(1)$ dans la phase principale du MA (avant restarts)	6000
mnrs	nombre max. de restarts	20
nrep	nombre de solutions remplacées dans chaque restart	8
rnpi	nombre max. de croisement productifs par restart	2000
rnwi	nombre max. d'itérations productives sans amélioration de la solution $\Pi(1)$, par restart	2000
p_r	taux de recherche locale dans les restarts	0.2

Table 3.2 – *Les paramètres standards du MA.*

En résumé, notre meilleur GA est incrémental. Il utilise une population sans clones de 30 chromosomes sans délimiteurs de tournées, évalués par *Split*. La population initiale inclut les solutions calculées par les heuristiques Path-Scanning, Augment-Merge et celle d'Ulusoy. Les parents choisis par tournoi binaire subissent un croisement OX. Un seul enfant est choisi à pile ou face puis subit une mutation par recherche locale avec un taux de 10% : le GA est donc en fait un algorithme mémétique (MA). L'enfant muté ou pas remplace un chromosome tiré au sort dans la seconde moitié de la population (coûts supérieurs au coût médian).

```
main program                                          //programme principal
  initialize (Π,nc,used,UB,mnt)                       //creation population initiale
  if F(Π(1)) > LB then begin                          //si LB non atteint
    search (Π,nc,used,LB,pm,mnpi,mnwi)                //phase d'exploration principale
    rs := 0                                           //initialise nombre de restarts
    while (rs < mnrs) and (F(Π(1)) > LB) do           //boucle sur les restarts
      rs := rs+1                                      //compte un restart
      partial_replacement (Π,nc,nrep)                 //remplacement partiel (3.5.7)
      search (Π,nc,used,LB,pr,rnpi,rnwi)              //exploration courte et intensive
    endwhile
  endif
endmain

procedure initialize (Π,nc,used,UB,mnt)               //procedure d'initialisation
  for k := 1 to UB do used(k) := false endfor         //initialise les coûts utilisés
  k := 0                                              //nombre de chromosomes générés
  exécuter les heuristiques PS, AM et UH              //calcule 3 bonnes solutions
  stocker solutions dans H(1),H(2),H(3)
  for i := 1 to 3 do                                  //ajouter si possible H(i) à Π
    convertir H(i) en un chromosome S; split(S)       //construction du chromosome S
    if not used(F(S)) then                            //si S n'est pas un clone
      k := k+1; Π(k) := S; used(F(S)) := true         //ajouter S à Π
    endif
  endfor
  repeat                                              //ajoute des solutions aléatoires
    try  := 0                                         //initialise le nombre d'essais
    repeat                                            //essais de génération
      try := try + 1                                  //incrémente le nombre d'essai
      générer S au hasard; split(S)                   //crée un chromosome aléatoire
    until (not used(F(S))) or (try = mnt)             //jusqu'à OK ou échec (3.5.5)
    if not used(F(S)) then                            //si S n'est pas un clone
      k := k+1; Π(k) := S; used(F(S)) := true         //ajoute S à Π
    endif
  until (k = nc) or (used(F(S))                       //Π est complète ou bien échec
  if used(F(S)) then nc := k endif                    //tronque Π en cas d'échec
  trier Π par coûts croissants                        //tri facilitant remplacements
endproc

procedure search (Π,nc,used,LB,pls,mpi,mwi)           //procedure d'exploration
  npi := 0                                            //croisements productifs
  nwi := 0                                            //idem, sans amélioration
  repeat                                              //boucle sur les croisements
    choisir parents P1,P2 par tournoi binaire         //selection (3.5.6.1)
    appliquer OX à P1,P2; tirer un fils C             //croisement (3.5.2)
    split(C)                                          //évaluation par Split (3.5.1)
    tirer k dans [⌊nc/2⌋,nc]                          //Π(k) à remplacer (3.5.6.1)
    if random < pls then                              //decision d'appliquer LS
      M := LS(C)                                      //appliquer LS (3.5.4)
      split(M)                                        //réévaluation (3.5.4)
      //si l'enfant muté M n'est pas un clone, il remplace C
      if (not used(F(M))) or (F(M) = F(Π(k))) then C := M endif
    endif
    if (not used(F(C))) or (F(C)=F(Π(k))) then        //teste si C n'est pas un clone
      npi := npi + 1                                  //compte un croisement productif
      if F(C) < F(Π(1)) then nwi := 0 else nwi := nwi+1 endif
      used(Π(k)) := false; used(C) := true            //mises à jour des coûts utilisés
      Π(k) := C                                       //effectue le remplacement
      décaler Π(k) pour re-trier Π                    //re-trier Π
    endif
  until (npi=mpi) or (nwi=mwi) or (F(Π(1))=LB)        //test d'arrêt
endproc
```

Algorithme 3.12 – *Structure générale du meilleur MA*

L'algorithme 3.12 résume la structure générale. La procédure *initialize* construit la population initiale. Le MA exécute ensuite une phase principale d'exploration (procédure *search*) dans laquelle la recherche locale est appliquée avec un taux p_m. A l'intérieur de *search*, le taux de recherche locale, le nombre maximum d'itérations et le nombre maximum d'itérations sans améliorations sont notés respectivement *pls*, *mpi* et *mwi*. La phase principale se termine après *mnpi* itérations (croisements), après *mnwi* itérations sans amélioration de la meilleure solution $\Pi(1)$ ou si une borne inférieure *LB* est atteinte ($F(\Pi(1)) = LB$).

Si, à la fin de la phase principale, $F(\Pi(1)) \neq LB$, l'algorithme exécute *mnrs* phases d'exploration plus courtes mais intensives, les *restarts*. Chaque restart renouvelle partiellement la population avec la procédure de l'algorithme 3.11 puis appelle la procédure *search* avec un taux de recherche locale augmenté p_r et des nombres réduits de croisements *rnpi* et *rnwi*.

3.7 Résultats des évaluations numériques

3.7.1 Format des tableaux de résultats

Le MA est comparé sur les instances de la littérature décrites en 3.6.1 par rapport au meilleur algorithme publié pour le CARP, l'excellente méthode tabou *Carpet* publiée par Hertz *et al*. dans *Operations Research* en 2000. Les tableaux 3.3 et 3.4 présentent des résultats pour la première version complète de l'algorithme mémétique que nous avons développée, celle qui utilise des chromosomes avec délimiteurs. Comparée à la version définitive de l'algorithme 3.12, la procédure *Split* est remplacée par la procédure d'évaluation séquentielle et il n'y a pas de restarts. Les tableaux suivants 3.5 à 3.7 donnent des résultats pour la version définitive : chromosomes sans délimiteurs, évaluation par *Split* et phase d'exploration principale suivie de restarts.

Les tableaux contiennent les colonnes suivantes. La colonne *file* donne le nom du problème. Celles libellées *n*, τ et *LB* indiquent respectivement le nombre de nœuds, le nombre de tâches et une borne inférieure. Il s'agit dans tous les cas de l'excellente borne de Belenguer et Benavent (2002), sauf pour l'instance *gdb*14 pour laquelle une borne meilleure a été trouvée par Amberg et Voβ (2002). Les colonnes *TS* et *MA* fournissent les résultats *standard* de *Carpet* et du MA, obtenus dans notre cas avec les paramètres de la table 3.2. La colonne *BKS* (*best-known solution*) indique la meilleure solution connue *avant* notre MA. Il s'agit en général d'une solution trouvée par *Carpet* avec divers réglages de paramètres.

Les coûts en gras sont des améliorations par rapport à *Carpet*. Les soulignés indiquent que le MA a amélioré la meilleure solution connue. Les astérisques signalent les optima prouvés, quand *LB* est atteinte. Les quatre dernières lignes indiquent respectivement pour chaque colonne l'écart moyen à *LB* en % (*Average*), le pire écart (*Worst*), le nombre d'optima prouvés par atteinte de *LB* (*Optima*) et le nombre de meilleures solutions détenues par chaque méthode, en tenant compte des éventuelles améliorations du MA (*New best*).

3.7.2 Résultats pour l'algorithme avec chromosomes sans délimiteurs

Sur les 23 instances *gdb* (table 3.3), le *MA* est tout à fait comparable à *Carpet*. Il améliore ce dernier une fois (en gras), fait moins bien deux fois, mais affiche le même écart moyen à la borne (0.48%). En fait, en utilisant plusieurs réglages, le MA retrouve toutes les meilleures solutions connues, mais n'en améliore aucune. Le temps de calcul moyen par instance est de trois minutes. Nous n'avons pas conservé les temps de calcul pour chaque fichier : cette version a été testée au début de la thèse, sur un PC qui a été depuis remplacé.

File	n	τ	LB	BKS	TS	MA
gdb1	12	22	316	316*	316*	316*
gdb2	12	26	339	339*	339*	339*
gdb3	12	22	275	275*	275*	275*
gdb4	11	19	287	287*	287*	287*
gdb5	13	26	377	377*	377*	377*
gdb6	12	22	298	298*	298*	298*
gdb7	12	22	325	325*	325*	325*
gdb10	27	46	344	348	352	356
gdb11	27	51	303	311	317	**311**
gdb12	12	25	275	275*	275*	275*
gdb13	22	45	395	395*	395*	395*
gdb14	13	23	450	458	458	458
gdb15	10	28	536	544	544	544
gdb16	7	21	100	100*	100*	100*
gdb17	7	21	58	58*	58*	58*
gdb18	8	28	127	127*	127*	127*
gdb19	8	28	91	91*	91*	91*
gdb20	9	36	164	164*	164*	164*
gdb21	11	11	55	55*	55*	55*
gdb22	11	22	121	121*	121*	123
gdb23	11	33	156	156*	156*	156*
gdb24	11	44	200	200*	200*	200*
gdb25	11	55	233	233*	235	235
Average				0.31%	0.48%	0.48%
Worst				2.64%	4.62%	3.49%
Optima				19	18	17
New best				23	20	20

Table 3.3 – *Résultats du MA avec chromosomes à délimiteurs sur les fichiers gdb*.

La table 3.4 concerne les 34 fichiers *val*. Le MA améliore *Carpet* 9 fois, fait aussi bien 11 fois et moins bien 14 fois, en particulier sur les 8 plus grosses instance avec 50 nœuds. Il trouve seulement 10 solutions optimales (contre 15 pour *Carpet*) mais semble un peu plus stable : son écart moyen à *LB* est 1.65% contre 1.90%. Le point fort du MA est ici l'amélioration de deux meilleures solutions connues : *val4d* et *val5a*. Les fichiers *val* étant plus gros que les *gdb*, le temps de calcul moyen passe à 10 minutes environ. C'est pourquoi nous avons renoncé à évaluer cette version du MA sur les fichiers *egl*, qui sont encore plus gros.

Bien qu'étant moins performant en moyenne que la version sans délimiteurs, l'algorithme mémétique avec délimiteurs de tournées supporte plus facilement des contraintes supplémentaires, qu'il suffit de tester au moment du découpage des enfants par la procédure d'évaluation séquentielle. Nous avons par exemple testé avec succès une contrainte d'autonomie des véhicules.

File	*n*	τ	LB	BKS	TS	MA
val1A	24	39	173	173*	173*	173*
val1B	24	39	173	173*	173*	179
val1C	24	39	235	245	245	245
val2A	24	34	227	227*	227*	227*
val2B	24	34	259	259*	260	**259***
val 2C	24	34	455	457	494	**457**
val 3A	24	35	81	81*	81*	81*
val 3B	24	35	87	87*	87*	87*
val 3C	24	35	137	138	138	138
val 4A	41	69	400	400*	400*	408
val 4B	41	69	412	412*	416	420
val 4C	41	69	428	430*	453	**434**
val 4D	41	69	520	546	556	**536**
val 5A	34	65	423	423*	423*	423*
val 5B	34	65	446	446*	448	**446***
val 5C	34	65	469	474	476	481
val 5D	34	65	571	593	607	**589**
val 6A	31	50	223	223*	223*	223*
val 6B	31	50	231	233	241	**233**
val 6C	31	50	311	317	329	**317**
val 7A	40	66	279	279*	279*	283
val 7B	40	66	283	283*	283*	283*
val 7C	40	66	333	334	343	**336**
val 8A	30	63	386	386*	386*	386*
val 8B	30	63	395	395*	401	401
val 8C	30	63	517	528	533	538
val 9A	50	92	323	323*	323*	327
val 9B	50	92	326	326*	329	336
val 9C	50	92	332	332*	332*	336
val 9D	50	92	382	399	409	410
val 10A	50	97	428	428*	428*	433
val 10B	50	97	436	436*	436*	446
val 10C	50	97	446	446*	451	456
val 10D	50	97	524	536	544	549
Average				0.55	1.90	1.65
Worst				4.26	8.57	7.33
Optima				22	15	10
New best				31	17	17

Table 3.4 *– Résultats du MA avec chromosomes à délimiteurs sur les fichiers val.*

3.7.3 Résultats pour l'algorithme avec chromosomes à délimiteurs

Intuitivement, cette version de l'algorithme mémétique est plus puissante que la précédente car la procédure d'évaluation *Split* découpe optimalement les enfants en solutions réalisables pour le CARP. Les tableaux de résultats présentent quelques différences de format : nous indiquons les temps de calcul de *Carpet* et du MA pour chaque instance (colonnes *Time*), les résultats des trois heuristiques classiques Path-Scanning (*PS*), Augment-Merge (*AM*) et celle d'Ulusoy (*UH*) et, enfin, les meilleurs résultats obtenus si nous utilisons plusieurs réglages des paramètres (colonne *BMA* pour *Best MA*).

Soulignons que nous sommes les premiers à évaluer de façon exhaustive l'heuristique d'Ulusoy, la référence originale (1985) ne donnant que quelques exemples, sans aucune comparaison avec d'autres algorithmes !

Pour une comparaison honnête des temps de calcul avec *Carpet*, rappelons que le MA est testé sur un Pentium III à 1 GHz. Les temps listés pour la méthode tabou ont été par contre mesurés sur une station de travail Silicon Graphics Indigo-2 à 195 MHz. Les benchmarks de l'organisme SPEC (2001) indiquent des indices de puissance de 8.88 pour la Silicon Graphics et de 48.2 pour notre PC. Donc, si *Carpet* était exécuté sur notre PC, ses temps de calcul seraient divisés par 48.2 / 8.88 = 5.43.

c) Résultats sur les fichiers gdb (table 3.5)

Concernant les heuristiques constructives, il faut souligner les bons résultats de la méthode d'Ulusoy par rapport à Path-Scanning et Augment-Merge, avec un écart moyen à *LB* de 6.4% au lieu de 10.4 et 8.4%. Quant au *MA*, il nous apporte une excellente surprise : il est au moins aussi bon que *Carpet* sur *toutes* les instances ! Même avec un seul jeu de paramètres, il trouve 21 solutions optimales, dont deux nouvelles (*gdb*11 et *gdb*15), et détient désormais 22 meilleures solutions. L'écart moyen et le pire écart par rapport aux bornes sont au moins divisées par deux. Le temps de calcul moyen est faible, autour de 5 secondes. Comme notre PC est environ 5 fois plus rapide que la Silicon Graphics, le MA est en moyenne 2 fois plus rapide que *Carpet*, à puissance de calcul égale.

En utilisant plusieurs combinaisons de paramètres (colonne *BMA*), nous avons pu améliorer le résultat standard du MA uniquement pour l'instance *gdb*10. Ainsi, le MA est devenu la seule métaheuristique capable de résoudre optimalement 21 instances *gdb* sur 23. Ces résultats montrent que ces jeux d'essai doivent désormais être considérés comme « faciles » et ne plus être utilisés dans des recherches futures sur le CARP.

File	n - τ	LB	BKS	TS	Time	PS	AM	UH	MA	Time	BMA
gdb1	12-22	316	316*	316*	17.1	350	349	330	316*	0.00	316*
gdb2	12-26	339	339*	339*	28.0	366	370	353	339*	0.44	339*
gdb3	12-22	275	275*	275*	0.4	293	319	297	275*	0.06	275*
gdb4	11-19	287	287*	287*	0.5	287*	302	320	287*	0.00	287*
gdb5	13-26	377	377*	377*	30.3	438	423	407	377*	0.11	377*
gdb6	12-22	298	298*	298*	4.6	324	340	318	298*	0.17	298*
gdb7	12-22	325	325*	325*	0.0	363	325*	330	325*	0.05	325*
gdb10	27-46	344	348	352	330.6	463	393	388	**350**	39.82	**348**
gdb11	27-51	303	311	317	292.2	354	352	358	**303***	7.09	**303***
gdb12	12-25	275	275*	275*	8.4	295	300	283	275*	0.06	275*
gdb13	22-45	395	395*	395*	12.4	447	449	413	395*	1.26	395*
gdb14	13-23	450	458	458	111.8	581	569	537	458	9.78	458
gdb15	10-28	536	544	544	13.1	563	560	552	**536***	7.42	**536***
gdb16	7-21	100	100*	100*	2.6	114	102	104	100*	0.05	100*
gdb17	7-21	58	58*	58*	0.0	60	60	58*	58*	0.00	58*
gdb18	8-28	127	127*	127*	9.2	135	129	132	127*	0.06	127*
gdb19	8-28	91	91*	91*	0.0	93	91*	93	91*	0.05	91*
gdb20	9-36	164	164*	164*	1.5	177	174	172	164*	0.11	164*
gdb21	8-11	55	55*	55*	1.1	57	63	63	55*	0.00	55*
gdb22	11-22	121	121*	121*	51.5	132	129	125	121*	0.33	121*
gdb23	11-33	156	156*	156*	6.1	176	163	162	156*	0.17	156*
gdb24	11-44	200	200*	200*	18.3	208	204	207	200*	3.35	200*
gdb25	11-55	233	233*	235	186.3	251	237	239	**233***	51.19	**233***
Average			0.33	0.50	48.9	10.4	8.4	6.4	0.17	5.29	0.08
Worst			2.64	4.62	330.6	34.6	27.0	19.9	2.23	51.19	2.23
Optima			19	18		1	2	1	21		21
Best			21	19		1	2	1	22		23

Table 3.5 – *Résultats du MA avec chromosomes sans délimiteurs sur les fichiers gdb.*

d) Résultats sur les fichiers *val* (table 3.6)

Les fichiers *val* semblent empiriquement plus durs que les *gdb* puisque l'écart moyen à *LB* augmente pour toutes les méthodes. L'algorithme d'Ulusoy reste cependant la meilleure heuristique constructive. Une fois encore, le MA avec réglage standard des paramètres est au moins aussi bon que *Carpet* et les écarts à la borne comme le temps de calcul sont divisés par deux en moyenne. Quatre meilleures solutions connues sont améliorées, incluant un nouvel optimum. En utilisant plusieurs réglages de paramètres, le MA améliore 6 des meilleures solutions connues avant lui et détient désormais toutes les meilleures solutions connues.

De plus, le temps de calcul reste raisonnable : 38 secondes en moyenne. La durée quasi-nulle pour *val*1*a* vient du fait que l'heuristique d'Ulusoy, utilisée pour la population initiale, trouve déjà une solution optimale : dans ce cas, le GA s'arrête à la fin de la génération de la population initiale et n'effectue aucun croisement.

File	n - τ	LB	BKS	TS	Time	PS	AM	UH	MA	Time	BMA
val1a	24-39	173	173*	173*	0.1	186	190	173*	173*	0.00	173*
val1b	24-39	173	173*	173*	50.2	209	196	197	173*	8.02	173*
val1c	24-39	235	245	245	506.1	331	294	280	245	28.67	245
val2a	24-34	227	227*	227*	0.9	259	238	255	227*	0.05	227*
val2b	24-34	259	259*	260	70.7	284	275	281	**259***	0.22	**259***
val2c	24-34	455	457	494	171.6	516	533	515	**457**	21.76	**457**
val3a	24-35	81	81*	81*	4.2	88	86	85	81*	0.05	81*
val3b	24-35	87	87*	87*	15.1	99	96	99	87*	0.00	87*
val3c	24-35	137	138	138	225.8	158	152	153	138	28.23	138
val4a	41-69	400	400*	400*	153.5	451	443	436	400*	0.72	400*
val4b	41-69	412	412*	416	410.1	487	487	468	**412***	1.21	**412***
val4c	41-69	428	430	453	379.7	539	483	486	**428***	19.11	**428***
val4d	41-69	520	546	556	1265.9	656	631	608	**541**	103.26	**530**
val5a	34-65	423	423*	423*	20.6	476	466	451	423*	1.86	423*
val5b	34-65	446	446*	448	224.4	508	487	486	**446***	1.04	**446***
val5c	34-65	469	474	476	288.7	544	509	504	**474**	101.01	**474**
val5d	34-65	571	593	607	1214.7	720	667	660	**581**	90.74	**581**
val6a	31-50	223	223*	223*	21.1	271	241	243	223*	0.17	223*
val6b	31-50	231	233	241	146.0	274	247	253	**233**	67.34	**233**
val6c	31-50	311	317	329	461.7	381	365	367	**317**	52.23	**317**
val7a	40-66	279	279*	279*	35.7	326	306	293	279*	1.97	279*
val7b	40-66	283	283*	283*	0.1	353	314	295	283*	0.44	283*
val7c	40-66	333	334	343	658.2	394	387	381	**334**	101.17	**334**
val8a	30-63	386	386*	386*	20.8	433	412	432	386*	0.66	386*
val8b	30-63	395	395*	401	441.5	455	426	439	**395***	9.95	**395***
val8c	30-63	517	528	533	798.9	596	604	603	**527**	71.46	**527**
val9a	50-92	323	323*	323*	154.5	358	348	345	323*	18.29	323*
val9b	50-92	326	326*	329	324.6	352	358	350	**326***	29.39	**326***
val9c	50-92	332	332*	332*	305.9	394	368	368	332*	71.19	332*
val9d	50-92	382	399	409	1914.8	492	436	462	**391**	211.13	**391**
val10a	50-97	428	428*	428*	29.9	453	453	452	428*	25.48	428*
val10b	50-97	436	436*	436*	99.9	474	460	457	436*	4.67	436*
val10c	50-97	446	446*	451	506.6	503	478	496	**446***	17.30	**446***
val10d	50-97	524	536	544	847.2	614	590	589	**530**	215.04	**528**
Average			0.82	1.90	346.2	16.8	11.4	10.9	0.61	38.35	0.54
Worst			5.00	8.57	1914.8	40.9	25.1	20.9	4.26	215.04	4.26
Optima			21	15		0	0	1	22		22
Best			28	17		0	0	1	32		34

Table 3.6 – *Résultats du MA avec chromosomes sans délimiteurs sur les fichiers val.*

e) Résultats sur les fichiers *egl* (table 3.7)

Pour certains fichiers, toutes les arêtes ne sont pas requises, contrairement aux problèmes précédents. Une colonne *m*/2 a donc été ajoutée au tableau pour indiquer le nombre d'arêtes. La colonne *BKS* a été supprimée car la seule méthode (à part le MA) testée sur ces instances est *Carpet*. Les résultats de *Carpet*, non publiés, ont été calculés par Mittaz à la demande de Belenguer et Benavent. Mittaz n'a pas mentionné les temps de calcul.

File	*n*	*m*/2	τ	LB	TS	PS	AM	UH	MA	Time	BMA
egl-e1-A	77	98	51	3515	3625	4115	4605	3952	**3548**	74.26	**3548**
egl-e1-B	77	98	51	4436	4532	5228	5494	5054	**4498**	69.48	**4498**
egl-e1-C	77	98	51	5453	5663	7240	6799	6166	**5595**	71.18	**5595**
egl-e2-A	77	98	72	4994	5233	6458	6253	5716	**5018**	152.58	**5018**
egl-e2-B	77	98	72	6249	6422	7964	7923	7080	**6340**	153.41	**6340**
egl-e2-C	77	98	72	8114	8603	10313	10453	9338	**8415**	129.63	**8395**
egl-e3-A	77	98	87	5869	5907	7454	7350	6723	**5898**	242.00	**5898**
egl-e3-B	77	98	87	7646	7921	9900	9244	8713	**7822**	255.35	**7816**
egl-e3-C	77	98	87	10019	10805	12672	12556	11641	**10433**	206.35	**10369**
egl-e4-A	77	98	98	6372	6489	7527	7798	7231	**6461**	291.87	**6461**
egl-e4-B	77	98	98	8809	9216	10946	10543	10223	**9021**	312.85	**9021**
egl-e4-C	77	98	98	11276	11824	13828	13623	13165	**11779**	252.38	**11779**
egl-s1-A	140	190	75	4992	5149	6382	6143	5636	**5018**	208.61	**5018**
egl-s1-B	140	190	75	6201	6641	8631	7992	7086	**6435**	208.77	**6435**
egl-s1-C	140	190	75	8310	8687	10259	10338	9572	**8518**	165.55	**8518**
egl-s2-A	140	190	147	9780	10373	12344	11672	11475	**9995**	874.36	**9995**
egl-s2-B	140	190	147	12886	13495	16386	15178	14845	**13174**	760.50	**13174**
egl-s2-C	140	190	147	16221	17121	20520	19673	19290	**16795**	746.93	**16715**
egl-s3-A	140	190	159	10025	10541	13041	11957	11956	**10296**	1070.50	**10296**
egl-s3-B	140	190	159	13554	14291	17377	15891	15663	**14053**	1064.01	**14028**
egl-s3-C	140	190	159	16969	17789	21071	19971	20064	**17297**	874.30	**17297**
egl-s4-A	140	190	190	12027	13036	15321	14741	13978	**12442**	1537.59	**12442**
egl-s4-B	140	190	190	15933	16924	19860	19172	18612	**16531**	1430.26	**16531**
egl-s4-C	140	190	190	20179	21486	25921	24175	23727	**20832**	1495.02	**20832**
Average					4.74	26.4	22.8	15.4	2.47	526.99	2.40
Worst					8.61	39.2	31.0	19.3	4.46	1495.02	4.46
Best					0	0	0	0	19		24

Table 3.7 – *Résultats du MA avec chromosomes sans délimiteurs sur les fichiers egl.*

Les fichiers *egl* sont encore plus durs que les *val* : la déviation moyenne à la borne *LB* (qui n'est jamais atteinte) augmente pour tous les algorithmes et devient franchement mauvaise pour l'heuristique Path-Scanning. L'algorithme d'Ulusoy se dégrade aussi mais reste néanmoins la meilleure heuristique constructive. Le MA avec paramètres standard surpasse *Carpet* dans tous les cas et divise encore une fois par deux l'écart moyen et le pire écart à la borne. L'utilisation de plusieurs réglages de paramètres permet d'améliorer encore plusieurs instances. Le prix à payer est un temps de calcul moyen atteignant 9 minutes.

La raison pour la non-atteinte de la borne peut être due à la mauvaise qualité des heuristiques ou de la borne. Selon Belenguer et Benavent, le graphe partiel des arêtes à servir est parfois non connexe et leur borne, qui n'exploite pas cette propriété, pourrait probablement être raffinée.

f) Comparaison globale des performances

La table 3.8 compare les résultats de notre meilleur algorithme mémétique avec les meilleures solutions publiées avant lui, si plusieurs réglages de paramètres peuvent être utilisés. La dernière ligne indique comment se répartissent les meilleures solutions connues, en tenant compte du MA. Les résultats sont éloquents : le MA est meilleur pour tous les critères du tableau . Rappelons aussi que, sur les 81 instances de la littérature que nous avons testées, le MA avec plusieurs réglages de paramètres n'est *jamais* battu par une autre méthode.

Critère	DeArmon 23 pbs		Benavent 34 pbs		Eglese 24 pbs	
	Autres	MA	Autres	MA	Autres	MA
Ecart moyen à LB %	0.33	0.08	0.82	0.54	4.74	2.40
Pire écart à LB %	2.64	2.23	5.00	4.26	8.61	4.46
Nombre d'optima prouvés	19	21	21	22	0	0
Meilleures solutions	21	23	28	34	0	24

Table 3.8 – *Comparaison globale des performances*.

3.7.4 Minimisation du makespan

Pour illustrer la flexibilité du MA, nous donnons dans la table 3.9 les résultats obtenus sur les fichiers *gdb* pour la minimisation de la durée de la plus longue tournée (*makespan*), sous la contrainte d'une flotte limitée de véhicules. Il s'agit de la première étude du CARP avec une fonction-objectif autre que le coût total. Les deux principaux changements dans l'algorithme génétique sont de remplacer l'algorithme 3.4 de *Split* par la version de l'algorithme 3.8 qui découpe optimalement un chromosome pour le nouvel objectif.

Une borne inférieure simple *LB*2 pour le makespan optimal peut être calculée comme suit. Soit r_{tot} la demande totale. Une borne inférieure K^* du nombre de véhicules est $\lceil r_{tot} / Q \rceil$. Cette borne est serrée pour les fichiers *gdb* puisqu'elle est toujours atteinte par le MA avec minimisation du coût total. D'autre part, la durée d'une tournée contenant seulement un arc requis u est $cost(u) = d(\sigma,u) + w(u) + d(u,\sigma)$. Donc, la durée minimale $\delta(u)$ d'une tournée réduite à une seule tâche-arête u est $\min(cost(u),cost(inv(u))$. Une première borne β du makespan est le maximum de ces coûts sur les tâches : $\beta = \max \{\delta(u) \mid u \in A, r(u) > 0\}$. Une deuxième borne est $\gamma = \lceil LB/K^* \rceil$, avec *LB* la borne inférieure du coût total figurant dans les tableaux précédents. Finalement, $LB2=max(\beta,\gamma)$.

Dans la table 3.9, les paramètres du MA sont ceux de la table 3.2, à l'exception du nombre de restarts *mnrs* qui est maintenant à 10. Le nombre de véhicules est fixé à K^*. La seule heuristique utilisée dans la population initiale est celle d'Ulusoy : *PS* et *AM* sont écartés parce qu'elles donnent souvent des solutions infaisables. Deux optimums sont prouvés et la déviation moyenne à *LB*2 est d'environ 6%. Cet écart est dû probablement à la mauvaise qualité de la borne. Un indice de cette mauvaise qualité est que la meilleure solution du MA est obtenue assez tôt (6 s en moyenne, colonne *Time**), comparé au temps de calcul total de 27.8 secondes (*Time*). Cela indique que d'autres solutions peuvent être aussi optimales.

File	n	τ	LB2	β	r_{tot}/W	γ	UH	MA	Time*	Time
gdb1	12	22	64	63	4.40	64	84	66	1.43	10.32
gdb2	12	26	59	59	5.20	57	81	60	3.13	17.03
gdb3	12	22	59	59	4.40	55	74	60	3.24	10.38
gdb4	11	19	72	64	3.80	72	98	74	0.11	6.04
gdb5	13	26	64	64	5.20	63	88	69	8.85	13.02
gdb6	12	22	64	64	4.40	60	75	68	0.22	10.44
gdb7	12	22	65	57	4.40	65	81	68	1.31	11.20
gdb8	27	46	38	38	9.22	35	54	40	11.43	33.12
gdb9	27	51	37	37	9.56	31	69	39	28.67	48.28
gdb10	12	25	69	39	3.70	69	86	73	0.22	10.43
gdb11	22	45	79	43	4.48	79	98	82	26.47	44.98
gdb12	13	23	93	93	6.06	64	124	96	0.11	6.97
gdb13	10	28	128	128	5.98	90	178	140	1.37	19.39
gdb14	7	21	20	15	4.24	20	27	21	0.16	24.99
gdb15	7	21	15	8	3.03	15	16	15*	4.83	4.83
gdb16	8	28	26	14	4.83	26	40	27	0.94	22.14
gdb17	8	28	19	9	4.10	19	22	19*	0.49	0.49
gdb18	9	36	33	19	4.14	33	40	34	3.90	46.90
gdb19	8	11	19	17	2.44	19	24	21	0.06	25.00
gdb20	11	22	31	20	3.96	31	45	32	1.15	16.75
gdb21	11	33	26	15	5.70	26	50	29	0.38	30.65
gdb22	11	44	25	12	7.59	25	45	28	17.13	107.71
gdb23	11	55	24	13	9.85	24	39	30	21.59	117.38
Average							39.0%	5.9%	5.96	27.76
Worst							92.3%	25.0%	28.67	117.38

Table 3.9 – *Minimisation du makespan sous contrainte d'une flotte limitée*.

3.8 Conclusion

Dans ce chapitre, nous avons développé les premiers algorithmes génétiques pour le CARP de base. Il s'agit *d'algorithmes mémétiques* (MA), c'est-à-dire qu'ils sont hybridés avec une recherche locale. En partant de versions simples, nous avons progressivement sélectionné des combinaisons de composants de base et des valeurs des paramètres conduisant à un MA redoutablement efficace.

Notre meilleur MA a été testé sur 81 instances de la littérature ayant jusqu'à 140 nœuds et 190 arêtes. Il est toujours au moins aussi bon que les meilleures métaheuristiques publiées. 32 meilleures solutions connues sont améliorées et 3 nouveaux optima ont été découverts. Les durées d'exécution sont raisonnables et en tout cas plus courtes que celles des méthodes tabou. Notre MA est désormais la meilleure méthode de résolution pour le CARP.

L'efficacité repose à la base sur un codage naturel des chromosomes sous forme de listes d'arcs reliés implicitement par des plus courts chemins et sur l'évaluation optimale par la procédure de découpage *Split*. En comparaison, *Carpet* utilise un codage beaucoup moins naturel qui

Ces principes de base sont renforcés par une synergie entre plusieurs composants qui sont individuellement assez simples mais qu'il a fallu concevoir, tester et assembler pour

trouver « la » bonne combinaison : une population de petite taille sans clones, l'ajout de bonnes solutions heuristiques dans la population initiale, l'utilisation de procédures d'amélioration locale comme opérateurs de mutation, des remplacements partiels suivis de courtes phases de redémarrage.

Le bon assemblage a été déterminé à la suite d'un plan d'expérience conduit sur des ensembles de problèmes-tests reconnus de la littérature. Il a été suivi d'une phase de réglage des paramètres. Enfin, l'implémentation a été particulièrement soignée en termes de complexité et de structures de données, pour diminuer les durées d'exécution.

A part ses excellents résultats, nos algorithmes mémétiques ont deux avantages spécifiques par rapport aux autres métaheuristiques. D'une part, la population finale fournit tout un panel de bonnes solutions et cette possibilité de choix est très appréciée des décideurs. D'autre part, les algorithmes sont très flexibles et peuvent gérer assez facilement des contraintes supplémentaires, qu'il « suffit » de tester au niveau de la procédure d'évaluation *Split*. Par exemple, pour gérer une limite de coût L sur les tournées (autonomie), on ignore dans le graphe auxiliaire de *Split* les arcs qui correspondent à des tournées de coût excédant L. Cette flexibilité est utilisée dans le chapitre 5 qui traite des extensions du CARP de base.

Au moment où nous écrivons ces lignes, nous constatons un intérêt croissant des chercheurs pour les problèmes de tournées sur arcs, notamment en ce qui concerne le développement de métaheuristiques puissantes. Nous tenons à citer en particulier deux travaux à paraître, réalisés par d'autres chercheurs pendant cette thèse: un algorithme hybride entre tabou et scatter search de Greistorfer (2003) et une méthode de recherche locale guidée (GLS) de Beullens et al. (2003). Le premier algorithme a été évalué sur les petites instances de Golden, Dearmon et Baker (gdb). Il surpasse Carpet mais n'atteint pas les performances de notre algorithme mémétique. Le second, plus prometteur, serait en moyenne aussi efficace que le MA sur les instances testées (gdb et val).

Nos travaux confirment que, contrairement à une opinion répandue, les algorithmes génétiques peuvent rivaliser avec des métaheuristiques établies depuis plus longtemps en optimisation combinatoire, comme le recuit simulé et la méthode tabou. Bien entendu, l'algorithme génétique de base n'est pas compétitif : il faut l'hybrider avec différentes techniques et lui incorporer des connaissances liées au problème, par exemple au travers des voisinages utilisés dans la recherche locale. Cette situation est similaire pour les méthodes tabou : les algorithmes d'origine ont été vite supplantés par des méthodes incluant des raffinements comme le *path relinking*.

Les recherches de ce chapitre ont fait l'objet de trois publications. La première version complète de l'algorithme mémétique (chromosomes avec délimiteurs de tournées, évaluation séquentielle, absence de restarts) a été présenté au congrès avec actes MOSIM'01. Elle était évaluée uniquement sur les fichiers *gdb* et *val*. L'algorithme a été ensuite accéléré et amélioré par l'utilisation de chromosomes sans délimiteurs, la procédure d'évaluation *Split*, des voisinages différents dans la recherche locale et des restarts. Ceci a permis de tester les grosses instances *egl*. Les résultats ont fait l'objet d'une communication au congrès Evostim, dont les actes ont été publiés chez Springer. Un article de revue , beaucoup plus détaillé, vient d'être accepté par *Annals of Operations Research*.

Références :

P. Lacomme, C. Prins, W. Ramdane-Chérif, Algorithmes génétiques pour le CARP, *MOSIM'01* (*3ème Conférence Francophone de Modélisation et Simulation*, Troyes, 25-27/04/01), A. Dolgui et F. Vernadat (éd.), pp. 117-124, SCS, 2001. ISBN 1-56555-212-1.

P. Lacomme, C. Prins, W. Ramdane-Chérif, A genetic algorithm for the capacitated arc routing problem and its extensions, dans *Applications of evolutionary computing*, E.J.W. Boers (éd.), pp. 473-483, Lecture Notes in Computer Science 2037, Springer, 2001.

P. Lacomme, C. Prins, W. Ramdane-Chérif, Competitive memetic algorithms for arc routing problems, à paraitre dans *Annals of Operations Research*.

CHAPITRE 4
CARP stochastique

4.1 Introduction

La modélisation des problèmes réels se fait souvent en supposant que les données nécessaires sont disponibles, parfaitement connues et plus généralement qu'il s'agit de données déterministes. Bien que les problèmes réels soient par nature stochastiques, on fait souvent une modélisation déterministe. Le problème d'optimisation consiste alors à trouver une solution exacte ou une bonne solution du modèle (déterministe) du problème réel (stochastique). La mise en œuvre de la solution trouvée par une telle approche met alors en évidence la présence de coûts supplémentaires qui n'ont pas été pris en compte lors de la modélisation et donc qui n'ont pas été minimisés.

L'idéal consiste à résoudre le problème stochastique, ce qui signifie qu'il faut proposer un modèle stochastique du problème ainsi que des méthodes de résolution adaptées. A l'exception de certains cas particuliers, cette démarche est difficile à mettre en œuvre car il s'agit de modéliser par des variables aléatoires le caractère stochastique du problème et, le plus souvent, de minimiser une fonction telle que l'espérance ou l'écart-type.

Une étape intermédiaire dans le cadre de l'optimisation combinatoire consiste à analyser les solutions trouvées à l'aide du modèle déterministe et de réaliser une étude de robustesse très utile d'un point de vue industriel mais aussi académique.

C'est dans cet esprit que nous voulons analyser les solutions du GA dans un contexte stochastique. Effectivement, dans le cas des applications réelles du CARP, la demande varie d'un jour à l'autre et on ne dispose que d'un volume moyen (ou d'un poids) pour chaque rue. Par exemple dans la collecte des déchets, les quantités à collecter sont imparfaitement connues, et on constatera lors de la mise en œuvre des solutions proposées une différence entre la demande utilisée dans le modèle déterministe (généralement moyenne) et la demande totale à traiter. Les véhicules sont alors susceptibles de faire plusieurs allers-retours supplémentaires au dépôt pour décharger, dès que leur capacité est atteinte.

Il paraît évident dans cette optique que, si les tournées calculées avec un modèle déterministe "chargent" au maximum les véhicules, le nombre d'allers-retours supplémentaires au dépôt risque d'être important. Intuitivement des tournées "moins chargées" devraient donc naturellement être plus robustes aux variations de la demande.

Ce constat nous a motivé pour proposer le *premier modèle stochastique en tournées sur arcs*, appelé *SCARP* (*Stochastic CARP*). Ce dernier permet de prendre en compte le cas où les quantités à collecter sont stochastiques et donc modélisées sous la forme de variables aléatoires. Notre objectif est d'étudier la robustesse des solutions calculées par une méthode déterministe.

Pour compléter les résultats théoriques du GA (chapitre 3), qui sont très satisfaisants, nous analysons dans ce chapitre sa robustesse face à des variations de la demande. Nous proposons de comparer le GA dans deux approches. La première est appelée *approche serrée* parce que la capacité des véhicules est maintenue à Q, ce qui signifie, compte tenu de la forte optimisation effectuée par le GA, que la quantité totale collectée dans chaque tournée est très proche ou égale à la capacité des véhicules. La seconde, appelée *approche relaxée*, consiste à limiter le taux de chargement des véhicules. Cela se fait facilement avec l'algorithme génétique en réduisant au début de la phase d'optimisation la valeur de Q.

Les deux approches commencent d'abord par résoudre le CARP et permettent d'obtenir une solution déterministe. La robustesse de la solution déterministe est ensuite évaluée en simulant les variations de la demande. Cette phase de simulation construit une suite de solutions perturbées indépendantes, permettant de collecter des informations statistiques sur la robustesse.

Le chapitre est organisé comme suit : nous commençons d'abord par définir le problème et les hypothèses de travail ainsi que notre modélisation des événements aléatoires pour ensuite donner un aperçu des travaux connus pour les tournées de véhicules stochastiques. Enfin nous présentons notre étude de la robustesse des solutions du CARP et les résultats numériques obtenus.

4.2 Définition du problème

Dans les applications réelles comme la collecte des déchets, les quantités sont imparfaitement connues. Pour une réponse à un appel d'offre d'une ville donnée, une société comme Onyx utilise des demandes moyennes, qui souvent estimées à partir des données connues d'une autre ville comparable. Donc, il est nécessaire de pouvoir estimer la variation du coût de la solution déterministe en fonction de la variation des quantités. Le SCARP considère un CARP de base où les demandes sont remplacées par des variables aléatoires.

Nous utilisons les notations du chapitre 2 mais adaptées si nécessaire au cas stochastique. On appelle *réplication* d'une solution déterministe S, une solution stochastique du SCARP obtenue en perturbant S par simulation de demandes aléatoires. Le problème peut mathématiquement être présenté sous la forme suivante :

S	solution déterministe du GA calculée avec des quantités moyennes
$cost(S)$	coût total de S
$nvu(S)$	nombre de tournées dans S
ne	nombre de réplications
ω_i (i :1,…,ne)	$i^{éme}$ réplication de S
$r(u,\omega_i)$	demande sur l'arc u dans la réplication ω_i
$cost(S,\omega_i)$	coût de S dans la réplication ω_i

$nvu(S,\omega_i)$	nombre de tournées de S dans la réplication ω_i
$\overline{cost(S,ne)}$	moyenne expérimentale calculée sur *ne* évaluations de $cost(S,\omega_i)$
$\overline{nvu(S,ne)}$	moyenne expérimentale calculée sur *ne* évaluations de $nvu(S,\omega_i)$
$\sigma(cost(S,ne))$	écart-type expérimental calculé sur *ne* évaluations de $cost(S,\omega_i)$
$\sigma(nvu(S,ne))$	écart-type expérimental calculé sur *ne* évaluations de $nvu(S,\omega_i)$
prob	une probabilité fixée à priori (par exemple 0.95 ou 0.99)

L'objectif du SCARP est alors de mesurer la robustesse des solutions vis-à-vis des variations des quantités à collecter. C'est-à-dire de minimiser la conséquence des événements aléatoires sur le coût et le nombre de tournées. Notre but n'est pas de résoudre le SCARP mais :

- d'étudier la sensibilité des solutions calculées par notre GA sur des demandes moyennes. Cela revient à évaluer $\overline{cost(S,ne)}$ et $\sigma(cost(S,ne))$ ainsi que $\overline{nvu(S,ne)}$ et $\sigma(nvu(S,ne))$ et de calculer une intervalle de confiance $[a, b]$, tel que $p\ (cost(S,\omega_i) \in [a,b]) \geq prob$.

- d'améliorer la robustesse des solutions par des modifications simples du GA déterministe, pour qu'elles soient moins sensibles à des variations des quantités.

4.3 Hypothèses de travail

Nous utilisons des hypothèses réalistes inspirées des règles de gestion connues dans les problèmes de collecte des déchets et qui peuvent se résumer aux trois points suivants :

- une solution prévisionnelle S est calculée à partir d'une estimation moyenne de la quantité de déchets à collecter sur chaque rue ;

- chaque véhicule exécute la tournée planifiée qui lui a été affectée et rencontre des demandes aléatoires. Il n'y a pas de collecte partielle d'une rue. Le véhicule doit donc rentrer au dépôt quand la prochaine rue a une demande qui dépasse la capacité résiduelle. Il revient ensuite au point d'interruption pour continuer la tournée prévue. Ces allers-retours au dépôt engendrent des tournées supplémentaires et donc augmentent le coût de la solution prévue.

- les coûts de traitement ne sont pas affectés par la variation des quantités. Cette hypothèse est respectée dans le cas où les coûts représentent des distances. Dans le cas où les coûts sont des temps, le nombre de containers dans une rue ne change pas mais c'est leurs charges qui varient. Comme les camions sont en général équipés de dispositifs de levage, on peut supposer que les temps de collecte sont peu affectés par les variations de quantité.

La figure 4.1 montre un exemple d'un CARP non orienté avec $Q = 4$, $\tau = 7$ tâches (les traits gras). Les demandes sont entre parenthèses. Les valeurs sur les traits fins représentent les plus courts chemins entre tâches. Les pointillés indiquent les plus court chemins entre les extrémités des tâches et le dépôt, ils valent tous 10. La gauche de la figure montre une solution déterministe calculée avec $r(u) = 1$ pour toutes les tâches. Cette solution coûte 137 et contient 2 tournées. Après perturbation des données, on obtient la solution de droite où chaque tournée prévue est découpée en deux. Le coût de la solution passe à 157 et le nombre de tournées devient 4.

Notons que les tournées supplémentaires dues aux allers-retours au dépôt non prévus sont inévitables dans des applications comme la collecte des déchets. Effectivement, il n'y a pas de système de communication entre les différents véhicules. Il s'agit du cas le plus fréquent dans les pays européens où les systèmes radio de communication entre les véhicules ne sont pas encore très répandus et sont encore à des stades expérimentaux. Dans ce cas, un véhicule doit nécessairement finir seul la tournée qui lui a été affectée. Même si ce système de communication existait, une collaboration ne serait possible qu'avec des véhicules du même secteur. Ce qui n'est pas évident parce que les véhicules exécutent leurs tournées en parallèle.

Quelques remarques sont utiles au sujet du coût total des tournées sur le terrain, comparé au coût de *S*. Par hypothèse, les coûts de traitement sont indépendants des quantités, il suffit par exemple d'imaginer qu'ils sont égaux aux coûts de traversée et que ces derniers sont des distances. Si la quantité totale rencontrée dans une tournée est plus faible que prévue, la tournée se termine plus tôt mais sa longueur ne change pas. Si la quantité totale dépasse *Q*, un aller-retour au dépôt est nécessaire. Ainsi, dans notre modèle, les variations de quantité augmentent le nombre de tournées et donc les coûts : $nvu(S,\omega_i) \geq nvu(S)$ et $cost(S,\omega_i) \geq cost(S)$.

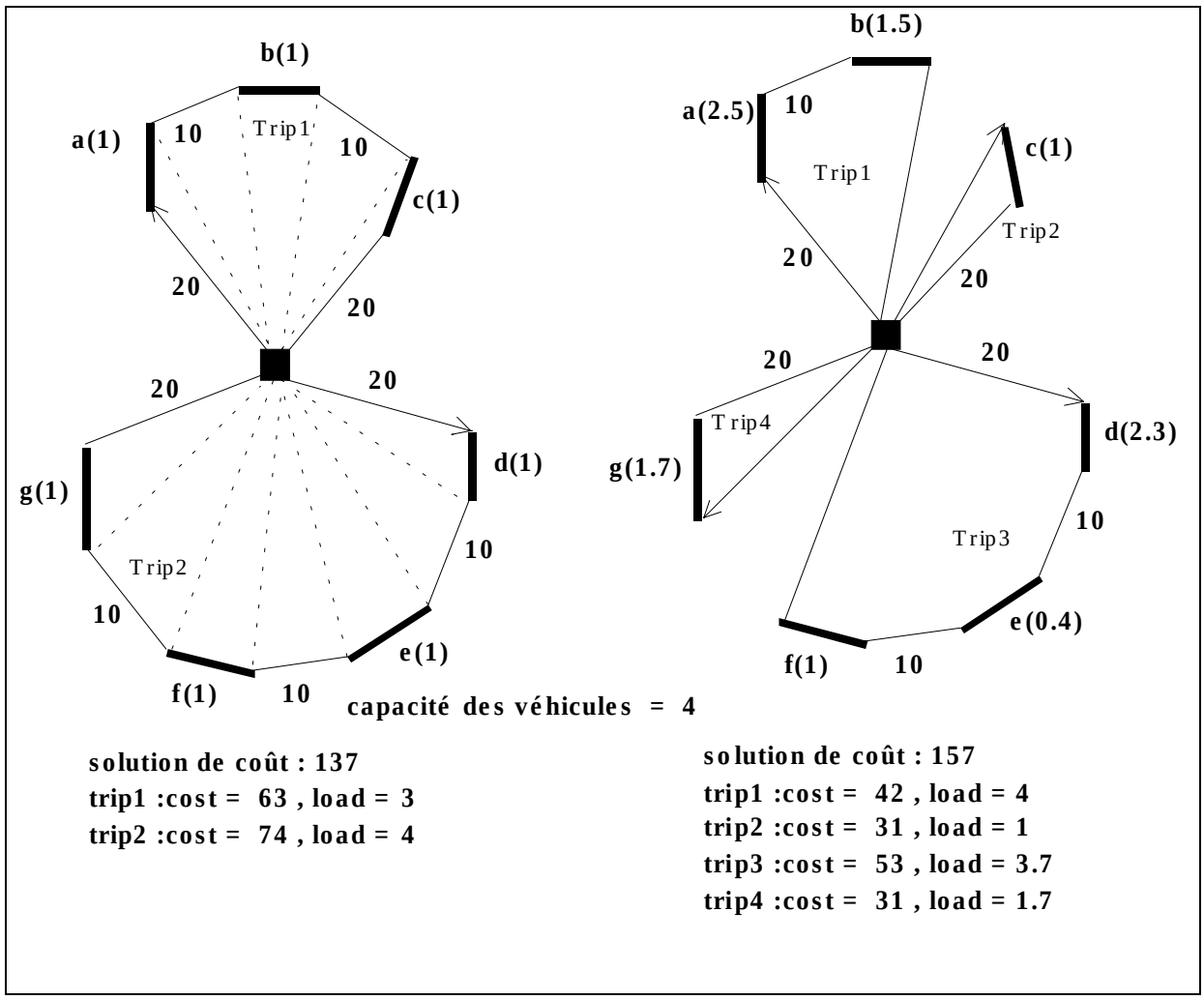


Figure 4.1 *– Comparaison entre la solution prévue avec des demandes déterministes (à gauche) et une solution perturbée par des demandes aléatoires (à droite).*

4.4 Calcul des réplications

Dans les problèmes de tournées sur arcs, les demandes sont en général une agrégation de demandes élémentaires. Dans la collecte des déchets, les quantités ramassées le long des arcs sont les sommes des contenus des poubelles de ces rues. Si on suppose que ces quantités sont des variables aléatoires indépendantes, d'après le *théorème central limite* la quantité à collecter le long de chaque arc suit une *loi normale*.

Nous utilisons une loi gaussienne $N(r_u,dev)$ pour modéliser des demandes aléatoires comme dans l'algorithme 4.1. Cette notation signifie que la loi a une moyenne égale à la demande déterministe r_u fournie dans les données et une variance *dev*. On pose $dev = r_u / 10$ et on élimine les cas où la quantité devient supérieure à Q ou négative (*gaussienne tronquée*). Dans ce deuxième cas, la demande est remplacée par une valeur minimale tolérée dans le réseau durant les réplications, appelée ε. Rappelons que cette façon de modéliser les demandes suppose que les variations des quantités sont indépendantes. Ceci est généralement vrai sauf dans quelques cas comme les lendemains de fêtes où les quantités augmentent simultanément sur l'ensemble des rues.

```
for all u ∈ R do
   r(u,ωi) := N(ru,dev)
   if r(u,ωi) > Q then r(u,ωi) := Q endif
   if r(u,ωi) < 0 then r(u,ωi) := ε endif
endfor
```

Algorithme 4.1 – *Génération des demandes aléatoires pour la réplication* ω_i.

Le but des réplications est d'évaluer une solution déterministe S (calculée avec des demandes moyennes). Pour simuler le comportement sur le terrain décrit dans les hypothèses 4.3, l'idée est de conserver toutes les tournées de S mais de les fragmenter si nécessaire dans les réplications. L'algorithme 4.2 calcule une solution stochastique obtenue à la réplication ω_i. Il parcours toutes les tournées de S et découpe séquentiellement chaque tournée S_k avec $\sum_{u \in S_k} r(u,\omega_i) > Q$.

```
Simuler des demandes aléatoires avec l'algorithme 4.1
nvu(S,ωi) := 0
cost(S,ωi):= 0
for k :=1 to |S| do
   //évaluation stochastique de la tournée Sk
   i,j := 1
   repeat
      load := S(k,i)
      while (j ≤ |Sk|) and (load + r(S(k,j+1),ωi) ≤ Q) do
         //la tâche n°j de Sk peut être traitée
         load := load + r(S(k,j+1),ωi)
         j := j + 1
      endwhile
      incrémenter nvu(S,ωi)
      cost(S,ωi) := cost(S,ωi) + cost(copy(S,i,j-1))
      i := j
   until j > |Sk|
endfor
```

Algorithme 4.2 – *Calcul de la solution stochastique* (*réplication* ω_i).

4.5 Littérature sur les problèmes de tournées stochastiques

A notre connaissance, aucun article n'a été publié sur les problèmes stochastiques en tournées sur arcs. Quelques travaux ont été réalisés sur les problèmes sur nœuds, notamment sur le VRP stochastique ou *SVRP*. Ils concernent surtout les cas où les clients et/ou les demandes sont aléatoires (Gendrau et *al.,* 1995 ; Laporte et Louveau, 1999 ; Verweij et *al.*, 2001).

Deux grandes approches de résolution directe sont utilisées pour résoudre ce type de problème : la première appelée *Chance Constrained Programming* (*CCP*), où les contraintes sont satisfaites avec une certaine probabilité et la seconde appelée *Stochastic Programming with Recourse* (*SPR*). Cette dernière consiste à calculer une solution préliminaire sans connaître la réalisation des variables aléatoires, pour ensuite appliquer des actions correctives dès qu'une contrainte du problème est violée.

Théoriquement, on peut tenter de transformer le SCARP en SVRP et donc utiliser la méthode SPR. Cette dernière a permis la résolution de SVRP allant jusqu'à 70 nœuds. Mais la transformation d'un CARP en VRP augmente considérablement la taille du problème.

En ce qui concerne l'optimisation stochastique en général, très peu de métaheuristiques ont été développées. Généralement, des hypothèses très restrictives portant sur le type de loi sont nécessaires pour appliquer ces méthodes. Il existe quelques exceptions comme l'algorithme *ASOVA* (*Algorithme Stochastique d'Optimisation d'une Variable Aléatoire*) proposé par Fleury en 1993, qui permet d'optimiser une variable aléatoire quelque soit sa loi de probabilité. La convergence en probabilité de cet algorithme est par ailleurs démontrée.

D'autres auteurs ont adapté les métaheuristiques classiques. Des algorithmes génétiques robustes ont été conçus pour l'optimisation des fonctions à plusieurs variables. Par exemple, Tsutsui et Ghosh (1997) ajoutent des perturbations à la fonction objectif tandis que Branke (1998) réalise une simulation pour évaluer la robustesse de chaque chromosome généré après croisement.

Notons enfin que la littérature sur ce sujet est nettement moins riche que la littérature concernant les problèmes d'optimisation déterministe.

4.6 Etude de la robustesse

Notre objectif n'est pas de résoudre directement le SCARP mais plutôt d'évaluer et d'analyser la robustesse des solutions calculées dans le cas déterministe. Nous souhaitons ensuite améliorer la robustesse des solutions du GA sans modifier la structure de l'algorithme, ni le schéma d'optimisation.

Nous proposons d'évaluer le GA dans deux approches : l'une appelée *serrée* et l'autre *relaxée*. Les deux utilisent une phase déterministe suivie d'une phase stochastique. La première phase consiste à calculer une solution déterministe *S* avec le GA du chapitre 3, en jouant éventuellement avec les paramètres. Dans la deuxième phase, une série de *ne* réplications ω_i de *S* est exécutée. Chaque ω_i calcule une solution perturbée à partir de *S* avec

l'algorithme 4.2. Au cours des *ne* réplications, des statistiques sur le coût et le nombre de tournées des solutions trouvées comme l'écart-type et la moyenne sont calculées. Une autre statistique très importante est le pourcentage des réplications générant des solutions avec des tournées supplémentaires. Ce dernier paramètre présente la probabilité qu'une solution initiale comporte au moins une tournée supplémentaire. Cette information présente un intérêt pratique évident car elle permet de connaître sur une année par exemple le nombre de jours où des situations coûteuses (tournées supplémentaires) vont se produire et perturber fortement la planification calculée.

Nous avons proposé une procédure *Perturbation*(*S*,*ne*) qui réalise *ne* réplications indépendantes successives à partir de *S*. Chaque réplication est calculée en appliquant l'algorithme 4.2 à *S*. La procédure calcule ensuite la moyenne et l'écart-type sur le coût et le nombre de tournées, le nombre de réplications *extratrip* ayant $nvu(S,\omega_i) > nvu(S)$, ainsi que les paramètres suivants :

$\Delta cost(S,ne) = \overline{cost(S,ne)} - cost(S)$
augmentation absolue du coût de la solution stochastique (après *ne* réplications) par rapport à la solution déterministe.

$\nabla cost(S,ne) = \Delta cost(S,ne)/cost(S) \times 100$
augmentation relative en % du coût de la solution stochastique par rapport à la solution déterministe.

$\Delta nvu(S,ne) = \overline{nvu(S,ne)} - nvu(S)$
augmentation absolue du nombre de tournées de la solution stochastique par rapport à la solution déterministe.

$\nabla nvu(S,ne) = \Delta nvu(S,ne)/nvu(S) \times 100$
augmentation relative en % du nombre de tournées de la solution stochastique par rapport à la solution déterministe.

$extratrip(S,ne) = extratrip/ne$
pourcentage des solutions stochastiques avec un nombre supplémentaire de tournées par rapport à la solution déterministe.

4.6.1 Approche *serrée*

Cette approche commence par une phase déterministe en utilisant le GA pour calculer une solution optimale *S* sur des demandes déterministes. L'idée est d'appliquer ensuite la procédure *Perturbation*(*S*,*ne*). Le but est d'analyser la sensibilité de la solution déterministe en la comparant avec la solution stochastique. Puisque le GA a tendance à construire des solutions avec des véhicules bien remplis, on peut supposer qu'elles sont sensibles à des variations de la demande. En effet, il n'y a pas assez de capacité résiduelle dans les véhicules et une augmentation des quantités engendre des allers-retours supplémentaires, et donc un coût plus élevé. C'est cette remarque qui nous a amenée à proposer l'approche suivante, dite *relaxée*.

4.6.2 Approche *relaxée*

A cause de la sensibilité de la solution calculée par l'approche précédente, nous proposons une approche dans laquelle la capacité des véhicules est réduite de *taux* pourcents dans la phase déterministe. De cette façon, une marge pour une charge supplémentaire est prévue dans chaque véhicule. Le GA est utilisé pour calculer une solution sur des demandes moyennes qui, comme dans l'approche précédente, sera suivie par une phase de réplications. Durant cette dernière, la capacité des véhicules est bien sûr restaurée à sa valeur initiale *Q*.

4.7 Evaluations numériques

Cette section présente les tests qui ont été effectués afin d'analyser en détail la robustesse des solutions calculées par le GA dans l'approche *serrée* et l'approche *relaxée*. Les exécutions utilisent un PC sous Windows 98 équipé d'un processeur Pentium III cadencé à 1 GHz. La programmation est en Delphi5. L'évaluation numérique utilise trois ensembles de problèmes-tests pour le CARP : les problèmes *gdb* de DeArmon (1981), les problèmes *val* de Belenguer et Benavent (2002) et enfin les problèmes *egl* d'Eglese et Li (1994).

Les statistiques calculées par la procédure *Perturbation* (*S*, *ne*) (4.6) permettent d'évaluer la solution stochastique par rapport à la solution déterministe pour chaque approche. Pour évaluer une solution S_1 du GA dans l'approche *serrée* et une solution S_2 du GA dans l'approche *relaxée*, nous calculons d'autres statistiques :

$\nabla cost(S_1,S_2) = (cost(S_2) - cost(S_1) / cost(S_1))\times 100$
augmentation relative en % du coût de la solution déterministe dans l'approche *relaxée* par rapport à la solution déterministe de l'approche *serrée*.

$\nabla cost(S_1,S_2,ne) = (\overline{cost(S_2,ne)} - \overline{cost(S_1,ne)})/\overline{cost(S_1,ne)}) \times 100$
augmentation relative en % du coût de la solution stochastique dans l'approche *relaxée* par rapport à la solution stochastique de l'approche *serrée*, après *ne* réplications.

$\nabla nvu(S_1,S_2) = (nvu(S_2) - nvu(S_1)/nvu(S_1))\times 100$
augmentation relative en % du nombre de tournées de la solution déterministe dans l'approche *relaxée* par rapport à celle de l'approche *serrée*.

$\nabla nvu(S_1,S_2,ne) = (\overline{nvu(S_2,ne)} - \overline{nvu(S_1,ne)})/\overline{nvu(S_1,ne)}) \times 100$
augmentation relative en % du coût de la solution stochastique dans l'approche *relaxée* par rapport à celle de l'approche *serrée*, après *ne* réplications.

$extranvu(S_1,S_2) = nvu(S_2) - nvu(S_1)$
nombre de tournées supplémentaires dans S_2 par rapport à S_1, à la fin de la phase déterministe.

$improv(S_1,S_2,ne)=(\nabla cost(S_1,ne) - \nabla cost(S_2,ne) / \nabla cost(S_1,ne)) \times 100$
pourcentage d'amélioration de la robustesse dans l'approche *relaxée* par rapport à l'approche *serrée*.

Nous posons ne = 1000, ε = 1 et *taux* = 10 %. Les tables 4.1 à 4.3 et 4.7 à 4.13 donnent les résultats du GA dans les deux approches sur les fichiers gdb, val et egl. Les tables 4.1 et 4.2 détaillent respectivement l'approche *serrée* et l'approche *relaxée* sur les fichiers gdb. De même 4.7 et 4.8 pour les fichiers val ainsi que 4.10 et 4.11 pour les données d'Eglese. Chaque table donne les résultats des réplications : $cost(S)$, $nvu(S)$, $\overline{cost(S,ne)}$, $\overline{nvu(S,ne)}$, $extratrip(S,ne)$, $\sigma(cost(S,ne))$, $\sigma(nvu(S,ne))$.

Les tables 4.3, 4.9 et 4.12 donnent la performance relative de l'approche *relaxée* par rapport à l'approche *serrée* respectivement sur les fichiers gdb, val et egl. Elles reprennent respectivement les paramètres suivants : $\nabla cost(S_1,S_2)$, $extranvu(S_1,S_2)$, $\nabla nvu(S_1,S_2,ne)$, $\nabla cost(S_1,S_2,ne)$, $\nabla nvu(S_1,S_2,ne)$, $extratrip(S_1,ne)$, $extratrip(S_2,ne)$, $improv(S_1,S_2,ne)$.

La table 4.13 présente une synthèse des déviations moyennes, maximales et minimales de la solution stochastique par rapport à la solution déterministe des deux approches.

4.7.1 Comparaison de la robustesse des deux approches sur les fichiers gdb

4.7.1.1 Approche *serrée*

La table 4.1 résume l'analyse de sensibilité du GA sur les 23 fichiers gdb pour l'approche *serrée*. Nous pouvons remarquer qu'il y a un grand écart entre la solution stochastique et la solution déterministe. Le coût de la solution et le nombre de tournées augmentent sur toutes les instances.

La solution stochastique augmente le coût de 8,05 % en moyenne. Cette augmentation est au minimum de 0.10 % et au maximum de 19,79 %. Pour le nombre de tournées, l'augmentation moyenne est de 22.68 % (0.75 au minimum et 46.83 au maximum). En moyenne, 70 % des réplications font état de tournées supplémentaires. Ce paramètre atteint 99.30 % pour gdb11 et gdb25.

Ces résultats montrent que les solutions du GA, même si elles sont optimales 21 fois sur 23 pour le CARP, sont très sensibles aux variations de la demande. Si on prend l'exemple du gdb1, le coût de la solution trouvée à la fin de la phase déterministe a un coût optimal de 316 (table 4.1). Dans la phase stochastique, le coût moyen après 1000 réplications atteint 351.23 donnant ainsi une déviation moyenne de 11,48 % par rapport à l'optimum. Le pourcentage des réplications générant des tournées supplémentaires est de 66,20 %. Nous pouvons conclure que les variations de la demande ont un impact très important sur les solutions du GA calculées avec l'approche *serrée*. L'écart-type est aussi important, il atteint 39.93 pour le coût et 1.50 pour le nombre de tournées.

file	cost(S_1)	nvu(S_1)	$\overline{\text{cost}(S_1, ne)}$	$\overline{\text{nvu}(S_1, ne)}$	extratrip(S_1, *ne*)	σ(cost(S_1,ne))	σ(nvu(S_1,ne))
gdb1	316	5	351.23	5.82	66.20	31.17	0.68
gdb2	339	6	389.50	8.08	94.20	28.06	1.10
gdb3	275	5	292.82	5.81	64.40	15.34	0.70
gdb4	287	4	320.93	5.24	79.70	23.58	0.86
gdb5	377	6	443.17	8.09	92.70	37.67	1.12
gdb6	298	5	335.00	6.25	79.40	27.98	0.86
gdb7	325	5	354.67	5.82	65.00	25.27	0.70
gdb10	350	10	385.10	11.99	92.30	21.19	1.09
gdb11	303	10	362.98	13.58	99.30	24.58	1.40
gdb12	275	4	286.29	4.43	43.00	13.12	0.50
gdb13	395	5	412.21	5.67	58.90	15.63	0.62
gdb14	458	7	511.76	8.18	78.20	39.93	0.85
gdb15	536	6	584.81	8.81	97.60	24.71	1.21
gdb16	100	5	102.45	5.47	40.30	3.74	0.63
gdb17	58	4	58.06	4.03	2.90	0.34	0.17
gdb18	127	5	134.46	6.63	88.50	5.23	0.94
gdb19	91	6	91.22	6.11	11.10	0.63	0.31
gdb20	164	5	167.02	5.30	28.70	4.91	0.49
gdb21	55	3	60.50	3.69	59.20	5.78	0.64
gdb22	121	4	129.40	5.74	91.00	5.39	0.95
gdb23	156	6	164.59	7.69	90.20	5.37	0.95
gdb24	200	8	205.82	9.64	87.90	3.63	1.02
gdb25	233	10	248.73	13.79	99.30	6.69	1.50

Table 4.1 – *Résultat de l'approche serrée sur les fichiers gdb.*

4.7.1.2 Approche *relaxée*

L'étude de la sensibilité du GA dans l'approche *relaxée* se fait de la même façon que dans l'approche précédente. Nous remarquons que cette fois les solutions restent stables.

Remarquons que 10 instances sur 23 affichent le même coût et le même nombre de tournées dans la solution stochastique et dans la solution déterministe. Pour les 13 instances restantes, la variation est faible : le coût et le nombre de tournées augmentent en moyenne respectivement seulement de 0.06 % (au maximum 0.46 %) et 0.11 % (au maximum 0.63 %). Le pourcentage moyen des réplications générant des tournées supplémentaires est d'environ 1.05 % (au maximum 6.80 %). L'écart-type atteint 12.13 pour le coût et 0.25 pour le nombre de tournées.

file	cost(S_2)	nvu(S_2)	$\overline{\text{cost}(S_2,ne)}$	$\overline{\text{nvu}(S_2,ne)}$	extratrip(S_2,*ne*)	σ(cost(S_2,ne))	σ(nvu(S_2, ne))
gdb1	337	6	337.00	6.00	0.00	0.00	0.00
gdb2	359	7	359.00	7.00	0.00	0.00	0.00
gdb3	296	6	296.00	6.00	0.00	0.00	0.00
gdb4	313	5	313.00	5.00	0.00	0.00	0.00
gdb5	409	7	409.00	7.00	0.00	0.00	0.00
gdb6	324	6	324.00	6.00	0.00	0.00	0.00
gdb7	351	6	351.00	6.00	0.00	0.00	0.00
gdb10	370	11	371.65	11.07	6.80	6.34	0.25
gdb11	331	12	331.93	12.05	4.70	4.58	0.22
gdb12	283	5	283.11	5.01	0.70	1.32	0.08
gdb13	403	6	403.00	6.00	0.00	0.00	0.00
gdb14	478	8	480.21	8.04	4.00	12.13	0.20
gdb15	544	7	544.23	7.01	1.00	2.60	0.10
gdb16	100	5	100.05	5.00	0.50	0.71	0.07
gdb17	58	4	58.00	4.00	0.10	0.06	0.03
gdb18	129	6	129.02	6.01	0.70	0.30	0.08
gdb19	91	6	91.00	6.00	0.00	0.00	0.00
gdb20	164	5	164.03	5.00	0.40	0.50	0.06
gdb21	63	4	63.00	4.00	0.00	0.00	0.00
gdb22	123	5	123.05	5.00	0.50	0.71	0.07
gdb23	158	7	158.05	7.01	1.20	0.44	0.11
gdb24	202	9	202.10	9.03	2.70	0.61	0.17
gdb25	237	12	237.03	12.01	1.00	0.27	0.10

Table 4.2 *– Résultat de l'approche relaxée sur les fichiers gdb.*

4.7.1.3 Comparaison de l'approche *serrée* et l'approche *relaxée* sur les fichiers gdb

La table 4.3 présente une comparaison des performances des deux approches. Les colonnes 2, 3 et 4 concernent la phase déterministe et les colonnes 5 et 6 la phase stochastique. Les colonnes 7 et 8 indiquent le pourcentage des réplications générant des tournées supplémentaires respectivement dans l'approche *serrée* et dans l'approche *relaxée*. Les valeurs de ces deux dernières colonnes sont déjà données dans les tables 4.1 et 4.2. Nous les reprenons dans ce tableau par souci de clarté. La dernière colonne mesure le pourcentage d'amélioration des résultats de l'approche *relaxée* par rapport à l'approche *serrée*.

En ce qui concerne la phase déterministe, la solution de l'approche *relaxée* est aussi bonne que la solution de l'approche *serrée* seulement dans 4 cas. Par contre dans toutes les autres instances, la solution est moins bonne. La déviation moyenne en terme de coût est de 4.43 % et 16.16 % en terme de nombre de véhicules. Nous concluons que l'approche *relaxée* donne une solution déterministe moins intéressante que l'approche *serrée*. Ceci était prévisible puisque le pourcentage de remplissage des véhicules a été volontairement limité dans cette dernière approche à 90% de leur capacité nominale.

Pour la phase stochastique, les solutions obtenues par l'approche *relaxée* sont moins bonnes que l'approche *serrée* dans seulement 2 cas. Dans toutes les autres instances, elles affichent une amélioration considérable. La déviation moyenne de l'approche *relaxée* par rapport à l'approche *serrée* est de -3,20 % pour le coût et -4,65 % pour le nombre de tournées.

La solution déterministe calculée par l'approche *relaxée* affiche un coût et un nombre de tournées très élevés comparé à l'approche *serrée*. Par contre cette dernière approche donne des solutions de meilleure qualité dans la phase stochastique. Le pourcentage d'amélioration de la robustesse par cette approche est de 98,58 %.

file	∇cost (S_1,S_2)	extranvu (S_1,S_2)	∇nvu (S_1,S_2)	∇cost (S_1,S_2,ne)	∇nvu (S_1,S_2,ne)	extratrip (S_1,ne)	extratrip (S_2,ne)	improv (S_1,S_2,ne)
gdb1	6.65	1	20.00	**-4.05**	3.09	66.20	**0.00**	**100.00**
gdb2	5.90	1	16.67	**-7.83**	**-13.37**	94.20	**0.00**	**100.00**
gdb3	7.64	1	20.00	1.09	3.27	64.40	**0.00**	**100.00**
gdb4	9.06	1	25.00	**-2.47**	**-4.58**	79.70	**0.00**	**100.00**
gdb5	8.49	1	16.67	**-7.71**	**-13.47**	92.70	**0.00**	**100.00**
gdb6	8.72	1	20.00	**-3.28**	**-4.00**	79.40	**0.00**	**100.00**
gdb7	8.00	1	20.00	**-1.03**	3.09	65.00	**0.00**	**100.00**
gdb10	5.71	1	10.00	**-3.49**	**-7.67**	92.30	6.80	92.63
gdb11	9.24	2	20.00	**-8.55**	**-11.27**	99.30	4.70	95.27
gdb12	2.91	1	25.00	**-1.11**	13.09	43.00	0.70	98.37
gdb13	2.03	1	20.00	**-2.23**	5.82	58.90	**0.00**	**100.00**
gdb14	4.37	1	14.29	**-6.16**	**-1.71**	78.20	4.00	94.88
gdb15	1.49	1	16.67	**-6.94**	**-20.43**	97.60	1.00	98.98
gdb16	0.00	0	0.00	**-2.34**	**-8.59**	40.30	0.50	98.76
gdb17	0.00	0	0.00	**-0.10**	**-0.74**	2.90	0.10	96.55
gdb18	1.57	1	20.00	**-4.05**	**-9.35**	88.50	0.70	99.21
gdb19	0.00	0	0.00	**-0.24**	**-1.80**	11.10	**0.00**	**100.00**
gdb20	0.00	0	0.00	**-1.79**	**-5.66**	28.70	0.40	98.61
gdb21	14.55	1	33.33	**4.13**	8.40	59.20	**0.00**	**100.00**
gdb22	1.65	1	25.00	**-4.91**	**-12.89**	91.00	0.50	99.45
gdb23	1.28	1	16.66	**-3.97**	**-8.84**	90.20	1.20	98.67
gdb24	1.00	1	12.50	**-1.81**	**-6.33**	87.90	2.70	96.93
gdb25	1.72	2	20.00	**-4.70**	**-12.91**	99.30	1.00	98.99
Average	**4.43 %**		**16.16 %**	**-3.20 %**	**-4.65 %**	**70.00 %**	**1.06 %**	**98.58 %**

Table 4.3 – *Comparaison des performances relatives des deux approches sur les fichiers gdb.*

La figure 4.2 présente deux courbes : une pour l'augmentation du coût moyen et l'autre pour l'augmentation du nombre de tournées. Remarquons que ces deux courbes prennent globalement des valeurs négatives.

La figure 4.3 donne le pourcentage des réplications générant un nombre supplémentaire de tournées dans les deux approches. On peut voir clairement que dans l'approche *relaxée* la courbe est quasiment sur l'axe des abscisses, tandis que dans l'approche *serrée* le pourcentage de solutions nécessitant au moins un aller-retour au dépôt atteint 100 %.

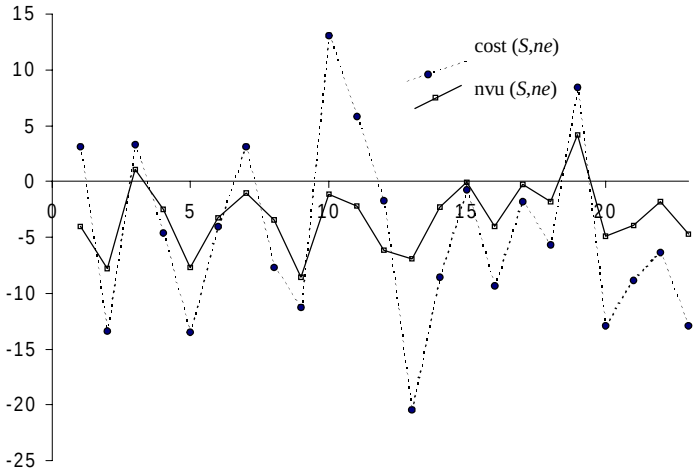


Figure 4.2 – *Résultats des paramètres* $\Delta cost(S_2, ne)$, $\Delta nvu(S_2, ne)$ *sur les fichiers gdb.* (1000 *réplications*).

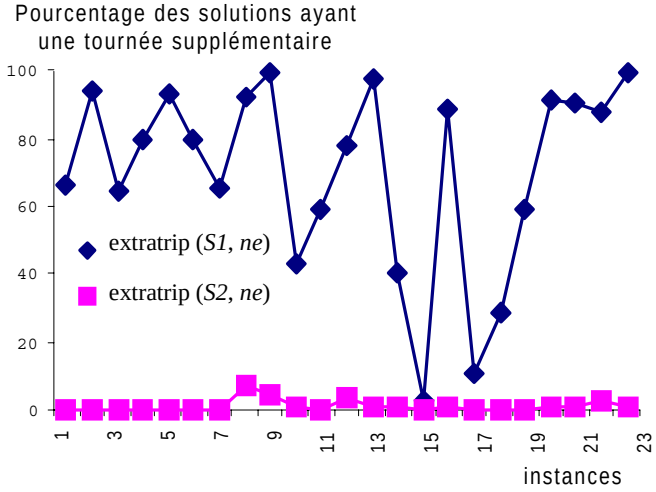


Figure 4.3 – *Comparaison de extratrip*(S_1, *ne*) *et extratrip*(S_2, *ne*) *sur les fichiers gdb.*

4.7.1.4 Exemple sur gdb19

Pour mettre en évidence l'impact des variations de la demande sur la solution, nous détaillons dans ce paragraphe l'instance gdb19 pour laquelle $Q = 27$. Supposons que toutes les arêtes du graphe présentées dans la table 4.4 soient à traiter.

u	c(u)=w(u)	r(u)	u	c(u)=w(u)	r(u)
[1,2]	4	8	[2,1]	4	8
[1,4]	3	3	[4,1]	3	3
[1,5]	1	5	[5,1]	1	5
[1,6]	2	8	[6,1]	2	8
[2,3]	1	4	[3,2]	1	4
[2,4]	9	6	[4,2]	9	6
[2,5]	5	1	[5,2]	5	1
[2,7]	2	9	[7,2]	2	9
[5,7]	7	9	[7,5]	7	9
[6,8]	5	5	[8,6]	5	5
[7,3]	6	8	[3,7]	6	8

Table 4.4 – *Description de l'instance gdb*19.

Avec l'approche *serrée* et la première phase déterministe, le GA calcule une solution de coût 55, composée de 3 tournées (figure 4.4 et table 4.6). Remarquons que deux véhicules sur trois sont pleins.

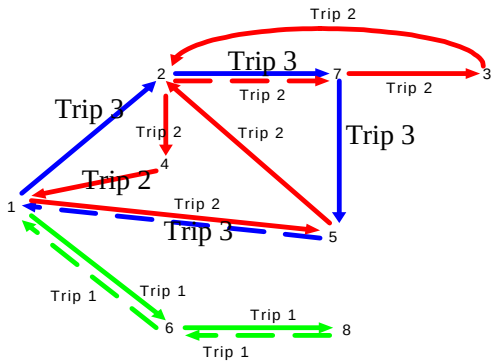


Figure 4.4 – *Représentation graphique de la solution déterministe de l'approche serrée du gdb*19.

Dans la phase stochastique (pour l'approche *serrée*), le coût de la solution augmente à 67. Les quantités ont été perturbées comme le montre la table 4.5. En effet, la dernière tournée de la solution déterministe a été découpée en deux (table 4.6). La solution est alors composée de 4 tournées. Nous donnons ci-dessous la liste des tâches par tournée :

Tournée 1 : ((1,6), (6,8))
Tournée 2 : ((1,5), (5,2), (7,3), (3,2), (2,4), (4,1))
Tournée 3 : ((1,2), (2,7))
Tournée 4 : (7,5)

Par contre l'approche *relaxée* donne une solution déterministe équivalente à la solution stochastique et qui coûte 63.

u	r(u)	r(u,ω_1)	u	r(u)	r(u,ω_1)
[1,2]	8	8.7	[2,1]	8	8.7
[1,4]	3	3.0	[4,1]	3	3.0
[1,5]	5	5.3	[5,1]	5	5.3
[1,6]	8	8.4	[6,1]	8	8.4
[2,3]	4	4.0	[3,2]	4	4.0
[2,4]	6	5.3	[4,2]	6	5.3
[2,5]	1	1.1	[5,2]	1	1.1
[2,7]	9	9.6	[7,2]	9	9.6
[5,7]	9	9.3	[7,5]	9	9.3
[6,8]	5	5.8	[8,6]	5	5.8
[7,3]	8	7.7	[3,7]	8	7.7

Table 4.5– *Génération aléatoire des données pour la réplication* ω_1 *sur gdb*19.

trip	solution de l'approche serrée avec demandes déterministes		solution de l'approche serré avec demandes aléatoires	
	load	cost	load	cost
Tournée1	14	14	14.20	14
Tournée2	27	27	26.40	27
Tournée3	26	14	18.30	12
			9.30	14

Table 4.6 – *Comparaison des solutions de l'approche serrée du gdb19 dans le cas déterministe et dans le cas stochastique*.

4.7.2 Comparaison de la robustesse des deux approches sur les fichiers val

La table 4.7 résume l'analyse de sensibilité du GA sur les 34 fichiers val pour l'approche *serrée*. Nous pouvons remarquer comme pour les gdb, qu'il y a un grand écart entre la solution stochastique et la solution déterministe. Le coût de la solution et le nombre de tournées augmentent sur toutes les instances sauf pour val1a. L'augmentation du coût pour la solution stochastique peut atteindre 22.59 % et le nombre de tournées peut atteindre 43 %. En moyenne, cette augmentation est de 6.54 % et 17.31 % respectivement pour le coût et le nombre de tournées.

La table 4.8 montre que les solutions restent stables dans l'approche *relaxée*. 20 instances sur 34 affichent le même coût et le même nombre de tournées pendant les réplications. Pour les instances restantes, la variation est faible : le coût et le nombre de tournées augmentent en moyenne respectivement de 0.07 % (au maximum 0.66 %) et 0.09 % (au maximum 1 %). Le pourcentage des *réplications* générant des tournées supplémentaires est au maximum égal à 8.80 %.

La comparaison des deux approches est présentée dans la table 4.9. Elle permet de conclure que :

- l'approche *relaxée* diminue le coût en moyenne de 1.99 % ;
- l'approche *relaxée* augmente le nombre de tournées en moyenne de 2.64 % ;
- le pourcentage des réplications générant un nombre de tournées supplémentaire est réduit de 53.05 % à 0.65 % (voir figure 4.5) ;
- l'approche *relaxée* améliore presque dans 100 % des cas l'approche *serrée*, le pourcentage moyen d'amélioration de la robustesse est de 96.33 %.

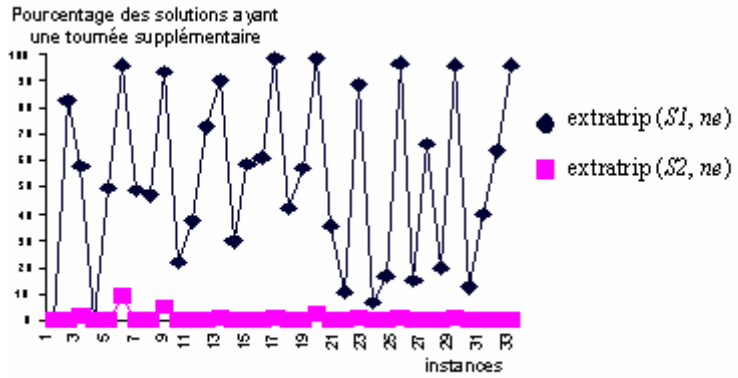


Figure 4.5 – *Comparaison de extratrip*(S_1, *ne*) *et extratrip*(S_2, *ne*) *sur les fichiers val.*

file	cost(S_1)	nvu(S_1)	$\overline{\text{cost}(S_1, ne)}$	$\overline{\text{nvu}(S_1, ne)}$	extratrip(S_1, *ne*)	σ(cost(S_1, ne))	σ(nvu(S_1, ne))
val1a	173	2	173.00	2.00	0.00	0.00	0.00
val1b	173	3	190.60	4.29	82.70	14.07	0.84
val1c	245	9	261.43	9.77	58.00	17.32	0.77
val2a	227	2	227.01	2.00	0.10	0.25	0.03
val2b	259	3	283.70	3.49	49.40	25.00	0.50
val2c	457	8	560.24	10.50	95.80	50.74	1.22
val3a	81	2	82.41	2.49	48.60	2.29	0.50
val3b	87	3	89.80	3.47	47.20	3.40	0.50
val3c	138	7	161.93	9.25	93.10	15.88	1.19
val4a	400	3	402.11	3.21	21.10	4.10	0.42
val4b	412	4	417.16	4.42	37.50	7.49	0.57
val4c	428	5	447.82	5.99	73.00	15.78	0.76
val4d	541	9	594.96	10.73	90.20	32.99	1.00
val5a	423	3	428.85	3.30	29.40	9.11	0.46
val5b	446	4	456.92	4.67	58.70	10.98	0.63
val5c	474	5	490.39	5.75	61.50	14.95	0.69
val5d	581	9	675.11	12.02	98.20	40.88	1.26
val6a	223	3	229.74	3.42	42.10	7.90	0.49
val6b	233	4	244.79	4.60	56.70	11.09	0.56
val6c	317	10	376.41	13.11	98.60	28.24	1.27
val7a	279	3	283.95	3.36	35.50	6.74	0.48
val7b	283	5	284.10	5.11	11.00	3.13	0.31
val7c	334	9	368.59	10.85	88.30	22.28	1.11
val8a	386	3	387.17	3.06	6.50	4.42	0.25
val8b	395	4	396.37	4.17	16.60	3.50	0.40
val8c	527	9	612.38	11.69	96.80	40.55	1.23
val9a	323	3	324.39	3.15	14.70	3.39	0.35
val9b	326	4	335.55	4.87	65.80	8.20	0.76
val9c	332	5	333.87	5.21	19.80	4.03	0.43
val9d	391	10	428.37	12.54	96.10	19.95	1.24
val10a	428	3	428.84	3.12	12.20	2.37	0.34
val10b	436	4	441.61	4.40	39.60	6.96	0.49
val10c	446	5	455.86	5.79	63.60	8.51	0.70
val10d	530	10	573.62	12.51	95.60	22.56	1.29

Table 4.7 – *Résultat de l'approche serrée sur les fichiers val.*

file	cost(S_2)	nvu(S_2)	$\overline{\text{cost}(S_2, ne)}$	$\overline{\text{nvu}(S_2, ne)}$	extratrip(S_2, ne)	σ(cost(S_2,ne))	σ(cost(S_2,ne))
val1a	173	2	173.00	2.00	0.00	0.00	0.00
val1b	179	4	179.03	4.00	0.20	0.63	0.04
val1c	261	10	261.22	10.01	1.40	1.97	0.12
val2a	227	2	227.00	2.00	0.00	0.00	0.00
val2b	262	3	262.00	3.00	0.00	0.00	0.00
val2c	499	9	501.82	9.09	8.80	10.49	0.30
val3a	81	2	81.00	2.00	0.00	0.00	0.00
val3b	89	4	89.01	4.00	0.20	0.23	0.04
val3c	146	8	146.97	8.05	5.10	4.47	0.23
val4a	406	4	406.00	4.00	0.00	0.00	0.00
val4b	418	5	418.00	5.00	0.00	0.00	0.00
val4c	448	6	448.00	6.00	0.00	0.00	0.00
val4d	565	10	565.47	10.01	1.00	4.90	0.10
val5a	441	4	441.00	4.00	0.00	0.00	0.00
val5b	466	5	466.00	5.00	0.00	0.00	0.00
val5c	497	6	497.00	6.00	0.00	0.00	0.00
val5d	617	10	617.23	10.01	0.80	2.73	0.09
val6a	223	3	223.00	3.00	0.00	0.00	0.00
val6b	245	5	245.01	5.00	0.10	0.25	0.03
val6c	337	11	337.38	11.02	2.40	2.81	0.15
val7a	279	4	279.00	4.00	0.00	0.00	0.00
val7b	283	5	283.00	5.00	0.00	0.00	0.00
val7c	352	10	352.11	10.01	0.70	1.27	0.08
val8a	395	4	395.00	4.00	0.00	0.00	0.00
val8b	409	5	409.00	5.00	0.00	0.00	0.00
val8c	565	10	565.19	10.01	0.70	2.39	0.08
val9a	326	4	326.00	4.00	0.00	0.00	0.00
val9b	332	5	332.00	5.00	0.00	0.00	0.00
val9c	340	6	340.00	6.00	0.00	0.00	0.00
val9d	409	11	409.11	11.01	0.70	1.40	0.08
val10a	436	4	436.00	4.00	0.00	0.00	0.00
val10b	446	5	446.00	5.00	0.00	0.00	0.00
val10c	459	6	459.00	6.00	0.00	0.00	0.00
val10d	555	11	555.03	11.00	0.10	0.95	0.03

Table 4.8 – *Résultat de l'approche relaxée sur les fichiers val.*

file	∇cost (S_1,S_2)	extranvu (S_1,S_2)	∇nvu (S_1,S_2)	∇cost (S_1,S_2,ne)	∇nvu (S_1,S_2,ne)	extratrip (S_1,ne)	extratrip (S_2,ne)	improv (S_1,S_2,ne)
val1a	0.00	0	0.00	0.00	0.00	**0.00**	**0.00**	0.00
val1b	3.47	1	33.33	**-6.07**	**-6.76**	82.70	0.20	99.76
val1c	6.53	1	11.11	**-0.08**	2.46	58.00	1.40	97.59
val2a	0.00	0	0.00	0.00	0.00	0.10	**0.00**	**100.00**
val2b	1.16	0	0.00	**-7.65**	**-14.04**	49.40	**0.00**	**100.00**
val2c	9.19	1	12.50	**-10.43**	**-13.43**	95.80	8.80	90.81
val3a	0.00	0	0.00	**-1.71**	**-19.68**	48.60	**0.00**	**100.00**
val3b	2.30	1	33.33	**-0.88**	15.27	47.20	0.2.	99.58
val3c	5.80	1	14.29	**-9.24**	**-12.97**	93.10	5.10	94.52
val4a	1.50	1	33.33	0.97	24.61	21.10	**0.00**	**100.00**
val4b	1.46	1	25.00	0.20	13.12	37.50	**0.00**	**100.00**
val4c	4.67	1	20.00	0.04	0.17	73.00	**0.00**	**100.00**
val4d	4.44	1	11.11	**-4.96**	**-6.71**	90.20	1.00	98.89
val5a	4.26	1	33.33	2.83	21.21	29.40	**0.00**	**100.00**
val5b	4.48	1	25.00	1.99	7.07	58.70	**0.00**	**100.00**
val5c	4.85	1	20.00	1.35	4.35	61.50	**0.00**	**100.00**
val5d	6.20	1	11.11	**-8.57**	**-16.72**	98.20	0.80	99.19
val6a	0.00	0	0.00	**-2.93**	**-12.28**	42.10	**0.00**	**100.00**
val6b	5.15	1	25.00	0.09	8.70	56.70	0.10	99.82
val6c	6.31	1	10.00	**-10.37**	**-15.94**	98.60	2.40	97.57
val7a	0.00	1	33.33	**-1.74**	19.05	35.50	**0.00**	**100.00**
val7b	0.00	0	0.00	**-0.39**	**-2.15**	11.00	**0.00**	**100.00**
val7c	5.39	1	11.11	**-4.47**	**-7.74**	88.30	0.70	99.21
val8a	2.33	1	33.33	2.02	30.72	6.50	**0.00**	**100.00**
val8b	3.54	1	25.00	3.19	19.90	16.60	**0.00**	**100.00**
val8c	7.21	1	11.11	**-7.71**	**-14.37**	96.80	0.70	99.28
val9a	0.93	1	33.33	0.50	26.98	14.70	**0.00**	**100.00**
val9b	1.84	1	25.00	**-1.06**	2.67	65.80	**0.00**	**100.00**
val9c	2.41	1	20.00	1.84	15.16	19.80	**0.00**	**100.00**
val9d	4.60	1	10.00	**-4.50**	**-12.20**	96.10	0.70	99.27
val10a	1.87	1	33.33	1.67	28.21	12.20	**0.00**	**100.00**
val10b	2.29	1	25.00	0.99	13.64	39.60	**0.00**	**100.00**
val10c	2.91	1	20.00	0.69	3.63	63.60	**0.00**	**100.00**
val10d	4.72	1	10.00	**-3.24**	**-12.07**	95.60	0.10	99.90
Average	**3.29 %**		**19.91 %**	**-1.99 %**	**+2.64 %**	**53.05 %**	**0.65 %**	**96.33 %**

Table 4.9 *– Comparaison des performances relatives des deux approches sur les fichiers val.*

4.7.3 Comparaison de la robustesse des deux approches sur les fichiers egl

La table 4.10 résume l'analyse de sensibilité du GA sur les 24 fichiers egl pour l'approche *serrée*. Nous pouvons remarquer, comme pour les gdb et val, qu'il y a un grand écart entre la solution stochastique et la solution déterministe. Le coût de la solution et le nombre de tournées augmentent sur toutes les instances. L'augmentation du coût estimé pendant les réplications peut atteindre en moyenne 23.04 % et le nombre de tournées 31.29 %.

La table 4.11 montre que les solutions restent stables dans l'approche *relaxée*. La variation est petite : le coût et le nombre de tournées augmentent en moyenne de 0.53 % et 0.67 %. Le pourcentage des réplications générant des tournées supplémentaires est au maximum égal à 42.50 %.

La comparaison des deux approches est présentée dans la table 4.12. Elle permet de conclure que :

- l'approche *relaxée* diminue le coût en moyenne de 11.09 % ;
- l'approche *relaxée* diminue le nombre de tournées en moyenne de 14.04 % ;
- le pourcentage des réplications générant un nombre de tournées supplémentaire est réduit de 96.87 % à 12.97 % (voir figure 4.6) ;
- l'approche *relaxée* améliore presque dans 100 % l'approche *serrée*, le pourcentage moyen d'amélioration de la robustesse est de 86.92%.

file	cost(S_1)	nvu(S_1)	$\overline{\text{cost}(S_1, ne)}$	$\overline{\text{nvu}(S_1, ne)}$	extratrip (S_1,ne)	σ(cost(S_1,ne))	σ(nvu(S_1,ne))
egl-e1-A	3548	5	3916.64	6.18	78.40	294.41	0.86
egl-e1-B	4498	7	5296.14	8.61	86.30	532.53	1.02
egl-e1-C	5595	10	7138.76	13.68	99.30	644.38	1.48
egl-e2-A	5018	7	5800.42	8.86	91.80	464.00	1.07
egl-e2-B	6340	10	7874.31	13.42	98.90	679.86	1.43
egl-e2-C	8415	14	10217.67	17.96	99.50	745.61	1.53
egl-e3-A	5898	8	6746.83	10.39	95.20	480.65	1.23
egl-e3-B	7822	12	9796.73	16.50	99.80	712.91	1.53
egl-e3-C	10433	17	13353.97	23.18	100.00	912.13	1.81
egl-e4-A	6461	9	7629.50	11.50	95.00	596.54	1.26
egl-e4-B	9021	14	11761.04	19.17	100.00	900.99	1.68
egl-e4-C	11779	20	14532.51	25.51	99.90	945.77	1.83
egl-s1-A	5018	7	5908.77	8.53	85.40	627.64	0.99
egl-s1-B	6435	10	7735.15	12.75	96.70	652.70	1.29
egl-s1-C	8518	14	10821.52	18.00	99.10	902.27	1.57
egl-s2-A	9995	14	12308.84	18.91	99.70	866.23	1.71
egl-s2-B	13174	20	17061.55	28.41	100.00	1123.60	2.22
egl-s2-C	16795	27	21851.50	36.97	100.00	1278.06	2.39
egl-s3-A	10296	15	12864.42	20.05	99.90	903.28	1.65
egl-s3-B	14053	22	17504.12	28.77	100.00	1072.17	2.00
egl-s3-C	17297	29	22510.87	39.47	100.00	1276.32	2.42
egl-s4-A	12442	19	14908.24	23.79	100.00	909.12	1.68
egl-s4-B	16531	27	20884.97	36.03	100.00	1183.05	2.26
egl-s4-C	20832	36	27156.85	48.90	100.00	1504.74	2.76

Table 4.10 – *Résultat de l'approche serrée sur les fichiers egl.*

file	cost(S_2)	nvu(S_2)	$\overline{\text{cost}(S_2, ne)}$	$\overline{\text{nvu}(S_2, ne)}$	extratrip (S_2,ne)	σ(cost(S_2,ne))	σ(nvu(S_2,ne))
egl-e1-A	3825	6	3833.31	6.01	1.50	68.36	0.12
egl-e1-B	4805	8	4822.79	8.04	4.40	85.25	0.21
egl-e1-C	6161	11	6219.58	11.11	10.70	186.29	0.32
egl-e2-A	5418	8	5430.94	8.03	2.60	83.83	0.16
egl-e2-B	6791	11	6809.15	11.04	4.30	90.20	0.20
egl-e2-C	9369	16	9467.07	16.2	18.70	223.90	0.44
egl-e3-A	6355	9	6361.31	9.01	1.30	58.99	0.13
egl-e3-B	8590	13	8653.53	13.14	13.70	167.20	0.37
egl-e3-C	11412	19	11488.62	19.17	16.00	187.27	0.40
egl-e4-A	6989	10	6998.36	10.02	1.80	70.61	0.13
egl-e4-B	9791	16	9861.22	16.15	14.10	191.62	0.37
egl-e4-C	12864	22	13037.68	22.35	30.00	309.80	0.60
egl-s1-A	5502	8	5511.16	8.02	2.40	72.37	0.15
egl-s1-B	6913	11	6934.01	11.05	4.60	102.71	0.21
egl-s1-C	9318	16	9350.22	16.06	5.30	146.75	0.24
egl-s2-A	10927	16	10934.66	16.01	1.40	65.72	0.12
egl-s2-B	14660	23	14738.68	23.12	11.30	230.59	0.33
egl-s2-C	18238	30	18513.7	30.51	41.90	406.62	0.68
egl-s3-A	11149	16	11162.94	16.03	2.70	90.42	0.16
egl-s3-B	15140	24	15211.26	24.19	16.90	188.45	0.43
egl-s3-C	19076	32	19310.07	32.54	42.50	369.37	0.71
egl-s4-A	13411	21	13437.76	21.05	5.40	124.01	0.23
egl-s4-B	18117	30	18203.24	30.20	17.70	212.14	0.45
egl-s4-C	22827	40	23077.83	40.51	40.30	392.21	0.71

Table 4.11 – *Résultat de l'approche relaxée sur les fichiers egl.*

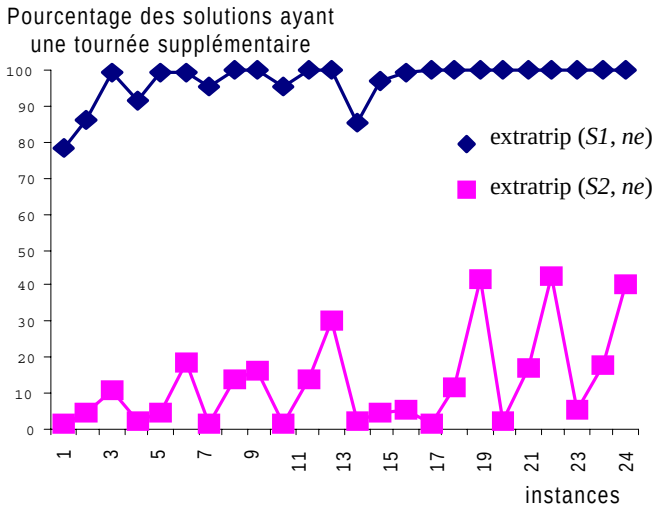


Figure 4.6 – *Comparaison de extratrip*(S_1, *ne*) *et extratrip*(S_2, *ne*) *sur les fichiers egl.*

file	∇cost (S_1,S_2)	extranvu (S_1,S_2)	∇nvu (S_1,S_2)	∇cost (S_1,S_2,ne)	∇nvu (S_1,S_2,ne)	extratrip (S_1,ne)	extratrip (S_2,ne)	improv (S_1,S_2,ne)
egl-e1-A	7.81	1	20.00	**-2.13**	**-2.75**	78.40	1.50	98.09
egl-e1-B	6.83	1	14.29	**-8.94**	**-6.62**	86.30	4.40	94.90
egl-e1-C	10.12	1	10.00	**-12.88**	**-18.79**	99.30	10.70	89.22
egl-e2-A	7.97	1	14.29	**-6.37**	**-9.37**	91.80	2.60	97.17
egl-e2-B	7.11	1	10.00	**-13.53**	**-17.73**	98.90	4.30	95.65
egl-e2-C	11.34	2	14.29	**-7.35**	**-9.80**	99.50	18.70	81.21
egl-e3-A	7.75	1	12.50	**-5.71**	**-13.28**	95.20	1.30	98.63
egl-e3-B	9.82	1	8.33	**-11.67**	**-20.36**	99.80	13.70	86.27
egl-e3-C	9.38	2	11.76	**-13.97**	**-17.30**	100.00	16.00	84.00
egl-e4-A	8.17	1	11.11	**-8.27**	**-12.87**	95.00	1.80	98.11
egl-e4-B	8.54	2	14.29	**-16.15**	**-15.75**	100.00	14.10	85.90
egl-e4-C	9.21	2	10.00	**-10.29**	**-12.39**	99.90	30.00	69.97
egl-s1-A	9.65	1	14.29	**-6.73**	**-5.98**	85.40	2.40	97.19
egl-s1-B	7.43	1	10.00	**-10.36**	**-13.33**	96.70	4.60	95.24
egl-s1-C	9.39	2	14.29	**-13.60**	**-10.78**	99.10	5.30	94.65
egl-s2-A	9.32	2	14.29	**-11.16**	**-15.34**	99.70	1.40	98.60
egl-s2-B	11.28	3	15.00	**-13.61**	**-18.62**	100.00	11.30	88.70
egl-s2-C	8.59	3	11.11	**-15.27**	**-17.47**	100.00	41.90	58.10
egl-s3-A	8.28	1	6.67	**-13.23**	**-20.05**	99.90	2.70	97.30
egl-s3-B	7.74	2	9.09	**-13.10**	**-15.92**	100.00	16.90	83.10
egl-s3-C	10.29	3	10.34	**-14.22**	**-17.56**	100.00	42.50	57.50
egl-s4-A	7.79	2	10.53	**-9.86**	**-11.52**	100.00	5.40	94.60
egl-s4-B	9.59	3	11.11	**-12.84**	**-16.18**	100.00	17.70	82.30
egl-s4-C	9.58	4	11.11	**-15.02**	**-17.16**	100.00	40.30	59.70
Average	**8.87 %**		**12.03 %**	**-11.09 %**	**-14.04 %**	**96.87 %**	**12.97 %**	**86.92 %**

Table 4.12 – *Comparaison des performances relatives des deux approches sur les fichiers egl.*

4.7.4 Synthèse des évaluations

En conclusion, soulignons que les résultats sur les 3 jeux de données confirment l'intérêt de l'approche *relaxée*. Ils prouvent que même si les solutions en fin de la phase déterministe sont moins intéressantes dans l'approche *relaxée*, elles sont moins sensibles aux variations de la demande. Cette caractéristique fait que ces solutions sont plus intéressantes du point de vue industriel.

Ces résultats montrent que, sans modifier la méthode déterministe (dans notre cas le GA) et en limitant le taux de remplissage des véhicules, nous pouvons créer des solutions plus robustes. De cette façon, la méthode de résolution reste opérationnelle même si les données sur la demande à satisfaire sont imparfaitement connues.

		Approche serrée (%)			*Approche relaxée* (%)		
		Moy.	*Max.*	*Min.*	*Moy.*	*Max.*	*Min.*
gdb	*cost*	8.05	19.79	0.10	0.06	0.46	0.00
	nvu	22.68	46.83	0.75	0.11	0.63	0.00
val	*cost*	6.54	22.59	0.00	0.07	0.66	0.00
	nvu	17.31	43.00	0.00	0.09	1.00	0.00
egl	*cost*	23.04	30.37	10.39	0.53	1.51	0.07
	nvu	31.29	42.05	21.85	0.67	1.70	0.06

Table 4.13 – *Déviations moyennes, maximales et minimales des solutions stochastiques par rapport aux solutions déterministes pour les deux approches.*

4.8 Vers un GA intrinsèquement robuste

4.8.1 Caractérisation analytique de la robustesse

L'étude précédente montre qu'on peut obtenir des solutions robustes par le GA mais il ne s'agit pas d'une résolution directe du SCARP. Une extension intéressante de cette étude serait d'adapter le GA pour l'optimisation du coût tout en donnant aux solutions des propriétés de robustesse. Il s'agit en fait d'intégrer la robustesse dans le processus d'optimisation.

L'idéal est de disposer d'une analyse mathématique permettant non pas de travailler avec une moyenne expérimentale calculée sur un nombre fini de réplications mais de calculer l'espérance mathématique de la variable aléatoire représentant le coût d'une solution. De même pour l'écart-type et pour tous les autres critères pertinents pour l'étude. Cela présente deux avantages :

- éviter la phase de simulation très coûteuse pour chaque solution construite par le GA ;
- éviter les imprécisions de la simulation. En effet, le nombre de réplications est limité à cause des temps de calcul, ce qui peut donner une mauvaise estimation de la moyenne. Des erreurs peuvent alors biaiser le fonctionnement de l'algorithme qui refuse des solutions jugées moins intéressantes que d'autre en utilisant la moyenne expérimentale alors que l'utilisation de l'espérance mathématique peut en décider autrement.

Grâce aux hypothèses 4.3, il est possible d'établir analytiquement la probabilité d'avoir un aller-retour supplémentaire au dépôt, et donc de calculer les espérances du coût des tournées et du nombre de tournées.

Soit $cost(S,\omega_i)$ et $r(u,\omega_i)$ des variables aléatoires représentant respectivement le coût de la réplication ω_i et la quantité sur l'arc u dans cette réplication. Rappelons que $r(u,\omega_i)$ suit une loi normale de moyenne $r(u)$ et d'écart-type $k.r(u)$ (avec $0 < k < 1$).

Sous les hypothèses H^{*0} et H^{*1} suivantes :

H^{*0} : un dépassement de capacité ne se produira qu'une fois au plus, et en fin de tournée, le plus souvent au cours de la collecte du dernier arc. *Dans ce cas, le coût supplémentaire* sup_j *pour un véhicule qui doit interrompre sa tournée* θ_j *pour retourner au dépôt, est donné par la formule* (4.1).

$sup_j = d(v, s) + d(s, u)$, avec u la dernière tâche de la tournée et $v = pre(u)$. (4.1)

De H^{*0} découle H^{*1} :

H^{*1} : *une tournée ne peut être interrompue pour un retour au dépôt qu'une fois au plus, et ce juste avant le dernier arc à collecter.*

Soit la tournée θ_j, la somme des demandes déterministes $load(\theta_j) = \sum_{u \in \theta_j} r(u)$ à collecter ne doit pas dépasser Q. Dans le cas stochastique, la somme des quantités aléatoires $\sum_{u \in \theta_j} r(u, \omega_i)$ dépasse Q avec une probabilité p_j donnée dans (4.2).

$$p_j = P\left\{ \sum_{u \in \theta_j} r(u, \omega_i) > Q \right\} = P\left\{ \sum_{u \in \theta_j} (r(u, \omega_i) - r(u)) > Q - \sum_{u \in \theta_j} r(u) \right\} \tag{4.2}$$

Puisque $\sum_{u \in \theta j} (r(u, \omega_i) - r(u))$ suit une loi normale centrée de variance $\sum_{u \in \theta_j} k^2 r(u)^2$, donc d'écart-type $k\sqrt{\sum_{u \in \theta_j} r(u)^2}$, alors la formule 4.3 donne la probabilité d'avoir un aller/retour supplémentaire au dépôt.

$$p_j = \left\{ N(0,1) > \frac{Q - \sum_{u \in \theta_j} r(u)}{k\sqrt{\sum_{u \in \theta_j} r(u)^2}} \right\} = 1 - F\left(\frac{Q - \sum r(u)}{k\sqrt{\sum r(u)^2}} \right) \tag{4.3}$$

F est la fonction de répartition d'une variable aléatoire normale centrée réduite :

$$F(x) = \frac{1}{\sqrt{2.\pi}} \int_{-\infty}^{x} e^{-\frac{t^2}{2}} .dt \tag{4.4}$$

Selon les hypothèses H^{*0} et H^{*1}, nous pouvons conclure que la loi de probabilité du coût de la tournée θ_j, $cost(\theta_j, \omega_i)$, prend les valeurs de la table 4.14.

valeur de $cost(\theta_j, \omega_i)$	$\sum_{u \in \theta_j} c_u$	$\sum_{u \in \theta_j} c_u + sup_j$
Probabilité	$1 - p_j$	p_j

Table 4.14 – *Loi de probabilité de cost*(θ_j,ω_i).

D'après la table 4.14 et les hypothèses H^{*0} et H^{*1}, nous pouvons déduire les formules de l'espérance du coût $\overline{cost(\theta_j, \omega_i)}$ (formule 4.5) et de la variance $\text{var}(cost(\theta_j, \omega_i))$ (formule 4.6).

$$\overline{cost(\theta_j, \omega_i)} = \sum_{u \in \theta_j} c(u) + sup_j \times p_j \tag{4.5}$$

$$\text{var}(cost(\theta_j, \omega_i)) = \text{sup}_j^2 \times (p_j - p_j^2) \tag{4.6}$$

L'optimisation de l'espérance mathématique du coût n'est qu'un exemple. Il faut analyser et comparer le comportement des solutions avec plusieurs fonctions en intégrant par exemple dans la fonction à minimiser la variance ou l'écart-type. On peut ajouter aussi des conditions particulières pour limiter l'augmentation du nombre de tournées.

4.8.2 Adaptation du GA

L'adaptation du GA du chapitre 3 est très simple une fois qu'on dispose de l'espérance mathématique du coût. On choisit au début la valeur de *k*, qui détermine la dispersion des demandes aléatoires. Chaque nouvel enfant obtenu par croisement est d'abord évalué de façon déterministe avec les demandes moyennes $r(u)$. Une solution prévisionnelle est calculée en découpant le chromosome avec la procédure Split (algorithme 3.4 du chapitre 3). On calcule ensuite l'espérance mathématique du coût total avec la formule 4.5.

Le GA stocke ensuite le chromosome avec pour fitness cette espérance mathématique. Au cours de ses itérations, il va donc sélectionner progressivement des solutions de plus en plus robustes. Nous appelons ce genre d'algorithme un GA *intrinsèquement robuste* car la robustesse est gérée pendant les interactions, et non a *posteriori*.

Il faut souligner que cette approche est novatrice car les GA robustes déjà publiés (pour l'optimisation en général, pas pour les problèmes de tournées) se contentent d'une évaluation par simulation de chaque nouveau chromosome, ce qui est très coûteux en temps de calcul (Tsutsui et Ghosh, 1997 ; Branke, 1998).

4.8.3 Evaluations numériques

Nous ne pouvons pas présenter de résultats au moment où nous écrivons ces lignes car les tests sont en cours. Nos résultats préliminaires indiquent déjà une amélioration de la robustesse par rapport à la méthode en deux phases de la section 4.6. Nous prévoyons une communication sur ce GA robuste à la conférence Odysseus en 2003.

4.9 Conclusion

Ce chapitre présente le premier modèle stochastique en tournées sur arcs, où les demandes sont des réalisations de variables aléatoires. Ce problème est noté SCARP. Il trouve son intérêt dans la possibilité d'intégrer lors du processus d'optimisation les incertitudes liées aux données et permet donc de se rapprocher des applications réelles.

Dans le CARP "classique" les demandes sur les arcs sont considérées comme parfaitement connues alors qu'elles ne sont que des estimations statistiques. Une bonne solution du SCARP est d'un point de vue pratique encore plus intéressante qu'une bonne solution du CARP. Les coûts supplémentaires susceptibles d'apparaître sous la forme d'allers-retours au dépôt sont directement pris en compte dans le processus d'optimisation. Estimer les conséquences des variations de la demande sur les tournées est une préoccupation majeure en collecte de déchets car il s'agit de concevoir des tournées peu coûteuses mais surtout robustes et d'estimer aussi cette robustesse.

La conception des tournées se pose de manière cruciale au moment des appels d'offre auxquels les sociétés spécialisées en collecte doivent répondre. Sachant que les réponses fournies les engagent pour plusieurs années, une négligence de l'aspect stochastique peut les

amener à proposer des solutions à des coûts trop faibles, c'est-à-dire à faire des propositions non rentables.

Nous nous sommes intéressés à cet aspect stochastique en tournées sur arcs, en proposant une formulation mathématique du SCARP et une modélisation des demandes aléatoires. L'efficacité du GA (chapitre 3) sur le CARP, a motivé l'analyse des résultats dans un cadre stochastique et l'intégration de la robustesse dans le processus d'optimisation.

Nous avons proposé d'analyser d'abord les résultats du GA sur ce nouveau modèle. Nous avons développé une approche *relaxée*, où le taux de remplissage des véhicules est limité pour faire face aux variations de la demande. Pour évaluer la qualité de cette approche nous l'avons comparé à une autre, appelée approche *serrée* pour laquelle le taux de remplissage des véhicules peut atteindre 100%. Chacune des deux approches est basée sur deux phases. Une phase d'optimisation utilisant le GA déterministe, suivie d'une phase d'évaluation de la robustesse par simulation.

Les résultats montrent que les solutions obtenues avec l'approche *serrée* sont très sensibles aux variations de la demande alors que celles obtenues avec l'approche *relaxée* sont beaucoup moins sensibles.

Par opposition à ces méthodes où la robustesse est constatée a *posteriori*, nous avons également développé une approche novatrice consistant à optimiser la robustesse pendant le GA. Bien que les tests ne soient pas terminés…

Ces travaux représentent une première étape pour traiter des problèmes plus réalistes en tournées sur arcs. Ils ont fait l'objet de deux conférences et d'un article soumis à une revue internationale.

L'extension du modèle est aussi prévue pour la prise en compte des coûts de collecte dépendants des demandes aléatoires et des variations globales des demandes dans le réseau (périodes de fêtes).

Références

G. Fleury, P. Lacomme, C. Prins, W. Ramdane-Chérif, Robustness Evaluation of Solutions for the Capacitated Arc Routing Problem, actes de AIS 2002 (*AI, Simulation and planning in high autonomy systems*), p. 290-295, édités par F. J. Barros et N. Giambiasi. SCS International, Lisbon, Portugal, 07-10/04/2002.

G. Fleury, P. Lacomme, C. Prins, W. Ramdane-Chérif, Étude de la robustesse des solutions pour le CARP. *Roadef 2002* (*Congrès de l'Association Française de Recherche Opérationnelle et d'Aide à la Décision*), p. 178-179, Paris, France, 20-22/02/2002.

G. Fleury, P. Lacomme, C. Prins**,** W. Ramdane-Chérif**,** Improving Robustness of Solutions to Arc Routing Problems, soumis à *JORS* (*Journal of the Operational Research Society*)

CHAPITRE 5
CARP étendu

5.1 Introduction

L'examen des applications en tournées sur arcs montre un grand écart entre problèmes réels et académiques. Suite à cette constatation, nous définissons dans la section 5.2 de ce chapitre un problème de tournées sur arcs plus réaliste que le CARP, le *ECARP* (*Extended CARP*). Il permet de gérer des complications comme des réseaux mixtes et des interdictions de tourner, entre autres. Le ECARP est évidemment extrêmement dur, puisqu'il inclut en cas particulier le CARP, qui est déjà NP-difficile. Nous avons donc conçu des algorithmes dans deux optiques : a) la résolution d'instances petites à moyennes, avec des méthodes de programmation linéaire et un algorithme génétique, et b) la résolution de grandes instances (jusqu'à 1200 arcs), avec des heuristiques constructives rapides.

La section 5.3 consacrée à la première optique commence par la programmation linéaire. Nous formulons le ECARP comme un programme en nombres entiers. Nous illustrons la flexibilité de ce modèle en l'adaptant au cas réel de véhicules avec compacteurs : il s'agit alors de minimiser hiérarchiquement deux critères, la surcharge (totale ou maximale) des véhicules et le coût total des tournées. Nous développons ensuite une méthode de coupes pour le premier modèle. Même quand elle ne trouve pas l'optimum, elle fournit au moins une borne inférieure. Cet apport est déjà appréciable puisque les bornes disponibles pour le CARP de base ne sont plus valides pour le ECARP, à cause des contraintes supplémentaires.

La section 5.3 se poursuit par une généralisation au ECARP du GA du chapitre 3. Nous montrons ensuite que le GA peut également traiter le cas bicritère évoqué ci-dessus. Les bornes inférieures du CARP de base n'étant plus valables, la méthode de coupes est utilisée pour évaluer le GA. Faute d'instances déjà disponibles pour le ECARP, il a fallu construire 11 jeux d'essai, appelés *gdbe*, en modifiant certains fichiers de Golden, DeArmon et Baker (*gdb*) déjà utilisés dans le chapitre 3.

La seconde optique (résoudre de grandes instances) fait l'objet de la section 5.4. Nous y généralisons de façon non triviale trois heuristiques constructives déjà présentées dans le chapitre 3 : *Path-Scanning*, *Augment-Merge* et la *méthode d'Ulusoy*. Comme l'objectif de ces versions étendues est de traiter de grands réseaux comme en collecte de déchets, nous décrivons un générateur aléatoire sophistiqué, capable de construire des graphes imitant les réseaux réels. Ce générateur est utilisé pour produire 15 instances de grande taille (dites *lpr*, comme Lacomme-Prins-Ramdane), sur lesquelles nous évaluons les trois heuristiques.

5.2 Définition du ECARP

Le CARP de la littérature considère un graphe non orienté. Le dépôt coïncide avec le lieu de vidage. La flotte est homogène. Le réseau comprend un sous-ensemble *R* d'arêtes à traiter, dans n'importe quel sens. Chaque arête a une demande et un coût de traversée (égal au coût de traitement et ne dépendant pas du sens de passage). Il faut calculer un ensemble de tournées de coût minimal, traitant tous les arcs de *R*, en respectant la capacité des véhicules. En collecte de déchets, ce CARP est restrictif : réseau de rues à double sens à traiter en tricot et minimisation de la distance parcourue. La plupart des méthodes publiées sont adaptables pour un réseau orienté. Dans ce cas des sens uniques et des rues à double sens à côtés collectables séparément peuvent être modélisées, mais pas des rues collectables en tricot.

Nous proposons un CARP étendu (ECARP) plus réaliste permettant de gérer plusieurs extensions. Cette prise en compte de contraintes supplémentaires est une étape importante dans la construction de solutions industriellement exploitables. Nous visons un CARP étendu où il faut toujours minimiser le coût total des tournées, mais avec les extensions suivantes :

- un multigraphe mixte combinant des arêtes et des arcs,
- des interdictions de tourner et des pénalités pour tourner,
- des coûts de traitement distincts des coûts de traversée,
- des coûts pouvant différer sur les deux sens d'une arête (problèmes venteux)
- un dépôt différent du lieu de vidage,
- plusieurs lieux de vidages,
- un coût fixe associé à l'opération du vidage,
- une limite de coût sur chaque tournée (autonomie).

Ce problème enrichi permet déjà de traiter des réseaux très réalistes de type centre-ville, et de minimiser au choix la distance parcourue ou le temps de travail. Notons que le ECARP est le premier problème de tournées sur arcs permettant de traiter toutes ces contraintes en même temps. Grâce à notre codage du graphe et à nos structures de données flexibles (voir chapitre 2), notre modèle pourrait autoriser aussi d'autres extensions que nous n'avons pas traitées dans nos méthodes de résolution, comme plusieurs dépôts, des points de collecte (demande sur nœuds), des restrictions d'accès pour certains véhicules, une flotte hétérogène et plusieurs produits à collecter (collecte sélective).

5.3 Méthodes pour les ECARP de taille petite à moyenne

Dans cette section, nous travaillons sur un modèle linéaire du ECARP, limité pour simplifier aux quatre premières extensions de la section 5.2, puis nous montrons comment nous pouvons l'adapter pour traiter des contraintes d'autonomie et un cas d'optimisation hiérarchique bicritère posé par les véhicules équipés de compacteurs. Nous commençons par présenter le programme linéaire, ses adaptations, puis l'algorithme de coupes. Pour avoir une base de comparaison, nous procédons ensuite à la généralisation du GA du chapitre 3, aussi bien pour minimiser le coût total des tournées que pour le cas bicritère. Enfin, nous décrivons la génération des premières instances pour le ECARP (fichiers *gdbe*) pour terminer par une évaluation numérique de la méthode de coupes et du GA étendu.

5.3.1 Une formulation linéaire du ECARP

Nous proposons un programme linéaire en nombres entiers pour le ECARP. Il généralise le modèle de Golden et Wong (1981) pour le CARP de base (*cf.* chapitre 1), qui est basé sur la propriété suivante : « *dans un problème de tournées sur arcs dans un réseau non orienté, il existe une solution optimale dans laquelle aucune tournée ne passe plus d'une fois dans chaque sens sur une arête* ». La propriété n'étant plus valable pour le ECARP, le modèle de Golden et Wong doit être profondément modifié.

5.3.1.1 Données

Nous rappelons que le réseau mixte est codé par un graphe interne orienté, que nous utilisons des distances entre arcs et que le dépôt et l'ensemble des vidages sont remplacés par des boucles fictives. Deux arcs opposés u et v sont indépendants si $inv(u) = inv(v) = 0$ mais forment une arête si $inv(u) = v$ et $inv(v) = u$. Comparé au CARP traité dans le chapitre 3, le nombre de véhicules est ici limité à une valeur nva. Les valeurs des coûts, des demandes et des capacités sont ici des nombres réels positifs.

5.3.1.2 Variables de décisions

Le modèle utilise deux groupes de variables, définies sur le graphe interne orienté :

- x_{uvk} : nombre de fois où le véhicule k traverse les arcs u puis v, $v \in suc(u)$
- l_{uk} : variable binaire égale à 1 si l'arc u est traité par le véhicule k, 0 sinon.

Grâce à la propriété en haut de page, le modèle de Golden et Wong utilise des variables binaires x_{uk} valant 1 ssi le véhicule k traverse l'arc u. Pour le ECARP, ces variables sont non-binaires car une tournée optimale peut traverser plusieurs fois un lien dans un sens donné. Ainsi, dans la figure 5.1, le dépôt est au nœud 1 et [3,4] est la seule arête à traiter. Une tournée optimale suit le chemin (1,3,4), traite l'arête dans le sens (4,3) puis retourne au dépôt par le chemin (3,4,1) : l'arête a été traversée 3 fois, dont 2 fois dans le sens (3,4).

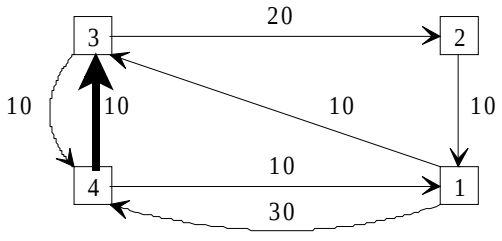


Figure 5.1 – *Un cas de traversée multiple d'arête.*

De plus, nos variables x sont indexées par deux arcs pour traiter les interdictions de tourner de façon transparente. Dans les réseaux réels, le cardinal de $suc(u)$ vaut en moyenne 4, d'où $4m$ tournants permis (u,v), voire moins avec les interdictions de tourner. Les variables l_{uk} ont besoin d'être définies seulement pour les ρ arcs requis. Ainsi, le nombre total de variables est environ $nva \cdot (4m + \rho)$ pour un graphe de type routier.

5.3.1.3 Modèle linéaire

$$\text{Minimiser} \sum_{k=1}^{nva} \sum_{v \in R} w_v l_{vk} + \sum_{k=1}^{nva} \sum_{v \in R} \sum_{u \in pre(v)} c_v (x_{uvk} - l_{vk}) + \sum_{k=1}^{nva} \sum_{v \in A-R} \sum_{u \in pre(v)} c_v x_{uvk} \quad (5.1)$$

$$\forall k = 1...nva, \forall u \in A : \sum_{v \in pre(u)} x_{vuk} = \sum_{v \in suc(u)} x_{uvk} \quad (5.2)$$

$$\forall u \in R,\ inv(u) = 0 : \sum_{k=1}^{nva} l_{uk} = 1 \quad (5.3)$$

$$\forall u \in R,\ inv(u) > 0 : \sum_{k=1}^{nva} (l_{uk} + l_{inv(u),k}) = 1 \quad (5.4)$$

$$\forall k = 1...nva, \forall u \in R : \sum_{v \in suc(u)} x_{uvk} \geq l_{uk} \quad (5.5)$$

$$\forall k = 1...nva : \sum_{u \in R} l_{uk} r_u \leq Q \quad (5.6)$$

$$\forall k = 1...nva, \forall i = 2...n : \sum_{\substack{u \in A \\ b(u)=i}} f_{uk} - \sum_{\substack{u \in A \\ e(u)=i}} f_{uk} = \sum_{\substack{u \in R \\ b(u)=i}} l_{uk} \quad (5.7)$$

$$\forall k = 1...nva, \forall u \in A : f_{uk} \leq n^2 \times \sum_{v \in suc(u)} x_{uvk} \quad (5.8)$$

$$\forall k = 1...nva, \forall u \in A, \forall v \in suc(u) : x_{uvk} \in \mathbb{N} \quad (5.9)$$

$$\forall k = 1...nva, \forall u \in R : l_{uk} \in \{0,1\} \quad (5.10)$$

$$\forall k = 1...nva, \forall u \in A : f_{uk} \in \mathbb{N} \quad (5.11)$$

La fonction-objectif (5.1) cumule les coûts de traitement des arcs et les coûts de traversée de tous les arcs. La distinction entre les deux premières sommes vient du fait que les nombres de traversées x_{uvk} incluent les traversées pour traitement. Les contraintes (5.2) sont des contraintes de conservation du flot pour chaque véhicule sur chaque arc : pour tout véhicule, le nombre d'entrées sur un arc doit être égal au nombre de sorties. Les contraintes (5.3) exprime le fait que chaque tâche-arc doit être traitée par 1 seul véhicule.

Les contraintes (5.4) concernent les tâches-arêtes, codées par deux arcs u et $inv(u)$. Dans ce cas, un seul des deux arcs doit être traité, et par un seul véhicule. Les contraintes (5.5) assurent la cohérence entre le service et le parcours des véhicules : si un arc est traité, alors il doit être traversé au moins une fois. Les contraintes (5.6) garantissent le respect de la capacité des véhicules. Quant aux contraintes (5.7) et (5.8), elles généralisent des contraintes d'élimination de sous-tours illégaux proposées par Golden et Wong. Le nombre total de contraintes (sans les contraintes d'intégrité 5.9, 5.10 et 5.11) est $nva \cdot (2m + n) + \rho \cdot (nva + 1)$, soit environ $9 \cdot n \cdot nva + \rho \cdot (nva + 1)$ pour un réseau routier avec $m \approx 4n$.

Des essais avec un solveur commercial (Xpress) nous ont permis de résoudre seulement de petits cas comme les plus petits *gdb* (7 nœuds, 20 tâches-arêtes). Les contraintes de sous-tours (5.7), (5.8) et (5.11) peuvent être remplacées par les contraintes (5.12) à (5.15), qui empêchent la formation de sous-tours sur chaque sous-ensemble non vide de nœuds V' n'incluant pas le dépôt. Mais, vu le nombre exponentiel de ces sous-ensembles (2^{n-1}-1), cette formulation alternative ne peut pas être résolue directement par un solveur.

$$\forall k = 1...nva, \forall V' \subseteq \{2,3,...,n\}, V' \neq \varnothing : \sum_{i \in V'} \sum_{j \in V'} x_{ijk} - n^2 y_1 \leq |V'| - 1 \quad (5.12)$$

$$\forall k \in 1...nva : \sum_{i \in V'} \sum_{j \notin V'} x_{ijk} + y_2 \geq 1 \quad (5.13)$$

$$\forall k \in 1...nva : y_1 + y_2 \leq 1 \quad (5.14)$$

$$\forall k \in 1...nva : y_1, y_2 \in \{0,1\} \quad (5.15)$$

5.3.1.4 Adaptations du modèle linéaire de base

Le modèle précédent est valable pour un ECARP avec minimisation du coût total des tournées (comme dans le CARP de base) et comprenant les quatre premières extensions (les principales) : un multigraphe mixte, des interdictions de tourner, des coûts de traitement différents des coûts de traversée et des coûts dépendant du sens (problèmes venteux). Nous donnons ici deux exemples d'adaptations à des problèmes voisins pour illustrer sa flexibilité.

a) Limitation de la durée des tournées (autonomie)

Il suffit d'ajouter les contraintes (5.16) qui permettent de vérifier que, pour chaque véhicule, le coût total de la tournée est inférieur à une autonomie donnée L. Cette autonomie peut être liée à la durée légale de travail des équipages ou à la capacité du réservoir de carburant du véhicule. Chaque contrainte comprend trois sommes : le coût total de traitement des arcs requis par la tournée, le coût total des traversées en HLP des arcs requis et, enfin, les coûts de traversée des arcs non requis.

$$\forall k = 1...nva : \sum_{v \in R} w_v l_{vk} + \sum_{v \in R} \sum_{u \in pre(v)} c_v (x_{uvk} - l_{vk}) + \sum_{v \in A-R} \sum_{u \in pre(v)} c_v x_{uvk} \leq L \quad (5.16)$$

b) Optimisation hiérarchique à deux objectifs

Dans certaines applications comme la collecte des déchets ménagers, la capacité des véhicules est souvent une contrainte molle (*soft constraint*). En effet, les véhicules ont des compacteurs qui permettent des surcharges allant de 10 à 20% par rapport à la charge utile spécifiée par le constructeur. En effet, cette dernière est une estimation calculée à partir du volume de la benne et d'une densité moyenne des déchets. Pour cette raison, les solutions avec dépassements de capacité peuvent être acceptables d'un point de vue industriel.

Dans cette partie, nous expliquons comment adapter notre modèle pour minimiser hiérarchiquement deux objectifs : la surcharge des véhicules puis le coût total. On se ramène en fait à un seul objectif $H = \mu \cdot H_1 + H_2$, dans lequel μ est un coefficient de pondération tel que $\mu > max(H_2)$, H_1 la *surcharge* (violation de capacité) et H_2 le coût total des tournées. Pour faciliter la lecture des résultats, nous prenons pour μ une puissance de 10. Ainsi, pour $\mu = 1000$, $H_1 = 11$ et $H_2 = 71$, alors $H = 11071$ représente une surcharge de 11 et un coût total des tournées de 71. Ci-après, nous adaptons le modèle linéaire pour deux interprétations de la surcharge d'une solution : la surcharge totale des véhicules et la surcharge maximale.

Cas de la surcharge totale

Considérons le cas où H_1 est la surcharge totale des véhicules. Le dépassement (algébrique) de capacité pour un véhicule *k* et la surcharge totale sont en théorie données par les expressions (5.17) et (5.18) :

$$Extra(k) = \sum_{u \in R} l_{uk} \times r_u - Q \tag{5.17}$$

$$H_1 = \sum_{k=1}^{nva} Max(Extra(k), 0) \tag{5.18}$$

Une formulation linéaire est possible en comptant uniquement les surcharges positives, grâce à la définition des *Extra*(*k*) comme des nouvelles variables positives (5.19) et au remplacement des contraintes de capacité (5.6) par les contraintes (5.20).

$$\forall k = 1...nva : Extra(k) \in \text{IN} \tag{5.19}$$

$$\forall k = 1...nva : \sum_{u \in R} l_{uk} r_u \leq Q + Extra(k) \tag{5.20}$$

Quant à la fonction-objectif, elle est mise à jour comme suit :

Minimiser

$$\mu \sum_{k=1}^{nva} Extra(k) + (\sum_{k=1}^{nva} \sum_{v \in R} w_v l_{vk} + \sum_{k=1}^{nva} \sum_{v \in R} \sum_{u \in pre(v)} c_v (x_{uvk} - l_{vk}) + \sum_{k=1}^{nva} \sum_{v \in A-R} \sum_{u \in pre(v)} c_v x_{uvk}) \tag{5.21}$$

Cas de la surcharge maximale

Le modèle précédent permet de calculer une solution avec une surcharge totale minimale. Or, la répartition de cette violation sur l'ensemble des tournées peut être inégale. Par exemple, supposons que nous ayons une solution de 3 tournées, une surcharge totale égale à 5, un coût total de 75 et des surcharges des véhicules valant 0, 1 et 4. On peut préférer une solution de même coût total mais plus équilibrée, par exemple avec des surcharges 2, 2 et 1. Calculer une solution la plus équilibrée possible revient à minimiser la surcharge maximale.

En conservant les contraintes du modèle précédent, la surcharge maximale est tout simplement le maximum des variables *Extra*(*k*). Pour linéariser ce *problème min-max*, nous utilisons l'astuce classique qui consiste à introduire une variable positive *Z* servant à majorer les surcharges de chaque véhicule, via les contraintes (5.22). Pour compléter le modèle, il suffit de remplacer le premier terme de la fonction-objectif est remplacé par $\mu \cdot Z$.

$$\forall k = 1...nva : Extra(k) \leq Z \tag{5.22}$$

5.3.2 Méthode de coupes

Nous avons vu qu'un solveur commercial comme Xpress ne peut résoudre que des instances minuscules des programmes linéaires précédents (7 nœuds et 20 tâches-arêtes). Pour résoudre exactement des cas un peu plus gros, deux grands groupes de méthodes de résolution sont possibles : les *méthodes arborescentes* de type *branch-and-bound* et les *méthodes de coupes* ou de *troncatures*. Le principe de ces dernières est de tenter de trouver *l'enveloppe convexe des solutions entières*, c'est-à-dire le plus petit polyèdre contenant toutes les solutions entières du programme linéaire en nombres entiers.

Nous avons choisi de développer une méthode de coupes basée sur notre formulation linéaire du ECARP, avec la seconde version des contraintes de sous-tours (5.12) à (5.15). Malgré leur nombre exponentiel, ces contraintes sont réputées plus fortes que les premières et on peut espérer que très peu d'entre elles seront actives dans une solution optimale.

Les études polyédrales du CARP de base étant déjà très complexes (Belenguer et Benavent, 1998), notre objectif a été de réaliser un algorithme itératif assez simple, dont le principe a été introduit par Dantzig et *al*. (1974) pour le TSP. La méthode consiste à résoudre d'abord une version relaxée *P'* du programme linéaire initial *P*, sans les contraintes de sous-tours. Si la solution optimale de *P'* contient des sous-tours, les contraintes d'élimination correspondantes (coupes) sont ajoutées à *P'*. Ce processus est répété jusqu'à que l'optimum du programme en cours ne contienne plus de sous-tours. Cette solution est alors optimale pour *P*. Si tout se passe bien, la méthode ajoute peu de coupes pour atteindre l'optimum. Cependant, dans le pire des cas, il peut arriver qu'un nombre exponentiel d'itérations soit nécessaire.

Pour définir notre algorithme de coupe, il faut définir l'ensemble des contraintes conservées (c'est à dire la relaxation courante de *P*), puis déterminer une méthode d'identification des contraintes de *P* violées dans la solution optimale *S* du programme courant *P'*.

5.3.2.1 Relaxation de *P*

L'ensemble des contraintes utilisées pour définir *P'* inclut toutes les contraintes de *P* sauf celles d'élimination des sous-tours. Par rapport au 5.3.1.3, on conserve donc les équations (5.1) à (5.6), (5.9), (5.10) et (5.12) à (5.15). La résolution de *P'* peut donc fournir des sous-tours ne passant pas par le dépôt. Notons que nous exigeons cependant que *P'* reste un problème en nombres entiers.

5.3.2.2 Identification des sous-tours

Une solution valide de *P* est un ensemble de tournées passant par le dépôt. Une solution invalide contient des circuits ne passant pas par le dépôt (sous-tours). Pour les identifier, nous utilisons un *algorithme d'exploration de graphe* (voir par exemple Cormen et *al*., 1990), qui va isoler les sous-tours en calculant les *composantes connexes* du graphe défini par les variables x_{uvk} à 1. Cet algorithme est utilisé dans la fonction *GetSubTours*(*S*), qui retourne pour une solution *S* du programme courant *P'* l'ensemble *X* des sous-tours de *S*.

5.3.2.3 Elimination des sous-tours

Chaque sous-tour dans X est effectué par un véhicule k sur un ensemble V' de nœuds. Il peut être éliminé définitivement dans les itérations suivantes si les contraintes (5.12) à (5.15) pour l'ensemble V' et le véhicule k sont ajoutées à P'.

5.3.2.4 Structure générale de l'algorithme de coupes

L'algorithme commence par relaxer le programme linéaire en nombres entiers P avec la procédure *Relaxation*(P) qui élimine toutes les contraintes de sous-tours. La structure générale de notre algorithme (5.1) utilise trois fonctions :

- *Solve*(P) permet de calculer une solution entière pour le problème P. Nous avons utilisé Xpress mais n'importe quel logiciel commercial de programmation linéaire convient.
- *GetSubTours*(S) identifie les contraintes violées de P en utilisant l'algorithme d'exploration de graphe sur la solution S de P et retourne l'ensemble X des sous-tours.
- *Alter*(P,X) ajoute les contraintes d'élimination des sous-tours de l'ensemble X à P.

La boucle principale de l'algorithme de coupes est répétée jusqu'à que S ne contienne aucun sous-tour (dans ce cas S est alors optimale pour P) ou quand un certain critère d'arrêt est vérifié, par exemple un nombre maximal d'itérations, un temps maximal de calcul ou d'autres critères défini par l'utilisateur.

```
Relaxation(P)           //élimination des contraintes de sous-tours de P
while not stop do
   S:= Solve(P)         //retourne la solution entière de P dans S
   X:= GetSubTours(S)   //calcule l'ensemble des sous-tours
   P:= Alter(P,X)       //ajoute les contraintes violées à P
endwhile
return(S)               //résultat : la solution du ECARP
```

Algorithme 5.1 - *Algorithme de coupes pour le ECARP*.

5.3.3 Adaptation de l'algorithme génétique

Nous adaptons l'algorithme génétique du chapitre 3 au ECARP, pour avoir un point de comparaison avec la méthode de coupes. Les trois sous-sections suivantes expliquent comment nous calculons le distancier arc à arc pour les interdictions de tourner, comment gérer les autres contraintes du ECARP et comment traiter la version bicritère présentée en 5.3.1.4 (surcharge maximale puis coût total des tournées).

5.3.3.1 Gestion des interdictions de tourner et distancier entre arcs

Si le réseau contient des interdictions de tourner, nous avons vu en 2.5.2.2 que le chemin optimal de i à j dépend des arcs pour arriver en i et quitter j. Nous calculons dans ce cas deux matrices $m \times m$: un distancier entre arcs D et une matrice P qui permet de stocker un des plus courts chemins entre deux nœuds quelconques u et v. Plus précisément, $d(u,v)$ est le

coût des plus courts chemins de l'arc *u* à l'arc *v* (non inclus) tandis que *p*(*u*,*v*) est l'arc qui précède *v* sur le chemin stocké.

L'algorithme 5.2 adapte l'algorithme de Dijkstra du chapitre 3 (algorithme 3.1) pour calculer les plus courts chemins partant de l'arc *u*, c'est-à-dire la ligne *u* de *D*. Rappelons que le graphe est défini en donnant pour tout arc *u* la liste *suc*(*u*) des arcs adjacents vers lesquels on peut tourner et que *pen*(*u*,*v*) est une pénalité pour tourner de *u* sur *v*. L'algorithme utilise pour tout arc *v* un booléen *fix*(*v*) qui est vrai ssi le label *d*(*u*,*v*) est définitif. Au début aucun arc n'est fixé et tous les chemins partant de *u* ont un coût infini. Chaque itération du *for count* cherche l'arc *v* non fixé de label minimal, le fixe, et met à jour les labels des successeurs.

```
for v := 1 to m do d(u,v) := ∞; fix(v) := false endfor
for each v in suc(u) do d(u,v) := pen(u,v); p(v) := u endfor
for count := 1 to m do
   v      := argmin{d(u,z): fix(z)=false}
   fix(v) := true
   for each z in suc(v) with d(u,v) + c(v) + pen(v,z) < d(u,z) do
      d(u,z) := d(u,v) + c(v) + pen(v,z)
      p(u,z) := v
   endfor
endfor
```

Algorithme 5.2 – *Version arc à arc de l'algorithme de Dijkstra.*

L'algorithme est clairement en $O(m^2)$. On peut faire mieux grâce à une *structure de tas* (Cormen *et al.*, 1990). Il faut alors faire un tas avec les arcs non fixés, avec l'arc de label minimal à la racine. Ce tas initial peut se construire en $O(m)$. Chacune des m itérations du *loop* sort v du tas en $O(\log m)$ puis ajuste le tas en $O(\log m)$ pour chaque successeur mis à jour. Le nombre total de successeurs mis à jour pendant l'algorithme est $O(h)$, $h = \Sigma_u |suc(u)|$ désignant le nombre total de tournants permis (u,v). La complexité devient donc $O(h.\log m)$.

Les matrices *D* et *P* complètes sont donc calculables en $O(mh \cdot \log m)$, en appelant l'algorithme m fois pour chaque arc de départ possible u. Pour un réseau routier, on a en général $m \approx 4n$ et $h \approx 16n$. La complexité chute alors à $O(n^2.\log n)$, expression déjà trouvée dans le chapitre 3 pour un distancier nœud à nœud. A l'ordre près, la présence d'interdictions de tournées n'affecte donc pas la complexité du calcul du distancier pour les graphes routiers.

Notons que contrairement au CARP de base, *D* n'est plus forcément symétrique, à cause des rues en sens unique ou des interdictions de tourner. De plus, la diagonale est en général non nulle puisque $d(u,u)$ correspond au plus court chemin de $e(u)$ à $b(u)$. Enfin, les chemins obtenus sont *non-élémentaires* (ils peuvent passer plus d'une fois par un nœud) mais ils restent heureusement *simples* (ils ne passent pas plus qu'une fois sur un même arc). On peut donc détailler en $O(m)$ le chemin entre un arc u et un arc v grâce à la matrice *P* (algorithme 5.3).

```
z := pred(u,v)
while z <> u
   ecrire (z)
   z := P(u,z)
endwhile
```

Algorithme 5.3 - *Extraction du plus court chemin de l'arc u à l'arc v.*

5.3.3.2 Prise en compte des autres contraintes du ECARP

Grâce à notre codage du réseau comme un graphe complètement orienté, il est facile de gérer des *graphes mixtes*. Entre deux nœuds, on peut avoir un arc (rue à sens unique) ou deux arcs opposés *u* et *v* (rues à double sens). Dans le second cas, on a une arête si $inv(u) = v$ et $inv(v) = u$ et deux arcs indépendants si $inv(u) = inv(v) = 0$. Les arcs parallèles (*multigraphe*) sont également gérables puisque nous désignons les arcs par des index entre 1 et m : rien n'interdit d'avoir deux arcs *u* et *v* avec $b(u) = b(v)$ et $e(u) = e(v)$. Cette possibilité permet de modéliser par exemple des contre-allées parallèles à une avenue comme les Champs Elysées.

Chaque arête étant codée par deux arcs, on peut mettre des coûts dépendant du sens pour traiter des *problèmes venteux*. Les *coûts de traitement distincts des coûts de traversée* étaient prévus dès le départ, voir par exemple la formule de calcul du coût d'une tournée en 2.5.4. En 2.5.2.4, nous avons décrit une technique simple pour gérer des exutoires multiples, en précalculant pour tout arc requis *u* le coût minimal $term(u)$ pour finir une tournée après *u*.

Quant à l'*autonomie*, elle peut être contrôlée dans les procédures d'évaluation des chromosomes. Pour des chromosomes avec délimiteurs, il suffit de découper le chromosome quand on atteint la capacité *Q* ou l'autonomie *L* dans la procédure d'évaluation séquentielle. Pour des chromosomes sans délimiteurs, le graphe auxiliaire utilisé par la procédure *Split* (*cf.* 3.3.3.2) doit ignorer les arcs représentant des tournées de coût dépassant *L*.

A part l'autonomie, qui affecte les procédures d'évaluation, les autres contraintes n'affectent pas directement l'algorithme génétique. en effet, elles sont implicitement gérées grâce au codage et au prétraitement des données. *Ceci souligne le rôle capital joué par les structures de données en optimisation combinatoire : à part une amélioration de complexité, souvent mise en avant, elles facilitent les extensions des algorithmes*.

5.3.3.3 Cas bicritère : surcharge maximale et coût total

Comme notre modèle linéaire, le GA est très flexible. Il peut lui aussi être adapté pour le cas bicritère du 5.3.1.4 avec minimisation de $H = \mu \cdot H_1 + H_2$, H_1 désignant la surcharge maximale et H_2 le coût total des tournées. La principale différence avec le GA du chapitre 3 est que le nombre de véhicules est fixé maintenant à *nva* au lieu d'être une variable de décision. Une possibilité serait de rejeter tout chromosome avec plus de *nva* tournées, mais alors aucune solution ne serait faisable quand $r_{tot} > nva \cdot Q$.

En fait, nous avons modifié le GA du chapitre 3 (version avec délimiteurs de tournées et évaluation séquentielle) pour qu'il refuse des solutions avec plus de *nva* véhicules, mais accepte des véhicules surchargés. Le seul cas restant d'infaisabilité (plus de *nva* tournées) peut apparaître dans la population initiale et par les croisements. En effet, la procédure d'évaluation séquentielle change de tournée à chaque fois que la capacité *Q* est atteinte.

Une telle solution *S* peut être convertie en une solution avec *nva* véhicules et surcharges autorisées grâce à une *procédure de réparation* très simple. Cette procédure commence par trier les véhicules par charges décroissantes, puis vide les $|S| - nva$ dernières

tournées en excès en insérant leurs tâches « au mieux » dans les *nva* premières tournées. *Au mieux* signifie qu'on insère chaque tâche de façon à minimiser la variation de la fonction-objectif *H*.

La principale modification pour que le GA adapté au ECARP puisse traiter le cas bicritère est d'appeler la procédure de réparation à chaque fois que la procédure d'évaluation séquentielle trouve plus de *nva* véhicules. La seule autre modification concerne la recherche locale du 3.5.4. Nous l'appelons maintenant sur un chromosome déjà évalué et réparé. Ce chromosome a donc *nva* tournées et la recherche locale, qui ne crée jamais de tournées, travaille sur une solution réalisable. La modification consiste à accepter les mouvements qui diminuent la fonction bicritère *H*, au lieu de ceux diminuant le coût total des tournées.

5.3.4 Evaluation numérique de la méthode de coupes et du GA

5.3.4.1 Génération des fichiers *gdbe*

Nos instances du ECARP sont dérivées des fichiers *gdb* déjà utilisés dans le chapitre 3 pour le CARP. Nous avons modifié en fait les fichiers *gdb*1 à *gdb*5 et *gdb*14 à *gdb*19. Les nouveaux fichiers ont les mêmes numéros, mais avec un « e » comme ECARP. Le nombre et la capacité des véhicules sont conservés, mais les contraintes suivantes ont été introduites manuellement :

- des interdictions de tourner ;
- des arêtes venteuses (coûts différents sur les deux sens de traversée) ;
- la mixité (remplacer une arête par deux arcs indépendants ou par un seul arc) ;
- des coûts de traitement distincts des coûts sans traitement ;
- et enfin quelques quantités nulles pour avoir de vrais CARP (les *gdb* étant des CCPP).

nom	*n*	*m*	τ	*nva*	*Q*
gdbe1	12	44	21	5	5
gdbe2	12	52	26	6	5
gdbe3	12	44	22	5	5
gdbe4	11	58	19	4	5
gdbe5	13	52	26	6	5
gdbe14	7	42	21	5	21
gdbe15	7	42	16	4	37
gdbe16	8	56	28	5	24
gdbe17	8	56	29	5	41
gdbe18	9	72	21	5	37
gdbe19	8	22	11	3	27

Table 5.1 – *Caractéristiques des fichiers gdbe.*

Les caractéristiques des problèmes *gdbe* sont listées dans la table 5.1. La table 5.2 détaille *gdbe*19, avec pour chaque arc les nœuds de début et de fin, le coût de traversée en HLP, le coût de traitement, la demande et la liste des arcs-successeurs. Dans ce problème, toutes les rues sont à double sens. Toutes sont des arêtes, sauf entre les nœuds 1 et 2 et entre 3 et 7. Tous les arcs sont requis, sauf les arcs 1, 9 et 20. Il est interdit de tourner de l'arc 22 vers

l'arc 19 et de l'arc 3 vers l'arc 18. La figure 5.3 donne une représentation graphique de *gdbe*19.

Caractéristiques	*u*	*b*(*u*)	*e*(*u*)	*c*(*u*)	*w*(*u*)	*r*(*u*)	*suc*(*u*)
Nombre de nœuds : 8	1	1	2	4	5	0	5, 6, 7, 8,12
Nombre d'arcs : 22	2	1	4	3	4	3	13,17
Véhicules : 3	3	1	5	1	2	5	9,14
Capacité : 27	4	1	6	2	8	8	10,15
Dépôt : 1	5	2	3	1	2	4	16,22
	6	2	4	9	10	6	13,17
inv(01) = 0	7	2	5	5	6	1	9,14,18
inv(11) = 0	8	2	7	2	3	9	11,19,20
inv(12) = 0	9	5	7	7	8	0	11,19,20
inv(22) = 0	10	6	8	5	6	5	21
	11	7	3	6	7	8	16,22
∀ *u* ∉ {1,11,12,22}:	12	2	1	4	5	8	1, 2, 3, 4
inv(*u*) = *u* + 11	13	4	1	3	4	3	1, 2, 3, 4
	14	5	1	1	2	5	1, 2, 3, 4
	15	6	1	2	3	8	1, 2, 3, 4
	16	3	2	1	2	4	5, 6, 7, 8,12
	17	4	2	9	10	6	5, 6, 7, 8,12
	18	5	2	5	6	1	5, 6, 7, 8,12
	19	7	2	2	3	9	5, 6, 7, 8,12
	20	7	5	7	8	0	9,14,18
	21	8	6	5	6	5	10,15
	22	3	7	6	7	8	11,20

Table 5. 2 – *Exemple de fichier gdbe* : *gdbe*19.

5.3.4.2 Evaluation de l'algorithme de coupes et de l'algorithme génétique

Quand la méthode de coupes ne trouve pas l'optimum, elle fournit au moins une borne inférieure. L'objectif de l'évaluation est de voir combien de fois les deux algorithmes trouvent l'optimum sur nos fichiers *gdbe* et de mesurer leur écart moyen. Nous considérons ici le ECARP monocritère du 5.3.1, avec minimisation du coût total des tournées.

Les deux algorithmes ont été programmés en Delphi 5 et testés sur un PC sous Windows 98, équipé d'un Pentium III cadencé à 1 GHz. La méthode de coupes appelle le solveur commercial Xpress-MP (*http://www.dashoptimization.com*), grâce à une librairie qui permet d'utiliser ce logiciel d'optimisation à partir de programmes d'application. Le GA a une population de 30 chromosomes et un taux de recherche locale de 10%. Il stoppe quand il atteint le résultat du CPA ou après 40000 croisements.

Les résultats obtenus sont résumés dans la table 5.3. Les cinq premières colonnes indiquent respectivement le nom du problème, le nombre de nœuds, le nombre d'arcs du graphe interne, le nombre de variables et le nombre de contraintes du programme initial relaxé (ces deux dernières valeurs sont données par Xpress en début de résolution). La colonne *CPA* (*Cutting Plane Algorithm*) donne le coût trouvé par la méthode de coupes et la colonne *GA* celle trouvée par l'algorithme génétique. La dernière colonne affiche l'écart de coût en % entre le GA et le CPA. Les astérisques indiquent les solutions optimales. Les parenthèses dans

la colonne CPA signalent les cas où il a fallu stopper la méthode de coupes avant d'avoir une solution sans sous-tours. Dans ce cas, le coût obtenu donne quand même une borne inférieure.

Nom	*n*	*m*	*variables*	*contraintes*	*CPA*	*GA*	*GA/CPA* %
gdbe1	12	44	1441	998	*311	335	7.71
gdbe2	12	52	2473	1466	(379)	400	5.54
gdbe3	12	44	1486	1123	*293	*293	0
gdbe4	11	38	937	742	*297	*297	0
gdbe5	16	52	2257	1664	(422.5)	451	6.74
gdbe14	7	42	2311	1001	*135	*135	0
gdbe15	7	42	2153	1090	*44	*44	0
gdbe16	8	56	3561	2270	(119)	129	8.40
gdbe17	8	56	3561	2265	(90)	91	1.11
gdbe18	9	72	5904	2421	(87)	111	27.59
gdbe19	8	22	481	333	*69	69*	0

Table 5.3 - *Résolution du ECARP par la méthode de coupes et du GA pour les fichiers gdbe.*

Dans la table 5.3, le CPA résout optimalement 6 instances sur 11. Le GA fournit un optimum prouvé dans 5 cas, en atteignant des valeurs optimales du CPA. Pour les autres problèmes, l'algorithme de coupes donne juste une borne inférieure et il est possible que d'autres solutions du GA soient optimales. En tout cas, l'écart moyen GA/CPA est raisonnable : 5.19 %. Le temps de calcul du GA est inférieur à 2 minutes par instance

La figure 5.2 représente une courbe d'évolution typique de la fonction-objectif, pour le problème *gdbe*17. On peut voir que 80% de la descente est réalisée en 50% des itérations et que les améliorations deviennent rares à la fin.

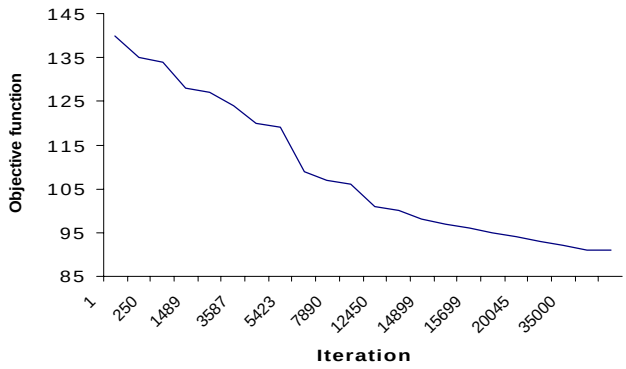


Figure 5.2 - *Courbe d'évolution de la fonction objectif du GA sur le problème gdbe*17.

Nous donnons maintenant un exemple complet d'exécution de la méthode de coupes, à l'aide de la représentation graphique du problème *gdbe*19 de la figure 5.3. La figure donne pour chaque arc son index et, entre parenthèses, les coûts de traversée et de service et la demande (comparer avec la table 5.2). La capacité du véhicule est de 27. Le dépôt est au nœud 1. Les deux interdictions de tourner ont été mises en évidence.

Les trois figures suivantes (5.4, 5.5, 5.6) montrent les solutions successives de la méthode de coupes, avec les sous-tours identifiés. Les traits continus représentent les passages avec traitement et les traits en pointillés les passages en HLP. Le n° sur chaque passage est

celui de la tournée. P'_i désigne le programme linéaire à l'itération i de l'algorithme et S_i la solution obtenue. P'_{i+1} est obtenu par l'ajout des contraintes de sous-tours violées dans S_i.

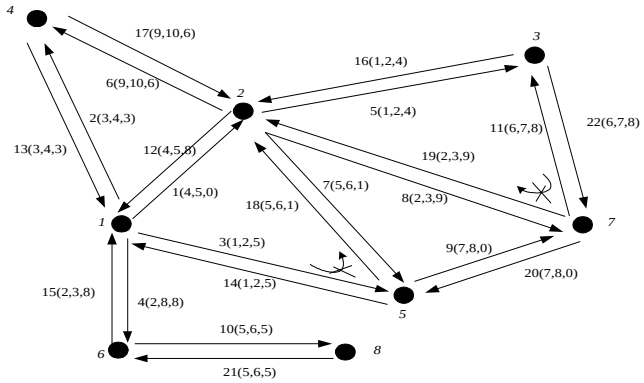


Figure 5.3 - *Représentation graphique du problème gdbe*19.

Sur le programme initial relaxé P'_1 (sans contraintes de sous-tours), Xpress trouve deux solutions entières de coûts 71 puis 69. La seconde contient deux sous-tours (figure 5.4).

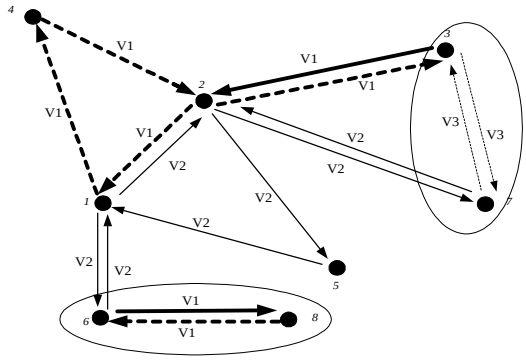


Figure 5.4 - *Solution du programme relaxé initial.*
Coût = 69, *deux sous-tours* : X_{11} = ((7,3),(3,7)) *et* X_{12} = ((6,8),(8,6)).

La résolution de P'_2 trouve encore 69, mais cette fois avec un seul sous-tour (figure 5.5).

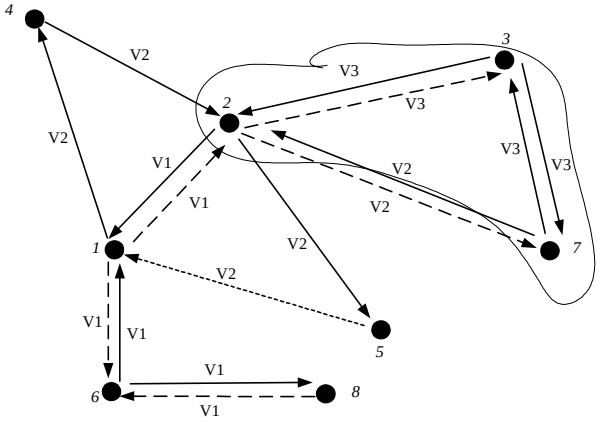


Figure 5.5 - *Solution obtenue à l'itération 2* (*problème* P'_2 : P'_1 + X_{11} + X_{12}).*Coût* = 69, *un sous-tour* : X_{21} = ((2,3),(3,7),(7,3),(3,2)).

Finalement, la résolution de P'_3 donne 4 solutions entières successives valant 77, 75, 71 et 69. L'algorithme s'arrête parce que la dernière ne contient aucun sous-tour. Cette solution est donc aussi optimale pour *P*. Elle est composée de 3 tournées (voir figure 5.6). En les notant comme des listes de nœuds, on a : tournée 1 = (1,6,8,6,1), tournée 2 = (1,4,2,7,2,1) et tournée 3 = (1,2,3,7,2,5,1)

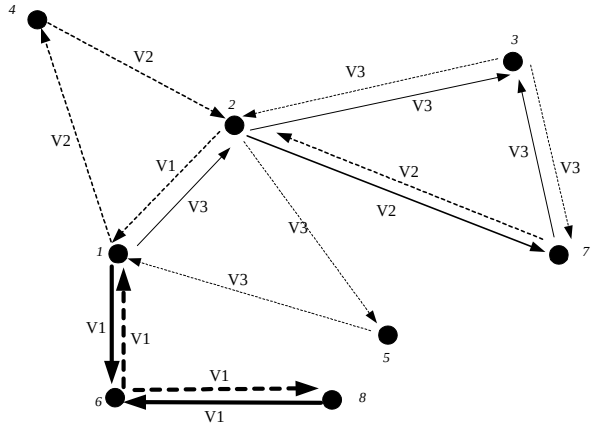


Figure 5.6 - *Solution obtenue à l'itération* 3 *(problème* $P'_3 : P'_2 + X_{21}$ *). Coût* = 69*, pas de sous tours. Solution optimale pour le problème gdbe*19.

5.3.4.3 Optimisation hiérarchique à deux objectives par l'algorithme génétique

Après avoir comparé le GA avec l'algorithme de coupes, nous montrons ici qu'il s'adapte bien au cas bicritère du 5.3.1.4 : minimiser la surcharge maximale *Z* puis le coût total des tournées. La table 5.4 présente les résultats. Les quatre premières colonnes sont copiées de la table 5.1. Le nombre de véhicules disponibles *nva*2 est maintenant volontairement limité pour provoquer des surcharges. Les trois dernières colonnes donnent les résultats : surcharge maximale *Z*, coût total des tournées avec rappel du GA monocritère et temps de calcul en minutes.

Nom	*n*	*m*	*nva*2	*Z*	*coût total des tournées*	*Rappel GA monocritère*	*temps* (*min*)
gdbe1	12	44	3	2	307	335	2.55
gdbe2	12	52	5	1	394	400	4.58
gdbe3	12	44	4	1	293	293*	3.17
gdbe4	11	38	3	2	286	297*	2.25
gdbe5	16	52	5	1	439	451	4.38
gdbe14	7	42	4	2	135	135*	1.51
gdbe15	7	42	3	0	44	44*	1.26
gdbe16	8	56	4	5	127	129	4.34
gdbe17	8	56	4	2	91	91	3.44
gdbe18	9	72	4	0	111	111	2.30
gdbe19	8	22	2	6	71	69*	1.39

Table 5.4 *– Résultats du GA pour le problème bicritère.*

Malheureusement, nous ne disposons pas de solutions optimales ou de bornes inférieures pour comparer nos résultats. On peut cependant penser que le GA donne de bons résultats puisque la possibilité de surcharger les véhicules lui permet toujours de diminuer le coût total par rapport au GA monocritère, sauf sur *gdbe*19.

L'examen de quelques courbes d'évolution de la fonction-objectif nous a aussi montré que la phase de minimisation de la surcharge est rapide : elle prend typiquement 10% du nombre total d'itérations du GA puis l'algorithme se concentre sur la minimisation du coût total des tournées. En particulier, quand il existe des solutions de surcharge maximale nulle, comme dans *gdbe*15 et 18, l'algorithme les trouve en général très rapidement.

5.4 Heuristiques pour les ECARP de grande taille

Seules des heuristiques constructives peuvent trouver rapidement des solutions pour des ECARP de grande taille. Dans cette section, nous généralisons de façon non triviale trois heuristiques efficaces déjà vues dans le chapitre 3 pour le CARP de base : Path-Scanning, Augment-Merge et la méthode d'Ulusoy. Nous appelons les versions étendues *EPS* (*Extended Path-Scanning*), *EAM* (*Extended Augment-Merge*) et *EUH* (*Extended Ulusoy Heuristic*). Elles sont comparées sur des instances de grande taille construites par un générateur *ad hoc*.

5.4.1 Path-Scanning étendu (EPS)

Nous procédons par analogie avec la version de base de Path-Scanning, qui a été détaillée dans l'algorithme 3.2 du chapitre 3 (3.3.1.2). En construisant la tournée courante parvenue en un nœud *i*, cette version cherche à chaque itération le plus proche nœud *j* ayant des arêtes non traitées puis applique un critère de sélection pour choisir la meilleure arête d'origine *j*. La version étendue ne peut pas procéder de cette façon car il est possible que les arcs partant de *j* soient en réalité à une distance supérieure à la distance nœud à nœud, à cause des interdictions de tourner.

En fait, l'heuristique modifiée utilise un distancier arc à arc *D* et prolonge en deux étapes une tournée courante parvenue en un arc *u* : elle calcule la distance minimale *dmin* entre *u* et les arcs non traités puis applique le critère de sélection *F* pour choisir le meilleur arc *v* parmi ceux de distance *dmin*. Les interdictions de tourner sont rendues transparentes grâce aux distances arc à arc. Le graphe mixte et les cas venteux ne posent pas plus de problèmes grâce au codage du réseau sous forme de graphe entièrement orienté. La seule précaution est de mettre l'indicateur de traitement *done*(*inv*(*u*)) à *true* si et seulement si $inv(u) \neq 0$.

Rappelons qu'à tout instant, une tournée non vide θ est fermée, c'est à dire que son coût *cost*(θ) inclut le coût de vidage et de retour au dépôt. Dans la section 2.5.2.4 des notations du chapitre 2, on a vu comment une fonction *term*(*u*) peut calculer simplement le coût de fin de tournée à partir d'un arc requis *u*, même s'il y a plusieurs lieux de vidage. Cette fonction est en *O*(1) si le nombre de vidages est « petit » (borné par une constante), ce qui est le cas en pratique. On suppose qu'elle est appelée automatiquement par les opérations de liste, pour mettre à jour le coût de la tournée en cas d'ajout ou d'enlèvement de la tâche de fin.

En fait, la version étendue de *Path-Scanning* appelle *term* explicitement uniquement pour les critères 1 et 2 (voir 3.3.1.2) et pour l'autonomie. Par exemple, le critère 2 consistait dans l'algorithme 3.2 à minimiser le coût de retour au dépôt $F_2(v) = d(e(v),\sigma)$: il devient maintenant $F_2(v) = term(v)$. Quant à une autonomie L, un arc v peut être ajouté à une tournée θ se finissant par u si $cost(\theta)$-$term(u)$+$d(u,v)$+$w(v)$+$term(v) \leq L$.

L'algorithme 5.4 détaille l'heuristique EPS. Chaque tournée part de la boucle-dépôt *s*. Le *loop* cherche la distance minimale *dmin* entre *u* et les *arcs candidats*, c'est-à-dire les arcs requis, non encore traités et compatibles avec la capacité résiduelle du véhicule et l'autonomie restante. Puis on calcule le candidat *v* minimisant le critère *F*. Pour alléger l'écriture, on définit une fonction booléenne *cand*(*z*) vraie si et seulement si *z* est un candidat réalisable : *cand*(*z*) = (not *done*(*z*)) and ($r(z) \leq Q$-*load*(θ)) and ($cost(\theta)$-*term*(*u*)+*d*(*u*,*z*)+*w*(*z*)+*term*(*z*) ≤ *L*).

```
clear(S)
ndone := 0
repeat
    θ := ∅
    u := s
    loop
      dmin := min{d(u,z): z∈R and cand(z)}
      exit when dmin = ∞
      v := argmin{F(z): z∈R and cand(z) and d(u,z)=dmin}
      done(v) := true
      if inv(v) ≠ 0 then done(inv(v)) := true endif
      enqueue(θ,v)
      u := v
    endloop
    enqueue(S,θ)
    ndone := ndone + |θ|
until ndone = τ.
```

Algorithme 5.4 – *Path-Scanning étendu*

Cet algorithme est exécuté cinq fois en instanciant *F* avec les critères de Golden et Wong décrits en 3.3.1.2 et renvoie la meilleure solution trouvée. La complexité devient clairement en $O(\tau^2)$, c'est-à-dire $O(m^2)$ si tous les arcs sont requis, $O(n^4)$ si *G* est complet, et $O(n^2)$ si *G* est routier. Cette dernière complexité est la même que pour la version de base.

Nous avons aussi testé une version *EPSR* (*Extended Path-Scanning - Randomized*), dans laquelle *v* est choisi aléatoirement parmi la liste des candidats, au lieu d'utiliser les critères de Golden et Wong. Cet algorithme est exécuté un nombre donné de fois et retourne la meilleure solution. Son principal avantage est de pouvoir construire un nombre de solutions supérieur à 5 et de donner de meilleures solutions au prix d'un temps de calcul plus grand.

5.4.2 Augment-Merge étendu (EAM)

Nous révisons la version de base du 3.3.2 (algorithme 3.3), dans laquelle il y a en théorie 8 façons de fusionner deux tournées distinctes $S(i)$ et $S(j)$: on peut concaténer $S(i)$ puis $S(j)$ ou le contraire et inverser ou non chaque tournée. En fait, la version de base se ramène à 4

cas car les coûts d'une tournée et de son inverse sont égaux dans le CARP de base: ainsi, concaténer $S(i)$ et $S(j)$ équivaut à concaténer l'inverse de $S(j)$ et l'inverse de $S(i)$.

Contrairement au CARP de base, une tournée dans un graphe mixte peut contenir des tâches-arcs et dans ce cas elle n'est pas inversible. De plus, une tournée inversible n'a pas nécessairement le même coût que son inverse. On a donc maintenant 8 cas de fusions pour chaque *paire* $\{i,j\}$ de tournées. Nous utilisons deux astuces pour accélérer les calculs.

Premièrement, on balaie en fait tous les *couples* (i,j) de tournées : 4 cas sont testés pour le couple (i,j) et, par symétrie, 4 autres avec le couple (j,i). Deuxièmement, nous évitons les inversions inutiles de tournées grâce à un tableau T parallèle à S : pour toute tournée inversible $S(i)$, la tournée inverse est calculée une seule fois et stockée dans $T(i)$. Par convention, on pose $T(i) = \varnothing$ si $S(i)$ n'est pas inversible.

L'algorithme 5.5 décrit l'initialisation de EAM (construction de la marguerite). Chaque tâche-arête peut être traversée dans deux sens différents et donner deux tournées de coûts distincts. En fait, le sens de traversée d'une tâche-arête dans la marguerite n'a pas d'importance puisque la phase *Merge* va considérer ensuite tous les sens lors du test des fusions possibles. Nous préférons une marguerite de coût minimal pour que chaque pétale respecte la contrainte d'autonomie éventuelle : il serait possible en théorie d'avoir un pétale infaisable pour l'autre sens de traversée.

```
clear(S)
clear(T)
for u := 1 to m with u∈R and done(u)=false do
   if inv(u) = 0 then
      enqueue(S,(u))
      enqueue(T,∅)
    elseif cost((u)) ≤ cost((inv(u))) then
      enqueue(S,(u))
      if cost((inv(u))) ≤ L then enqueue(T,(inv(u))) else enqueue(T,∅)
    else
      enqueue(S,(inv(u)))
      if cost((u)) ≤ L then enqueue(T,(u)) else enqueue(T,∅)
    endif
    done(u) := true
    if inv(u) ≠ 0 then done(inv(u)) := true endif
endfor
```

Algorithme 5.5 – *Initialisation de la marguerite dans EAM*

L'algorithme initialise deux listes vides de tournées S et T puis balaie les arcs requis. Une tâche-arc u donne lieu à une tournée dans S et, comme elle n'est pas inversible, à une tournée vide dans T. Si u code une arête, on crée dans S une tournée avec le sens de coût minimal (toujours réalisable par hypothèse) et on crée dans T la tournée inverse si elle respecte l'autonomie. On mémorise que l'arc u est traité, ainsi que son inverse s'il existe. La notation *enqueue*$(S,(u))$ n'est pas une erreur : d'après nos conventions du chapitre 2, S est une liste de tournées et par conséquent *enqueue* lui ajoute la *tournée* (u), pas *l'arc u*.

Concernant la phase *Augment*, rappelons qu'elle consiste à absorber une tournée avec une seule tâche-arête *u* quand cette arête est traversée en HLP par une plus grande tournée. Dans le CARP de base, le coût total diminue après une telle absorption mais le coût de la tournée absorbante ne change pas car le coût de traversée de *u* est égal à son coût de traitement. L'intérêt de cette phase pour le ECARP nous a paru douteux et ceci est confirmé par les évaluations numériques en fin de chapitre. C'est pourquoi nous distinguons une version d'*Augment-Merge* conservant la phase *Augment* (*EAM*) et une version sans *Augment* que nous notons *EM* comme *Extended Merge*.

L'algorithme 5.6 donne le principe général de la phase *Augment* quand elle est conservée (version *EAM*). Dans le ECARP, il est facile de voir que le coût total diminue aussi à chaque absorption mais que le coût de la tournée absorbante peut diminuer ou augmenter selon les valeurs relatives de $c(u)$ et $w(u)$. En théorie, il faudrait donc rechercher à chaque itération la variation minimale du coût total, mais quelques tests ont donné des résultats décevants. L'algorithme se contente donc de la règle de priorité de Golden et Wong pour le CARP de base : les tournées sont triées par coûts décroissants et, pour $i = 1, 2, \ldots, |S|-1$, S_i tente d'absorber dans l'ordre $S_{i+1}, S_{i+2}, \ldots, S_{|S|}$. Comme les absorptions ne se font pas nécessairement en tête ou en fin de tournées comme dans la phase *Merge*, nous recalculons explicitement les tournées inverses à la fin de la phase *Augmen*t.

```
trier S par coûts décroissants des tournées
for i := 1 to |S| with S(i)≠∅ do
  for j := i+1 to |S| with S(j)≠∅ and load(S(i))+load(S(j)) ≤ Q do
    if S(j,1) est insérable à coût nul dans S(i) à un certain rang k then
       insert(S(i),k,S(j,1))
       S(j) := ∅
    endif
  endfor
endfor
for i := 1 to |S| with S(i)≠∅ do T(i) := inverse(S(i)) endfor
```

Algorithme 5.6 – *Phase Augment dans EAM*

L'algorithme 5.7 concerne la phase *merge*. La boucle *loop* effectue tant qu'elle en trouve des fusions dont la variation de coût total *vmin* est négative. Pour cela, la boucle *for* examine chaque couple (i,j) de tournées et ses quatre cas de fusion. Un cas donné est désormais impossible dans les trois situations suivantes :

- demande totale excessive ($load(S(i))+load(S(j)) > Q$) ;
- le cas nécessite une inversion de tournée impossible à cause d'une tâche-arc ;
- la tournée obtenue après fusion excèderait l'autonomie maximale L.

Pour chaque fusion possible, le *for* calcule la variation de coût *v* et met à jour *vmin* à chaque meilleure variation trouvée. La meilleure fusion est effectuée et l'inverse de la nouvelle tournée est calculé aussitôt dans le tableau *T*. Pour alléger l'écriture, nous utilisons une fonction $sub(i)$ égale à la variation de coût quand on remplace $S(i)$ par $T(i)$ et une fonction booléenne $LOK(i,j,v)$ pour l'autonomie, équivalente à : $cost(S(i)) + cost(S(j)) + v \leq L$.

Implémentées telles quelles, *EAM* et *EM* ont la même complexité que la version de base : $O(\tau^3)$. L'amélioration basée sur une structure de tas et décrite en 3.3.2.3 est encore possible, conduisant à des versions en $O(\tau^2.\log \tau)$, soit $O(n^2.\log n)$ pour un graphe routier.

```
loop
   vmin := 0
   for i := 1 to |S| with S(i)≠∅ do
     for j := i+1 to |S| with S(j)≠∅ and load(S(i))+load(S(j)) ≤ Q do
       //calcule le premier et dernier arc requis de chaque tournée
       bi := S(i,1); ei := last(S(i))
       bj := S(j,1); ej := last(S(j))
       v  := d(ei,bj)-term(ei)-d(s,bj)    //cas 1 : fusion S(i)+S(j)
       if LOK(i,j,v) and (v < vmin) then
         vmin := v; kind := (+i,+j)
       endif
       if T(j)≠∅ then                     //cas 2 : S(i)+T(j)
         v := sub(j) - term(ei) + d(ei,inv(ej)) - d(s,inv(ej))
         if LOK(i,j,v) and (v < vmin) then vmin := v; kind := (+i,-j) endif
       endif
       if T(i)≠∅ then                     //cas 3 : T(i)+S(j)
         v := sub(i) - term(inv(bi)) + d(inv(bi),bj)-d(s,bj)
         if LOK(i,j,v) and (v < vmin) then vmin := v; kind := (-i,+j) endif
       endif
       if T(i)≠∅ and T(j)≠∅ then          //cas 4 : T(i)+T(j)
         v := sub(i)+sub(j)-term(inv(bi))+d(inv(bi),inv(ej))-d(s,inv(ej))
         if LOK(i,j,v) and (v < vmin) then vmin := v; kind := (-i,-j) endif
       endif
     endfor
   endfor
   exit when vmin = 0
   i := abs(kind(1)); j := abs(kind(2))
   if kind(1) < 0 then inverse(S(i)) endif
   if kind(2) < 0 then inverse(S(j)) endif
   S(i) := S(i) + S(j)
   S(j) := ∅
   if T(i) ≠ ∅ and T(j) ≠ ∅ then
      T(i) := inverse(S(i))
   else
      T(i) := ∅
   endif
endloop
```

Algorithme 5.7 *– Phase Merge de EAM*

5.4.3 Heuristique d'Ulusoy étendu (EUH)

5.4.3.1 Version respectant l'orientation des arêtes du tour géant

Dans le chapitre 3, en 3.3.3.2, nous avons proposé avec l'algorithme 3.4 une procédure *Split* qui découpe optimalement un tour géant donné *S* en tournées réalisables pour le CARP. Notre version de base de l'heuristique d'Ulusoy consistait à appeler Path-Scanning avec un critère à la fois et une capacité infinie, à découper les cinq tours géants obtenus avec *Split* et, enfin, à retourner la meilleure des cinq solutions obtenues.

La version étendue *EUH* consiste à remplacer *Path-Scanning* par sa version étendue *EPS* pour le calcul des tours géants et à modifier légèrement *Split* comme dans l'algorithme 5.8. L'adaptation consiste simplement à a) remplacer le coût de retour au dépôt par la fonction *term* pour gérer éventuellement plusieurs exutoires et b) à ajouter des tests pour le respect de l'autonomie *L*. La démonstration de complexité du 3.3.3.2 est toujours valable et la version étendue de *Split* est implémentable en $O(\tau^2)$, c'est-à-dire $O(n^2)$ si *G* est de type routier.

```
V(0),N(0) := 0
for i := 1 to τ do V(i) := ∞ endfor
for i := 1 to τ-1 do
   load, cost := 0; j := i
   repeat
     load := load + r(S(j))
     if i = j then
       cost := d(σ,S(i)) + w(S(i)) + term(S(i))
     else
       cost := cost - term(S(j-1)) + d(S(j-1),S(j)) + w(S(j)) + term(S(j))
     endif
     if (load ≤ Q) and (cost ≤ L) then
       VNew := V(i-1) + cost
       if (VNew < V(j)) or ((VNew = V(j)) and (N(i-1) + 1 < N(j)) then
         V(j) := VNew
         N(j) := N(i-1) + 1
         B(j) := i-1
       endif
       j := j + 1
      endif
   until (j > τ) or (load > Q) or (cost > L)
endfor
```

Algorithme 5.8 – *Adaptation de Split pour l'heuristique d'Ulusoy étendue.*

5.4.3.2 Version calculant les meilleures orientations des arêtes du tour géant

Les algorithmes 3.4 pour le CARP de base et 5.8 pour le ECARP décrivent des versions de *Split* qui respectent à la fois l'ordre des tâches et l'orientation des tâches-arêtes définie par le tour géant θ donné en entrée. Il est possible de concevoir une version améliorée capable de déterminer en plus la meilleure orientation de chaque tâche-arête.

Pour déterminer les orientations d'arêtes donnant un coût minimal pour la tournée, on construit un graphe sans circuit *G*(*i*,*j*) comme celui de la figure 5.7. Ce graphe comprend une entrée (le dépôt), une sortie (l'exutoire, unique dans la figure pour simplifier) et les arcs requis de θ (barrettes de la rangée supérieure). Pour tout arc requis de θ codant une arête, on place en dessous l'arc inverse (barrettes de la rangée inférieure) : c'est le cas de tous les arcs requis de la figure, sauf le troisième qui représente une tâche-arc. Il reste à ajouter des arcs de transition représentant les plus courts chemins dans le réseau réel.

L'algorithme de Bellman permet de calculer un plus court chemin entre le dépôt et l'exutoire dans le graphe *G*(*i*,*j*). Les arcs du plus court chemin (barrettes grises) indiquent s'il

faut conserver les arcs de θ ou prendre l'arc inverse. Le coût du plus court chemin est celui de la tournée correspondante.

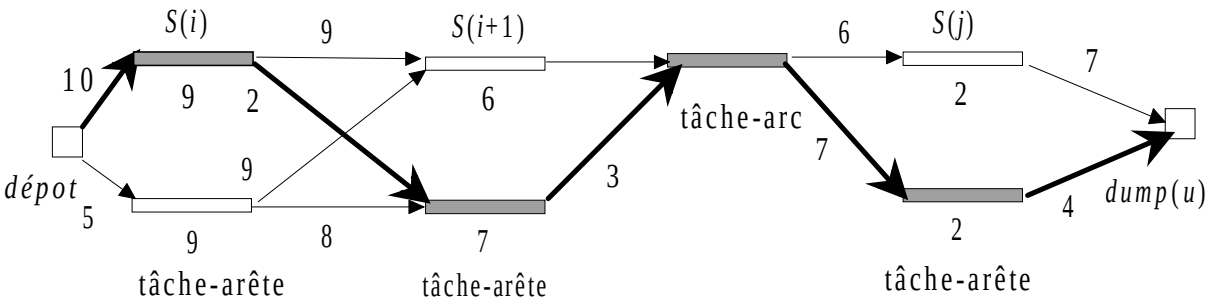


Figure 5.7 – *Recherche des meilleures orientations d'arêtes dans G*(*i*,*j*).

Les instructions affectées dans l'algorithme 5.8 sont celles en face du filet dans la marge, elles concernent le calcul du coût de la tournée (θ_i, θ_{i+1}, …, θ_j). Les nouvelles instructions sont données par l'algorithme 5.9, qui calcule le plus court chemin sans construire le graphe *G*(*i*,*j*) de manière explicite. Le label L1 définit le coût des meilleures séquences se terminant par θ(*j*) tandis que L2 est le coût minimal des séquences se terminant par *inv*(θ(*j*)). L'algorithme est incrémental, c'est-à-dire qu'il met à jour les labels pour j fixé à partir des valeurs des labels pour j-1.

```
if i = j then  //initialise labels pour la 1ère tâche
   //L1 est toujours défini
   L1 := d(s,θ(i)) + w(θ(i)) + term(θ(i))
   //L2 est défini seulement si θ(i) est une arête
   if inv(θ(i)) > 0 then
      L2 := d(s,inv(θ(i))) + w(inv(θ(i))) + term(inv(θ(i)))
   endif
else  //met à jour L1,L2 à partir des valeurs pour j-1
   //sauve valeurs de L1,L2
   P1 := L1; P2 := L2
   //initialise nouveau L1 en allant de θ(j-1) à θ(j)
   L1 := P1 - term(θ(j-1)) + d(θ(j-1),θ(j)) + w(θ(j)) + term(θ(j))
   //essaie de diminuer L1 en venant de inv(θ(j-1)), s'il existe
   if inv(θ(j-1)) > 0 then
      L1 := min(L1,
      P2-term(inv(θ(j-1)))+d(inv(θ(j-1)),θ(j))+w(θ(j))+term(θ(j)))
   endif
   //met à jour L2, défini seulement si θ(j) est une arête
   if inv(θ(j)) > 0 then
      //initialise nouveau L2 en allant de θ(j-1) à inv(θ(j))
      L2 := P1-term(θ(j-1))+d(θ(j-1),inv(θ(j)))+w(inv(θ(j)))+term(inv(θ(j))
      //essaie de diminuer L2 en venant de inv(θ(j-1)), s'il existe
      if inv(θ(j-1)) > 0 then
         L2 := min(L2,P2-term(inv(θ(j-1)))
         +d(inv(θ(j-1)),inv(θ(j)))+w(inv(θ(j)))+ term(inv(θ(j)))
      endif
   endif
endif
cost := min (L1,L2) //meilleur coût pour θ(i)… θ(j)
```

Algorithme 5.9 – *Adaptation de Split pour l'heuristique d'Ulusoy étendue.*

5.4.4 Génération aléatoire des grandes instances LPR

Le test d'algorithmes pour le CARP étendu nécessite de nombreux jeux d'essai pertinents. Or, l'examen d'instances de la littérature pour le CARP montre souvent des graphes non planaires ou de densité trop forte, n'ayant aucun sens dans un contexte routier. De plus, les instances disponibles sont limitées en taille (97 arêtes au plus) et ne concernent que le CARP de base non orienté. Il nous faut donc des instances réalistes, de grande taille, en nombre suffisant et intégrant nos extensions du CARP de base.

Un bon moyen de créer des instances de taille quelconque et en nombre illimité est la génération aléatoire. Le générateur doit être largement paramétrable pour produire divers types de réseaux, dont ceux qui nous intéressent pour la collecte des déchets :

- planaires,
- avec des nœuds de coordonnées connues, pour les représentations graphiques,
- fortement connexes,
- imitant les réseaux routiers réels.

La suite décrit les grandes lignes suivies dans la génération aléatoire des LPR.

5.4.4.1 Génération d'un squelette planaire et non orienté

Un réseau est au départ une grille de grandes mailles carrées, définie par un nombre de mailles verticales (*nsqv*) et un nombre de mailles horizontales (*nsqh*). Les carrés ont un côté de longueur *side* fixée, en m. Au début, seuls les (*nsqv*+1)×(*nsqh*+1) nœuds aux coins des carrés existent. Leurs coordonnées sont définies dans un repère orthonormé dont l'origine correspond au coin supérieur gauche, l'axe des *x* aux déplacements horizontaux et l'axe des *y* aux déplacements verticaux. La figure 5.8 est un exemple avec *nsqv* = 2 et *nsqh* = 3 :

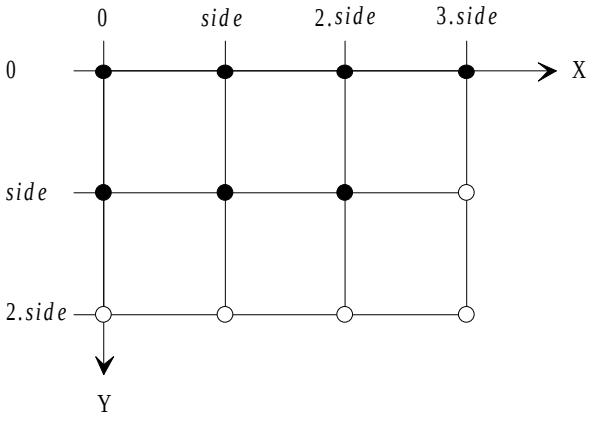


Figure 5.8 – *Exemple de squelette de départ avec nsqv* = 2 et *nsqh* = 3.

Nous considérons les quatre modifications de maille suivantes : coupure en deux parties de surface égale verticalement (type V), horizontalement (H), le long de la 1ère diagonale (D1) et le long de la seconde diagonale (D2). Notez que les modifications V et H créent deux nœuds supplémentaires.

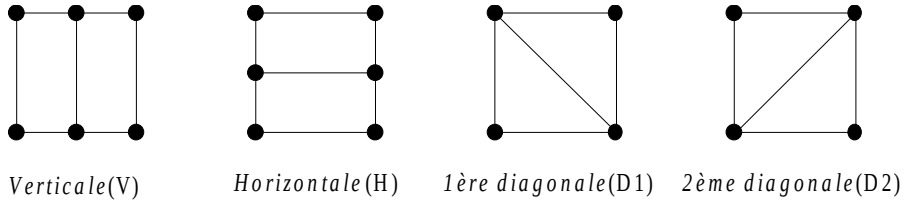


Figure 5.9 – *Les modifications possibles d'une maille.*

Pour chaque maille de la grille, on applique indépendamment et avec des probabilités fixées P_V, P_H, P_{D1} et P_{D2} chacune des modifications. Une maille peut ainsi subir entre 0 et 4 modifications. Notons qu'un nœud central est créé à partir de 2 modifications pour préserver la planarité. La figure 5.10 donne des exemples de modifications multiples.

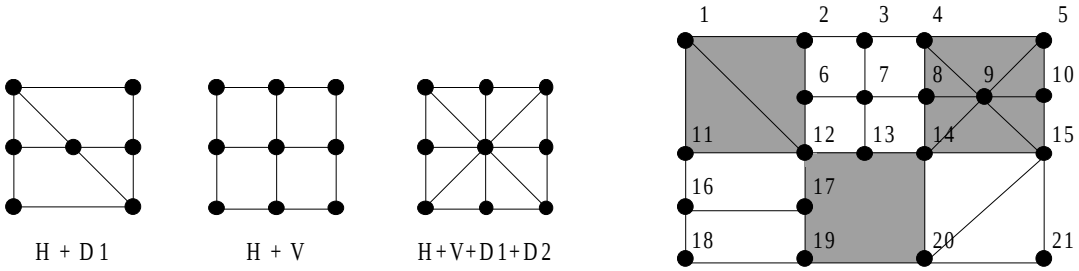


Figure 5.10 – *Exemple de modifications multiples de mailles et de graphe modifié.*

Nous obtenons après modifications de mailles un graphe non orienté, planaire par construction, avec (2.*nsqv*+1)×(2.*nsqh*+1) nœuds au maximum. Les nœuds sont définitifs et peuvent être numérotés dans l'ordre naturel de lecture, ligne par ligne et colonne par colonne. La droite de la figure 5.10 donne un exemple possible obtenu à partir du réseau initial de la figure 5.8, les grisés rappelant les mailles de départ.

Ce graphe trop régulier est déformé grâce à un paramètre *MoveFrac* égal à la fraction de côté de maille dans laquelle nous pouvons déplacer un nœud. Nous déplaçons donc chaque nœud aléatoirement dans un cercle de rayon *side*×*MoveFrac* comme dans l'exemple de la figure 5.11. Les déplacements doivent être limités (10% de *side* par exemple) pour éviter des croisements d'arêtes dans la représentation graphique. A ce stade, les longueurs provisoires des arcs (en mètres) sont calculées à partir des coordonnées définitives des nœuds. La figure 5.12 donne un exemple de graphe après déformation.

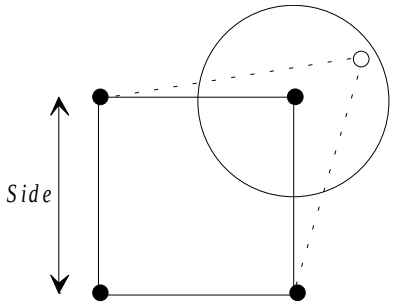


Figure 5.11 – *Un déplacement de nœud dans un cercle de rayon 0.5×side.*

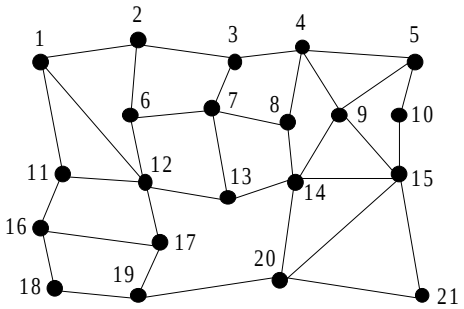


Figure 5.12 – *Graphe obtenu après déplacement des nœuds.*

Nous supposons les nœuds ponctuels, ce qui fait que les deux arcs codant une arête ont la même longueur. Pour simuler des rues incurvées (sans les dessiner), nous augmentons aléatoirement les longueurs avec un pourcentage tiré dans un intervalle donné [0,*GrowFact*]. Pour simplifier, nous appliquons le même pourcentage aux deux arcs d'une arête, ce qui maintient des longueurs égales.

Un graphe encore moins régulier peut être obtenu grâce à des *clusters*. Un cluster est un sous-rectangle des mailles carrées initiales dans lequel les probabilités des transformations V, H, D1 et D2 sont différentes. On peut ainsi simuler des zones plus ou moins urbanisées. Le graphe de la figure 5.13 est un cas extrême : les probabilités de modification utilisées pour les mailles valent 1 pour le cluster des deux mailles grisées et 0 pour les autres mailles.

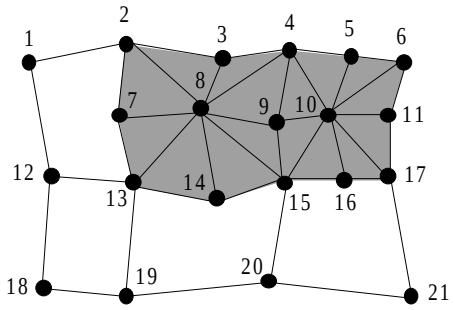


Figure 5.13 - *Graphe avec deux clusters.*

5.4.4.2 Introduction des sens uniques

Le graphe obtenu a pour l'instant des rues à double sens. Le pourcentage désiré de rues à sens unique est défini par une constante *p*1*dir*. Supprimer arbitrairement avec une probabilité *p*1*dir* un des arcs de chaque arête donne en général un graphe non fortement connexe. La forte connexité est garantie par la technique suivante, qui calcule un circuit non simple (et donc non élémentaire) d'arcs dits *protégés*, qui visite tous les nœuds du graphe. Les suppressions d'arcs pour créer des sens uniques ne pourront ensuite se faire que sur les arcs non protégés.

Nous effectuons une *exploration en profondeur* du graphe à l'aide d'une pile *P*, en partant du nœud 1 (coin supérieur gauche). Les arcs empruntés par l'exploration sont dits protégés et forment une arborescence. A chaque fois que l'exploration bute sur une feuille *i* (tous les nœuds voisins étant déjà marqués), nous cherchons un arc de *i* vers son plus ancien successeur *j* dans *P* (c'est-à-dire un nœud *j* voisin de *i*, rencontré en 1er quand on balaie le contenu de *P*). L'arc (*i*,*j*) reçoit aussi un statut protégé, bien qu'il ne soit pas dans l'arborescence d'exploration.

En fin d'exploration, nous cherchons le 1er nœud *i* au sud de 1 et nous protégeons l'arc (*i*,1) s'il n'est pas déjà protégé. On peut montrer que les arcs protégés forment un circuit (en général non simple) sur tous les nœuds. De plus, ce circuit ne protège jamais deux arcs opposés (*i*,*j*) et (*j*,*i*) simultanément, ce qui fait qu'une création de sens unique est possible entre toute paire de nœuds. Ci-dessous, dans la figure 5.14, nous supposons que l'exploration essaie les arcs de chaque nœud dans le sens des aiguilles d'une montre, en commençant au nord. Les numéros donnent l'ordre de visite au moment où on bute sur la 1ère feuille, au rang 14. Les arcs dans la pile ont été protégés et sont indiqués en traits gras.

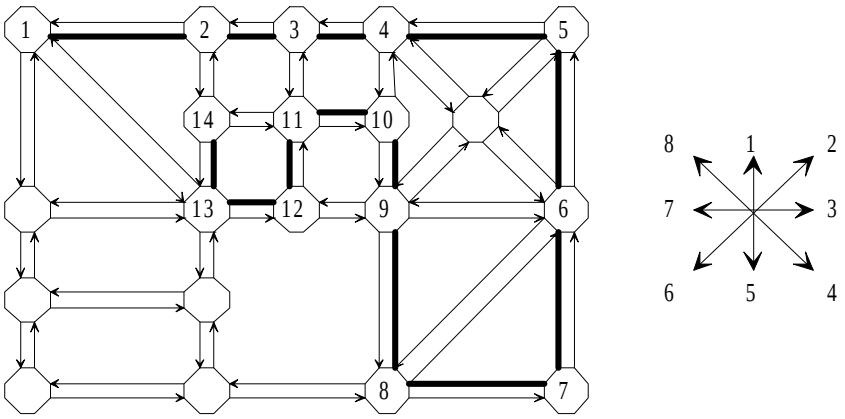


Figure 5.14 – *Recherche des arcs protégés*.

Nous protégeons l'arc reliant la feuille (rang 14) et son successeur en pile le plus ancien (rang 2), dans le but de former un circuit sur les nœuds 2 à 14. Une autre méthode serait de protéger (14,13) mais cela empêcherait tout sens unique entre ces deux nœuds. L'exploration en profondeur se poursuit en dépilant jusqu'au nœud 13, nœud le plus récent ayant encore des successeurs non visités. Elle butte ensuite au rang 19 : on protège donc l'arc (19,1) reliant le nœud de rang 19 à son plus ancien successeur en pile (rang 1). On a ensuite un backtrack au nœud de rang 9, ce qui permet d'atteindre le dernier nœud (rang 20). On protège finalement (20,4) pour former un circuit passant par la dernière feuille. La figure 5.15 montre le circuit protégé obtenu.

Il ne reste plus qu'à décider des sens uniques : pour chaque arête (rue à double sens), on décide avec une probabilité *p*1*dir* si elle est mise en sens unique. Si oui, on supprime l'arc non protégé (si l'arête a un arc protégé) ou bien on tire à pile ou face un arc à supprimer (si l'arête n'a aucun arc protégé).

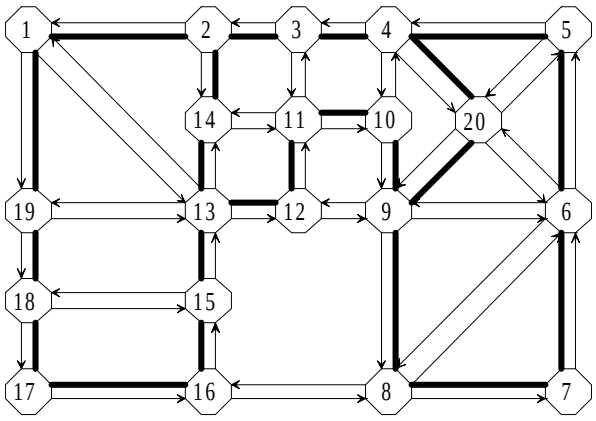


Figure 5.15 – *Circuit protégé garantissant la forte connexité.*

Les premiers tests ont produit des configurations peu réalistes, comme la présence alternée de rues à double sens et à sens unique autour du graphe. La génération préserve donc un « boulevard périphérique » à double sens autour du graphe, sauf quand on veut un graphe avec 100% de sens uniques.

Un autre exemple de cas aberrant est quand un nœud *j* n'est relié qu'à deux nœuds *i* et *k*, par deux paires d'arcs opposés. Si, par exemple, on garde (*i*,*j*) et (*j*,*i*) mais qu'on supprime (*k*,*j*), le seul moyen d'atteindre (*j*,*i*) est un demi-tour obligatoire de (*i*,*j*) vers (*j*,*i*). La génération évite en fait ce type de demi-tours forcés. Ces ajustements expliquent que le nombre final de sens uniques est souvent sensiblement inférieur au nombre initial de rues à double sens, multiplié par *p*1*dir*.

5.4.4.3 Sélection des tâches, des quantités et des arêtes

Cette sélection est pilotée par les paramètres suivants : le pourcentage de côtés de rue à traiter *prside*, le pourcentage de tâches avec des quantités nulles *prside*0, l'intervalle entier [*QMin*,*QMax*] dans lequel on tire la quantité (en kg) pour une tâche non nulle, la probabilité *pedge* qu'une rue à double sens ayant au moins un côté requis soit une vraie arête.

Un arc est traité avec une probabilité *prside*. S'il n'est pas traité, il reçoit une quantité nulle. S'il est traité, il peut devenir une *tâche nulle* de quantité 0 avec une probabilité *prside*0. De telles tâches permettent de modéliser des visites d'arcs sans livraison ni collecte d'un produit, comme dans l'inspection de lignes à haute tension. Les tâches non nulles reçoivent une quantité dans [*QMin*,*QMax*]. On décide avec une probabilité *pedge* si une rue bidirectionnelle, avec au moins un côté à traiter, va former une arête. Si oui, on lie les deux arcs avec les pointeurs *inv* et on leur affecte la somme des quantités des deux côtés.

5.4.4.4 Autres données

La capacité des camions est donnée par une constante *Capa* en kg. Le coût de traversée d'un arc est une durée en secondes calculée à partir de la longueur et d'une vitesse moyenne donnée *TravSpeed*. Notez que la symétrie des longueurs implique celle des durées de traversée.

Le coût de traitement est calculé à partir de la quantité, de la charge moyenne des poubelles *BinLoad* et de leur durée moyenne de traitement *TimePerBin* : on calcule d'abord le nombre fractionnaire de poubelles dans la rue, puis la durée totale en secondes pour les traiter. On ajoute au résultat le temps de traversée, pour tenir compte du déplacement du camion pendant la collecte. Avec nos hypothèses, le coût de traitement est supérieur ou égal au coût de traversée et les coûts de traitement sont égaux pour les deux arcs d'une arête.

Le placement du dépôt est de type *Corner* (au nœud 1) ou *Central* (à peu près au centre du réseau). Il y a *NDumps* ≤ 5 sites de vidages. Le placement de ces sites est de type *Depot* (vidage au dépôt, valide si *NDumps* = 1), *Corner* (placement aléatoire sur les coins sauf celui éventuellement utilisé par le dépôt) ou *Random* (placement aléatoire sur des nœuds distincts et excluant le dépôt). La durée d'une opération de vidage est définie par *DumpCost*.

5.4.4.5 Les fichiers générés

Nous avons construit avec le générateur 15 fichiers LPR (Lacomme-Prins-Ramdane) dont les caractéristiques sont listées dans la table 5.7. Ils sont répartis en 3 sous-groupes A, B et C de 5 fichiers chacun. Les 5 instances de chaque groupe ont environ 50, 100, 300, 500 et 800 tâches. La figure 5.18 montre une représentation agréable (on y roule à droite !) du plus petit fichier, le *lpr-b*-01, qui contient 28 nœuds, 68 arcs internes et 50 tâches. Le plus gros fichier, *lpr-c*-05, a 369 nœuds, 1228 arcs et 803 tâches.

Le *groupe A* concerne des villes modernes à avenue larges (style USA), avec une majorité de rues à double sens et très peu d'arêtes. On peut parler de *graphes majoritairement orientés mais symétriques*. Le *groupe B* concerne des centres-villes anciens, avec une majorité de sens uniques. On peut parler de *graphes majoritairement orientés mais asymétriques*. Enfin, le *groupe C* concerne des quartiers résidentiels à faible circulation (pavillons), avec une majorité de rues à double sens et collectées en tricot. On peut parler de *graphes majoritairement non orientés*, se rapprochant du CARP de base de la littérature.

Le rectangle initial va de 2×3 à 9×12 mailles carrées avec un côté *side* = 200 m. Les probabilités de couper une maille horizontalement ou verticalement sont identiques (P_H et P_V), ainsi que celles pour couper en diagonale (P_{D1} et P_{D2}). On distingue des modifications *denses* (P_H = 0.70, P_{D1} = 0.20) ou *peu denses* (P_H = 0.20 et P_{D1} = 0.10) et 4 types de clusters : *None* (aucun cluster), *OneDense* (cluster dense entouré d'une périphérie peu dense), *OneLight* (cluster peu dense entouré d'une périphérie dense) et *Two* (coupure du graphe en quatre sous-rectangles, avec deux rectangles denses à coins opposés et deux peu denses).

Le déplacement des nœuds est défini par *MoveFrac* = 0.10 (cercles de rayon 20m). Le pourcentage d'augmentation des longueurs est *GrowFact* = 0.10. Les coûts de traversée sont calculés à partir des longueurs et d'une vitesse *TravSpeed* = 20 km/h. Toutes les tâches ont des quantités non nulles (*prside*0 = 0). Les quantités sont définies par *QMin* = 10 et *QMax* = 400. La charge moyenne par poubelle est *BinLoad* = 10 kg, avec une manutention *TimePerBin* de 10 secondes. Ce temps très bas s'explique par les deux ouvriers (rippeurs) qui assurent en général la manutention des poubelles : chacun d'eux met en fait 20 secondes par poubelle. La capacité des camions vaut *Capa* = 10000 kg.

La génération des différents fichiers dépend donc des paramètres restants : *nsqv*, *nsqh*, *p*1*dir*, *prside*, *pedge*, le placement du dépôt et le placement des lieux de vidage. Le tableau 5.7 précise les valeurs de ces paramètres pour chacun des 15 fichiers.

5.4.5 Evaluation numérique des heuristiques

Les heuristiques étendues EPS, EAM, EM (EAM sans la phase Augment) et EUH ont été programmées en Delphi 5 sur un PC à 1 GHz puis exécutées sur les grandes instances LPR. Comme ces algorithmes pour le ECARP peuvent résoudre en particulier le CARP de base, nous les avons également appliqués aux instances classiques *gdb*, *val* et *egl*, déjà décrites et utilisées dans le chapitre 3 sur le CARP de base.

La table 5.5 et la figure 5.16 concernent les 15 fichiers LPR. Pour chaque problème, la table rappelle les paramètres de taille puis indique une borne inférieure (LB), les coûts calculés par les quatre heuristiques puis la déviation en % entre la meilleure heuristique (signalée par une astérisque) et la borne. Les trois dernières lignes *Average*, *Worst* et *Best* donnent respectivement pour chaque heuristique l'écart moyen par rapport à *LB*, le pire écart à *LB* et le nombre de fois où l'heuristique est la meilleure. La figure 5.16 représente graphiquement les écarts aux bornes pour chaque problème et chaque heuristique.

Nom	*n*	*m*	τ	*LB*	*EPS*	*EAM*	*EM*	*EUH*	*dev .LB*
LPR-A-01	28	94	52	13477	13601	13670	13504*	13537	0.20
LPR-A-02	53	174	104	27740	29228	28960	28704	28580*	3.03
LPR-A-03	146	502	304	75670	79285	78431	77280*	78151	2.13
LPR-A-04	195	685	503	126498	132949	130449*	130718	131874	3.12
LPR-A-05	321	1114	806	202174	215416	209724*	210890	212167	3.73
LPR-B-01	28	68	50	14764	14952	14944	14877	14868*	0.70
LPR-B-02	53	126	101	28141	29505	29044	29224	28947*	2.86
LPR-B-03	163	387	305	76889	80558	79972	79808*	79910	3.80
LPR-B-04	248	590	501	126468	133530	130447*	130898	132241	3.15
LPR-B-05	401	913	801	208220	223535	216276*	218375	219702	3.87
LPR-C-01	28	91	50	18639	18874	19038	18855	18706*	0.36
LPR-C-02	53	178	100	36339	36870	37466	37042	36760*	1.16
LPR-C-03	163	557	302	111107	115421	115242	114354*	114493	2.92
LPR-C-04	277	966	504	168327	174655	175207	171583*	173153	1.93
LPR-C-05	369	1228	803	257031	268125	266131	263472*	266001	2.51
Average					4.08 %	3.22 %	2.71%	2.96 %	2.36 %
Worst					7.35 %	4.39 %	4.87	5.51 %	3.87 %
Best					0	4	6	5	0.20 %

Table 5.5 – *Résultats des heuristiques sur les fichiers LPR*

La borne inférieure a été calculée par Belenguer et Benavent à l'aide d'une méthode de coupes très récente (juin 2002) et non encore publiée. Cette méthode est basée sur une nouvelle formulation du CARP mixte sous forme d'un programme linéaire en nombres entiers. Comme cette borne est encore en cours de développement, les valeurs fournies sont provisoires et pourront probablement être améliorées.

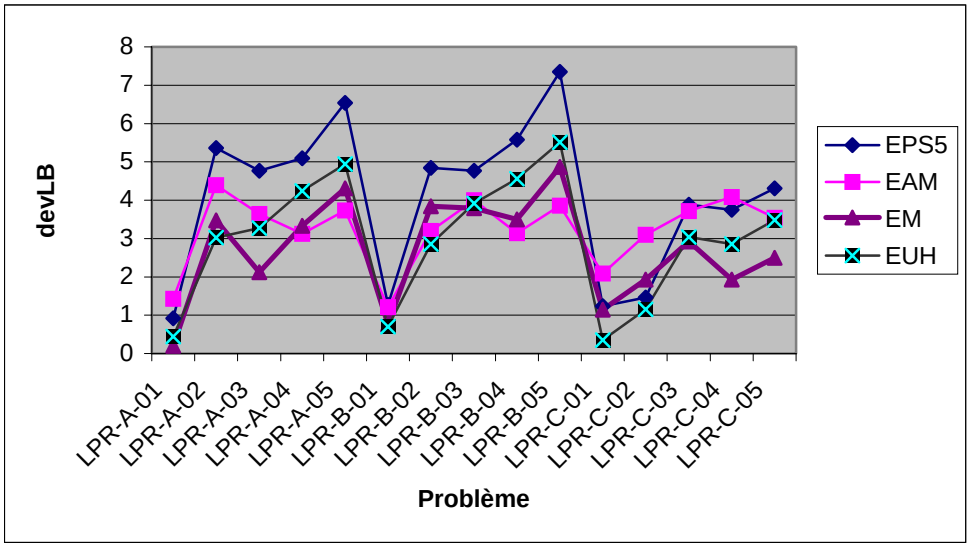


Figure 5.16 – *Déviations par rapport à LB en % par problème et par heuristique.*

On peut constater que la meilleure heuristique est EM, avec un écart moyen à *LB* de 2.71 %, suivie de EUH avec un écart de 2.96 %. L'étude du comportement de cet écart par problème et par heuristique dans la figure 5.16 montre que EM et EUH ont pratiquement la même performance pour les premiers fichiers (les plus petits) de chaque série A, B et C mais que EM surpasse EUH quand la taille augmente. L'heuristique la plus simple, EPS, est également la plus mauvaise : elle ne trouve jamais l'optimum.

La plus grande surprise vient du fait que l'heuristique de fusion de tournées EAM devient meilleure en moyenne quand on supprime la phase Augment (version EM). On peut même voir sur la figure 5.16 que la phase Augment a un rôle franchement nuisible sur les instances du groupe C, qui ont une majorité d'arêtes.

Ces tendances sont confirmées quand on ajoute les fichiers *gdb*, *val* et *egl*, voir le tableau 5.6 et la figure 5.17. La méthode d'Ulusoy étendue EUH est recommandée sur les instances jusqu'à 100 tâches comme les *gdb*, les *val* et les plus petits *lpr*. EM devient le meilleur algorithme sur les instances plus grandes.

La phase Augment de EAM a peu d'effet sur les *gdb* et les *val* : légère dégradation pour les *gdb*, amélioration mineure pour les fichiers *val*. Elle est nuisible, nous l'avons vu, sur les *lpr* et donne carrément des résultats désastreux sur les fichiers *egl*. Nous avons cherché des explications à ces phénomènes. En fait, la phase Augment qui a été décrite en 3.3.1 a été conçue par Golden et Wong pour des CCPP comme les *gdb* et les *val*, dans lesquels toutes les arêtes sont à traiter. Dans ce cas, les tâches-arêtes absorbées par une tournée forment des chaînes non fragmentées par des parcours en HLP.

Par contre, quand tous les liens ne sont pas à traiter, les tournées absorbent souvent des tâches-arêtes non adjacentes au cours de la phase Augment. Ceci crée des parcours en HLP qui sont ensuite irrécupérables par la phase Merge, qui concatène des tournées seulement par leurs extrémités. La situation empire sur les *egl* dans lesquels la demande moyenne est élevée par rapport à la capacité des véhicules (les tournées ont en moyenne 6 à 7 tâches) : Augment sature les véhicules et laisse peu de fusions réalisables pour Merge.

Fichiers	EPS		EAM		EM		EUH	
	Dev	CPU	Dev	CPU	Dev	CPU	Dev	CPU
GDB	10.98	9	7.87	8	6.67	7	4.31	8
VAL	16.19	16	11.01	20	12.71	28	10.76	44
EGL	26.02	42	23.10	13	6.70	114	14.30	37
LPR	4.08	217	3.22	201	2.71	611	2.96	666

Table 5.6 – *Comparaison des heuristiques sur les fichiers gdb, val, egl et lpr* (*temps en ms*)

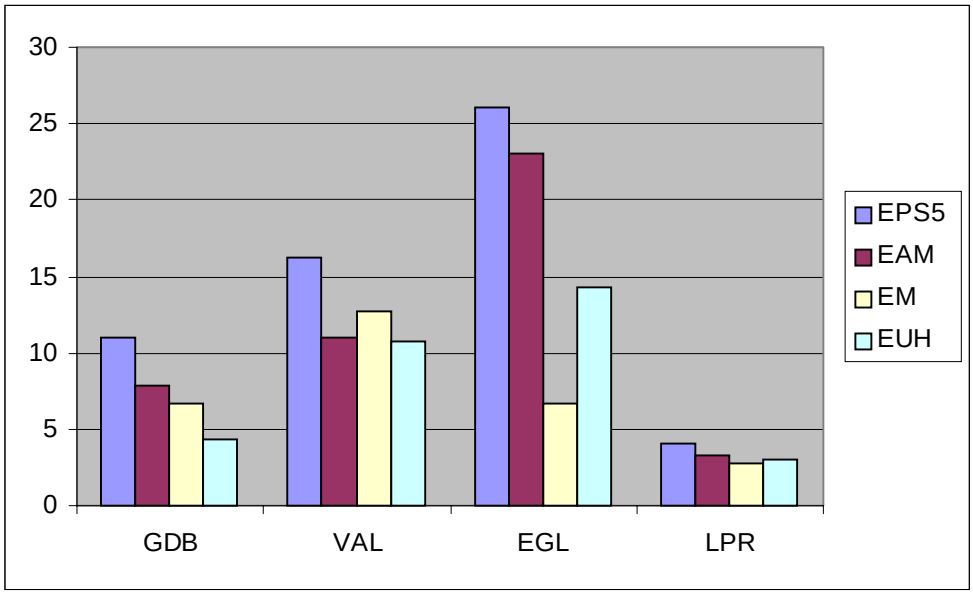


Figure 5.17 – *Comparaison des écarts des heuristiques sur les fichiers gdb, val, egl et lpr*

Globalement, pour les quatre ensembles de jeux d'essai, la table 5.6 montre qu'il existe toujours une heuristique dont le temps de calcul moyen est inférieur à 0.7 seconde et la déviation moyenne inférieure à 11 %. Sur les problèmes pas trop grands, on peut évidemment faire mieux avec l'algorithme génétique. Hélas, même le GA nécessite trop de temps pour converger quand les graphes sont très grands. Nos heuristiques ont été conçues justement pour de telles instances. La bonne surprise est leur très bon comportement sur nos grands ECARP qui imitent des réseaux réels, les fichiers *lpr* : ces graphes planaires et peu denses sont finalement plus faciles que certains *gdb* très denses, improbables dans les applications réelles.

5.5 Conclusion

Dans cette partie de notre recherche, nous avons défini une nouvelle extension du CARP, appelée ECARP, plus adaptée aux applications réelles. Nous avons proposé une formulation linéaire du ECARP dérivée d'un modèle linéaire de Golden et Wong (1981) et deux approches résolution : une méthode de coupes pour les problèmes de taille petite à moyenne taille et des heuristiques rapides pour les problèmes de grande taille.

L'algorithme génétique du chapitre 3 a pu être généralisé pour traiter le ECARP et servir de point de comparaison avec la méthode de coupes. Nous avons également conçu une

version pour un cas très réaliste d'ECARP bicritère, dans lequel des surcharges de véhicules sont tolérées. Des jeux d'essai de taille moyenne (*gdbe*) ont été construits en convertissant en ECARP des instances classiques (*gdb*) bien connues pour le CARP de base

Pour les grands problèmes, nous avons généralisé trois heuristiques du CARP de base. Cette généralisation n'est pas du tout triviale dans le cas d'Augment-Merge et de la méthode d'Ulusoy. Un générateur aléatoire sophistiqué a été développé pour créer des ECARP de grande taille imitant les réseaux réels. La comparaison des heuristiques a révélé des comportements surprenants, comme l'inutilité de la phase Augment dans Augment-Merge et un résultat très encourageant en vue des applications : les réseaux réels semblent plus faciles en moyenne que les instances classiques.

Nous envisageons et nous avons déjà commencé à étudier de nouvelles pistes de recherches pour continuer cette progression vers des modèles plus réalistes et des algorithmes toujours plus efficaces en tournées sur arcs. Nous les résumons ci-dessous :

- amélioration de l'algorithme de coupes, en cherchant à intégrer des inégalités valides et en utilisant d'autres formulations sous forme de programmes linéaires en nombres entiers.
- continuer la génération des fichiers LPR en les complétant avec des instances avec interdictions de tourner et/ou plusieurs lieux de vidages. Pour l'instant, les LPR ne contiennent que des CARP mixtes, pour rester utilisables par d'autres chercheurs.
- étude d'une approche de résolution exacte du CARP et du ECARP, combinant ainsi l'algorithme génétique et la méthode de coupes. Il s'agit d'un travail prévu dans le cadre d'une collaboration avec l'équipe de Belenguer et Benavent de l'Université de Valencia.
- application des heuristiques et du GA sur des données réelles de la ville de Limoges, en attente de la part d'Onyx, filiale propreté du groupe Vivendi.

Les travaux de ce chapitre ont donné lieu à des publications. Le modèle linéaire et la méthode de coupes ont été présentés à ESS 2001, la comparaison avec le GA étendu au ECARP et les versions bicritères ont été présentées à IFAC 2002 et font l'objet d'un article en cours de révision pour IIE Transactions. Les heuristiques rapides ont permis une communication à IFORS 2002.

Références

P. Lacomme, C. Prins, W. Ramdane-Chérif, An integer linear model for general arc routing problems, *ESS 2001* (*13th European Simulation Symposium*, Marseille, 18-20/10/01), pp. 427-432, SCS, 2001.

P. Lacomme, C. Prins, W. Ramdane-Chérif, Fast algorithms for general arc routing problems, *IFORS 2002*, 08-12/07/2002, Edimbourg, UK.

P. Lacomme, C. Prins, W. Ramdane-Chérif, General routing problems solved by a cutting plane algorithm and a genetic algorithm, *IFAC 2002* (21-26/07/02, Barcelone, Espagne), pp. 1195-1200.

P. Lacomme, C. Prins, W. Ramdane-Chérif, New algorithms for general arc routing problems, en révision pour *IIE Transactions*.

Nom des fichiers et paramètres de génération aléatoire										Caractéristiques des réseaux obtenus									
Nom	*nsqv*	*nsqh*	*clusters*	*p1dir*	*prside*	*pedge*	*depot*	*dumps*	*nd*	*n*	*m*	*e*	τ	*1-way*	*2-way*	*ST*	*SC*	*SQ*	*Trips*
LPR-A-01	2	3	0	0.00	0.60	0.00	central	depot	1	28	94	0	52	0	47	1418	34754	22225	3
LPR-A-02	4	5	1 dense	0.15	0.60	0.10	corner	depot	1	53	174	5	104	8	83	3121	77928	46400	5
LPR-A-03	6	8	2	0.10	0.62	0.10	central	depot	1	146	502	33	304	16	243	8114	229000	127805	13
LPR-A-04	6	9	1 light	0.10	0.75	0.10	corner	depot	1	195	685	34	503	29	328	10050	363000	214690	22
LPR-A-05	9	12	2	0.15	0.75	0.10	corner	depot	1	321	1114	58	806	72	521	17318	575576	336758	34
LPR-B-01	2	3	0	0.90	0.80	0.30	central	depot	1	28	68	5	50	26	21	1013	44693	24037	3
LPR-B-02	4	5	1 dense	0.90	0.85	0.20	central	depot	1	53	126	9	101	56	35	2310	80381	46123	5
LPR-B-03	6	9	2	0.85	0.80	0.20	central	depot	1	163	387	26	305	197	95	6122	215145	125595	13
LPR-B-04	7	10	1 light	0.85	0.85	0.10	corner	depot	1	248	590	8	501	340	125	8658	323634	204985	21
LPR-B-05	9	12	0	0.90	0.90	0.30	corner	depot	1	401	913	37	801	615	149	13071	556745	338656	34
LPR-C-01	2	3	0	0.10	0.90	0.90	corner	depot	1	28	91	39	50	3	44	1365	94053	32936	4
LPR-C-02	4	5	1 dense	0.10	0.90	0.90	corner	depot	1	53	178	77	100	4	87	3226	176992	62606	7
LPR-C-03	6	9	2	0.15	0.85	0.90	corner	depot	1	163	557	241	302	27	265	8803	554096	193476	20
LPR-C-04	8	10	1 light	0.15	0.75	0.85	central	depot	1	277	966	362	504	66	450	14219	816696	295479	30
LPR-C-05	9	12	1 light	0.25	0.90	0.70	central	depot	1	369	1228	387	803	156	536	18120	1121209	448598	45

groupe A : majorité de rue à 2 sens à côtés indépendants (villes modernes)
groupe B : majorité de rue à 1 sens (centres-villes anciens)
groupe C : majorité de rue à 2 sens avec collecte en tricot (zones résidentielles)

nd indique le nombre de dépôts

n nombre de nœuds
m nb d'arcs du codage
e nb d'arêtes
τ nb de tâches
1-*way* nb de rues à sens unique

2-*way* nb de rues à double sens
ST coût total de traversée
SC coût total de traitement
SQ demande totale
Trips nombre minimal de tournées

Table 5.7 – *Caractéristiques des fichiers lpr.*

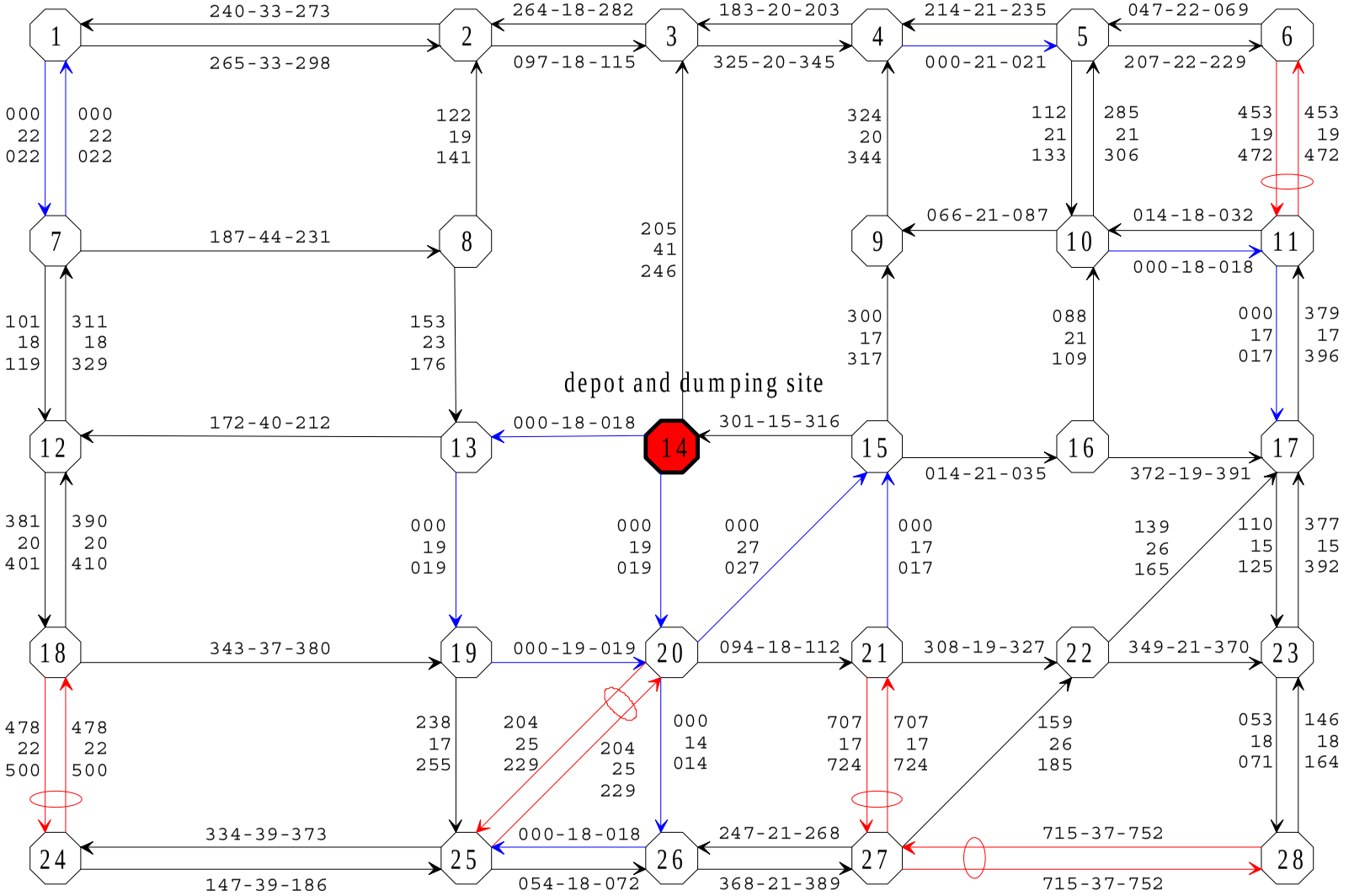


Figure 5.18 – *Exemple de lpr-b*-01 : n = 28, m = 68, τ = 50.
Valeurs sur les arcs : quantité (kg) – temps de traversée en HLP (s) – temps de collecte (s). Les ellipses indiquent les arêtes.

CHAPITRE 6
CARP périodique

6.1 Introduction

La gestion de production a popularisé une classification des problèmes selon l'horizon temporel considéré. Les décisions à long terme ou *stratégiques* impliquent l'entreprise sur un horizon d'un an ou plus, par exemple pour concevoir une nouvelle voiture ou construire une usine. Le calcul d'un plan de production valide pour quelques mois est typiquement une décision à moyen terme ou *tactique*. Le niveau *opérationnel* est représenté par exemple par l'ordonnancement d'atelier et considère un horizon limité à quelques journées.

La recherche en tournées sur arcs a traité essentiellement des problèmes opérationnels. Le CARP, le ECARP et le SCARP entrent dans ce cadre car ils peuvent être vus comme des problèmes de tournées sur un ensemble d'arcs à traiter pour un jour donné. En fait, en particulier en collecte des déchets ménagers, il faut résoudre des problèmes stratégiques et tactiques avant de pouvoir définir l'ensemble des tâches du jour. Ainsi, il faut déterminer l'emplacement du dépôt et des exutoires (les nœuds du réseau) avant de calculer des tournées.

Ces constatations et des discussions avec le principal industriel du secteur (Onyx) nous a conduit à étudier dans ce chapitre une nouvelle généralisation du CARP, appelée *PCARP* ou *Periodic CARP*, consistant à calculer des tournées sur un horizon de plusieurs jours. Nous définissons dans ce chapitre le PCARP, en partant d'un CARP de base et en ajoutant un horizon temporel ainsi que des fréquences de collectes pour les arcs. Nous montrons ensuite comment gérer des contraintes additionnelles comme des demandes et coûts dépendants du temps et des espacements à respecter entre traitements.

Nous proposons deux approches de résolution. La première est basée sur des heuristiques constructives et la deuxième est de type évolutionniste. L'approche évolutionniste comprend deux stratégies, une hiérarchique et une globale. La hiérarchique procède en deux phases : elle résout d'abord le problème d'affectation des tâches aux jours de l'horizon (niveau tactique), puis résout le CARP obtenu dans chaque période avec le GA du chapitre 3 (niveau opérationnel). Notre stratégie globale, très innovante, consiste à traiter simultanément les niveaux tactiques et opérationnels par un nouveau GA.

La suite du chapitre est organisée comme suit. Nous décrivons le processus de planification en collecte de déchets et définissons le PCARP. Puis nous présentons les heuristiques et le nouveau GA avant de finir par des évaluations numériques sur des jeux d'essai construits spécialement.

6.2 Travaux publiés sur les problèmes de tournées périodiques

Nous avons analysé la littérature existante sur les problèmes de tournées périodiques, en gardant à l'esprit les particularités des applications en tournées sur arcs. Cette analyse a révélé uniquement des problèmes sur nœuds dont le principal est le *PVRP* ou *Periodic Vehicle Routing Problem*. Le PVRP généralise le VRP. Il considère lui aussi un ensemble de clients à livrer avec une flotte de véhicules identiques. Mais, cette fois, chaque client nécessite un nombre donné de visites (appelé aussi *fréquence*) sur un horizon multipériode. L'objectif est de calculer un ensemble de tournées pour chaque période, de façon à satisfaire les fréquences et à minimiser le coût total des tournées sur l'horizon.

Dans les années 70, les problèmes de tournées périodiques commencent à intéresser aussi bien les chercheurs que le monde industriel, soucieux des économies pouvant être réalisées sur le coût total du transport. Les recherches, très souvent suscitées par des applications réelles, se sont développées à partir des années 80, notamment avec Christofides et Beasley (1984) et Gaudioso et Paletta (1992). La plupart des applications concernent des PVRP où il faut réapprovisionner automatiquement des clients, comme dans la distribution des gaz industriels ou le réapprovisionnement de distributeurs de boissons. Plus récemment, Witucki *et al*. (1997) ont écrit un état de l'art approfondi qui resitue les problèmes multipériodes dans la grande famille des problèmes de tournées. Ils présentent un exemple d'application dans la distribution de gaz liquéfié, pour lequel les auteurs ont intégré des techniques de recherche opérationnelle dans un SIAD (système interactif d'aide à la décision).

Le PVRP est extrêmement dur puisqu'il se réduit au VRP, qui est déjà NP-difficile, dans le cas d'une seule période. La première borne non triviale a été présentée oralement très récemment par Aristide Mingozzi, au congrès CO'2002 en avril 2002 à Paris, mais elle n'est pas encore publiée. Cette difficulté explique le fait que toutes les méthodes publiées soient des heuristiques. La meilleure est probablement une méthode tabou de Cordeau *et al*. (1997).

6.3 La planification de la collecte des déchets dans la réalité

La collecte des déchets ménagers est organisée selon un processus complexe passant par la résolution de problèmes stratégiques, tactiques puis opérationnels. Les problèmes stratégiques comme la localisation des dépôts et des exutoires sortent du cadre de cette thèse. Un exemple d'une étude de planification stratégique et tactique de la gestion des déchets d'un département français peut être trouvé dans Bostel *et al*. (2002). Nous évoquons donc ici uniquement les problèmes du niveau tactique.

Les entreprises de gestion des déchets connaissent la production moyenne quotidienne de déchets des ménages en fonction de la région, de la taille de la ville et du type d'habitat. Les centres-villes avec immeubles collectifs sont caractérisés par de faibles possibilités de stockage des déchets (containers en sous-sol par exemple) et nécessitent en général une collecte quotidienne. Les quartiers à maisons individuelles ont en général un garage ou jardin avec un container permettant de stocker des déchets pendant quelques jours. A cela peuvent s'ajouter des contraintes imposées par les municipalités, par exemple une volonté politique de collecter quotidiennement certaines zones pour des raisons d'image ou d'hygiène.

Ces données se traduisent par l'établissement d'une *fréquence de collecte* pour chaque rue, par exemple « tous les jours » ou « deux fois par semaine ». A partir de ces fréquences, la compagnie remportant l'appel d'offres essaie en général de minimiser en priorité le coût des investissements, c'est-à-dire principalement la taille de la flotte de véhicules à acheter.

Le nombre de véhicules est minimisé en lissant l'activité de collecte sur l'horizon, ce qui s'effectue en déterminant pour chaque rue un nombre de visites égal à sa fréquence. Le critère d'équilibrage est par exemple un tonnage à collecter variant très peu d'un jour à l'autre dans un secteur donné, ce qui favorise les recherches sur les heuristiques en deux phases : affectation des rues aux jours puis calcul des tournées pour les rues de chaque jour.

En fait, contrairement aux tournées sur nœuds en distribution, les jours de collecte d'une rue donnée doivent être les mêmes chaque semaine. Il y a à cela trois raisons principales. Premièrement, la production de déchets est relativement stable, ce qui autorise le calcul des tournées sur une semaine-type. Deuxièmement, les habitants oublieraient de sortir leurs containers sur le trottoir si les jours de passage changeaient chaque semaine. Troisièmement, les équipages mettent du temps pour bien connaître les rues d'un quartier et préfèrent donc des tournées stables d'une semaine à l'autre.

Cette rigidité apparente est en fait un avantage, car on va pouvoir planifier les tournées sur une longue période, typiquement un an. De plus, rien n'interdit de concevoir des algorithmes ne fonctionnant pas en deux phases, puisqu'il faut simultanément fournir à la ville les jours de collecte de chaque rue et la composition des tournées pour chaque jour.

6.4 Description et formalisation du PCARP

6.4.1 Version de base du PCARP et complexité

Le CARP de base est un problème non périodique défini sur un réseau non orienté. Nous présentons un nouveau problème théorique que nous nommons *PCARP* (*Periodic CARP*) ou Problème de Planification en Tournées sur Arcs, dont l'objectif est de planifier les tournées sur un horizon H de plusieurs périodes. Comme il serait trop complexe dans un premier temps de faire un ECARP ou un SCARP périodique, nous rendons multipériode le CARP de base, avec son réseau non orienté, mais avec les trois extensions suivantes. Ces contraintes sont en effet facilement gérées grâce à notre codage du réseau sous forme de graphe complètement orienté et l'utilisation de distances entre arcs (cf. chapitre 2) :

- interdiction de tourner et pénalités pour tourner,
- des coûts de collecte différents des coûts de traversée sans collecte,
- des coûts différents sur les deux sens d'une arête.

L'horizon H a np périodes ou jours et la flotte est homogène et limitée à nva véhicules. Cette limitation est naturelle car, sans elle, une solution optimale du PCARP tend à rassembler tous les traitements sur certaines périodes alors que d'autres sont complètement vides. Chaque arc requis u a une fréquence $f(u)$ définissant le nombre de traitements sur l'horizon, avec $1 \leq f(u) \leq np$ et, pour une tâche-arête, $f(u) = f(inv(u))$. Chaque arc requis est traité au plus une fois par période. Le nombre total de services ns à assurer sur H est donc, avec la condition $inv(u) < u$ pour ne compter qu'un arc requis pour chaque tâche-arête :

$$ns = \sum_{\substack{u \in R \\ inv(u) < u}} f(u) \tag{6.1}$$

Le PCARP dans sa forme de base consiste à affecter $f(u)$ jours de traitement dans H à chaque tâche u et à résoudre les CARP obtenus pour chaque jour, de façon à minimiser un coût total sur l'horizon. C'est clairement un problème du niveau décisionnel tactique.

Nous pouvons montrer que le PCARP est NP-difficile. En effet, il se réduit au CARP de base pour $np = 1$. Il reste difficile même avec des hypothèses très restrictives. Considérons par exemple une instance avec $np = 2$, $c(u) = w(u) = 0$ pour tout arc u, et $f(u) = 1$ pour tout arc requis et une demande totale de $2.Q$: le problème revient à un *problème de partition* des tâches en deux sous-ensembles de demande totale Q, connu pour être NP-difficile.

Le problème a des liens intéressants avec la gestion de production. L'affectation des jours aux tâches correspond au *problème de planification* en production, où on détermine l'ensemble des tâches à faire dans chaque période, tandis que la résolution fine du CARP défini par les tâches d'un jour donné correspond à *l'ordonnancement*, où il s'agit d'affecter les tâches aux processeurs (machines ou camions) et à les séquencer sur ces machines.

6.4.2 Contraintes additionnelles

L'examen des applications réelles révèle des complications comme des espacements entre collectes ou des variations de quantités selon les jours. Nous montrons dans cette section comment ajouter ces contraintes au modèle de base et proposons une petite classification des problèmes périodiques en tournées sur arcs, tels qu'ils sont rencontrés dans les applications.

6.4.2.1 Demandes et coûts dépendant du temps

En général, le coût de traitement $w(u)$ d'une rue u dépend de sa longueur et de sa demande $r(u)$. La longueur est souvent fournie par un *système d'information géographique* (SIG, en anglais GIS). On peut donc calculer le coût si la demande est connue. En collecte des déchets, les compagnies construisent ainsi des tables définissant le temps de collecte d'une rue en fonction de la longueur, de la quantité de déchets, mais aussi du nombre d'ouvriers manipulant les poubelles (rippeurs) et de la vitesse moyenne de déplacement du camion. Hélas, la demande $r(u,p)$ et le coût de traitement $w(u,p)$ de u peuvent dépendre de la période p dans le PCARP. Nous proposons alors la classification simple suivante.

a) PCARP de classe A

Cette classe concerne les problèmes les plus simples, où les demandes ne dépendent pas d'une "production" qui s'accumule avec le temps. C'est le cas du désherbage régulier des voies ferrées, du sablage de routes, de l'inspection de lignes électriques. Pour toute tâche, la demande et le coût de traitement ne changent pas d'une période à l'autre (formule 6.2).

$$\forall\, p \in \{1,\ldots, np\},\ \forall\, u \in R : r(u, p) = r(u) \text{ et } w(u, p) = w(u) \tag{6.2}$$

b) PCARP de classe B

Il s'agit des problèmes où une production journalière *prod*(*u*, *p*) s'accumule avec le temps. C'est le cas de la collecte des ordures ménagères et du fauchage des bas-côtés des routes en campagne. Dans le cas général, ces productions peuvent être définies pour chaque tâche par un vecteur de *np* entiers. Deux cas particuliers sont une production constante *prod*(*u*,*p*) = *prod*(*u*) ou une variation globale sur le réseau. Dans le second cas (lié au climat par exemple, comme la pousse du gazon déclenchant la tonte des pelouses), nous avons une production de référence *ref*(*u*) et une variation $\alpha(t)$: la production journalière est alors *prod*(*u*,*p*) = *ref*(*u*). $\alpha(t)$. Les problèmes B peuvent être *non cycliques* (B1) ou *cycliques* (B2).

Problèmes non cycliques (B1)

Dans le cas B1, l'horizon est linéaire et la demande *r*(*u*,*p*) de *u* en période *p* ne dépend que de la date du traitement précédent. Si les *f*(*u*) jours de traitement de *u* dans une solution *S* sont donnés en ordre croissant par une liste *days*(*S*,*u*), les demande sont calculables selon (6.3). Rappelons (*cf.* chapitre 2) que *prev* donne le prédécesseur d'un élément dans une liste :

$$\forall u \in R, \forall p \in days(S,u) : r(u,p) = \sum_{q=prev(days(S,u),p)+1}^{p} prod(u,q) \qquad (6.3)$$

Par convention, nous supposons dans cette formule que la production de la période *q* est disponible et peut être enlevée *en fin de période*. La formule est valable pour la première visite *p* = 1, en posant par convention une visite fictive en période 0. Ceci permet aussi de spécifier un « stock initial » *prod*(*u*,0). Le coût de traitement peut ensuite être calculé facilement à partir de la demande, grâce aux fonctions utilisées par les professionnels.

Un exemple de problème B1 est le fauchage des bas-côtés de routes en campagne. La végétation cesse de pousser en hiver et chaque nouvelle campagne de fauchage au printemps n'a pas à tenir compte de l'année précédente.

Problèmes cycliques (B2)

Dans le cas cyclique B2, l'horizon (semaine par exemple) se répète et l'affectation des jours aux tâches est identique sur toutes les semaines. C'est évidemment le cas de la collecte des déchets ménagers : la quantité à collecter dans une rue à la 1ère visite de la semaine dépend du temps écoulé depuis la dernière visite de la semaine précédente. Les demandes peuvent être calculées par la formule 6.3, sauf que si *p* est le jour du premier traitement (*p* = *first*(*days*(*S*,*u*))), alors *q* varie cycliquement de *last*(*days*(*S*,*u*)) + 1 à *np* puis de 1 à *p*.

La classe B2 pose des problèmes pour les approches séquentielles de type programmation dynamique, qui décomposent le problème dans le temps : on ne peut pas calculer les tournées du lundi tant que les jours de traitement des tâches dans la semaine ne sont pas complètement connus, car les quantités sont encore inconnues !

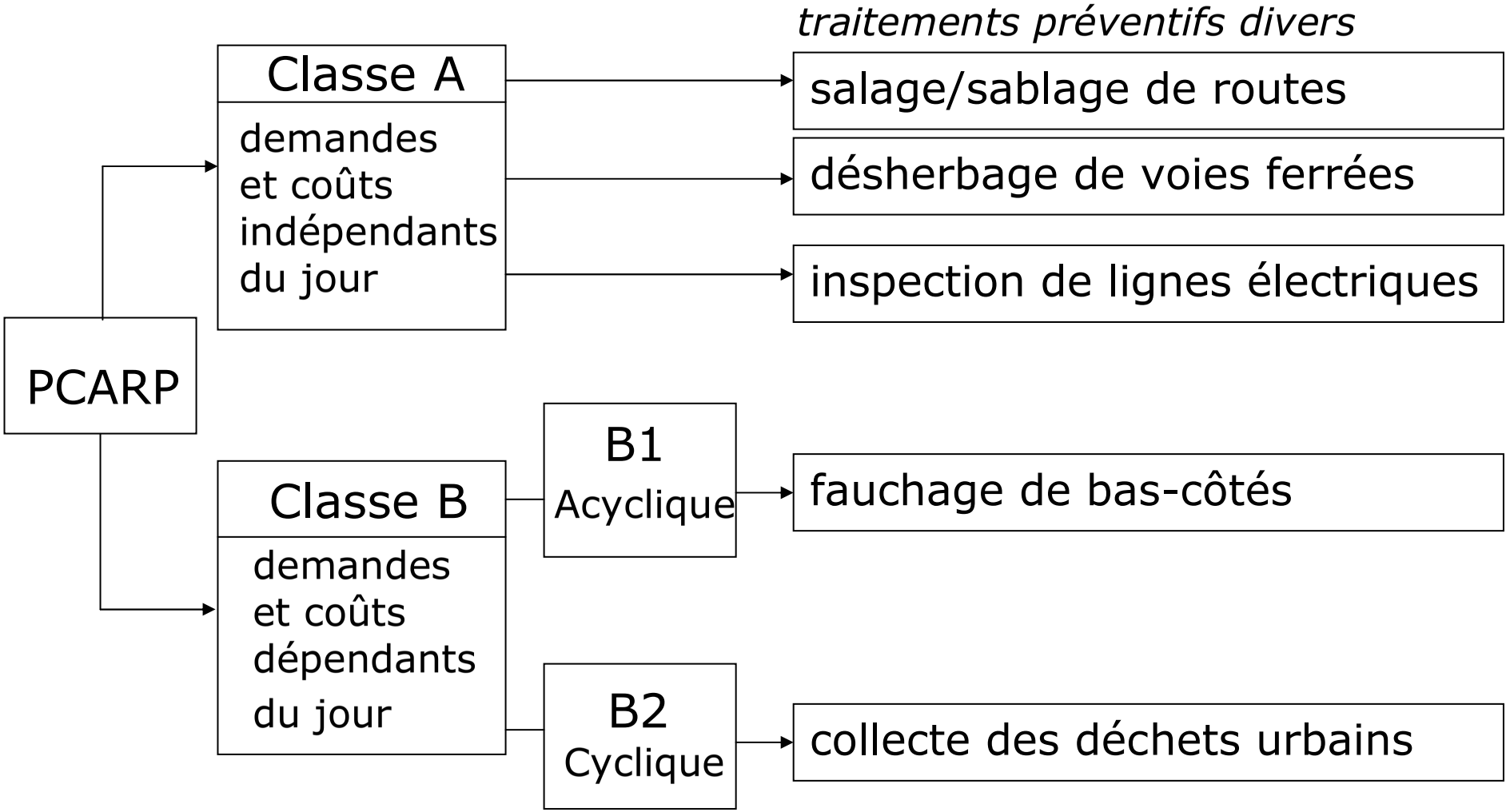


Figure 6.1 – *Une classification simple des problèmes périodiques en tournées sur arcs*

6.4.2.2 Espacements

Les solutions peuvent être inacceptables si des visites sont trop rapprochées. Nous supposons que des *espacements* à respecter sont spécifiés au niveau stratégique et fournis avec les données. C'est par exemple le cas en collecte des déchets, où les fréquences et les espacements dépendent du type d'habitat (immeubles ou pavillons) et des possibilités de stockage des déchets (container individuel ou collectif). Nous proposons deux modes de spécifier des espacements pour un arc requis u.

Le premier mode consiste à préciser un *espacement minimal* et un *espacement maximal* entre deux visites consécutives, notés $spmin(u)$ et $spmax(u)$ (*sp* comme *spacing*). Par convention, ce sont des différences entre dates de visites consécutives, et non des nombres de jours inactifs entre deux visites. Par exemple, si pour $f(u) = 2$ nous traitons un arc le lundi (période 1) et mercredi (période 3), l'espacement est de 2 jours. Dans les problèmes non cycliques, des espacements entre une visite et le début ou la fin de l'horizon peuvent être spécifiés en définissant des visites fictives aux jours virtuels 0 et $np + 1$.

Le second mode utilise des *combinaisons de jours* de visite. Une combinaison est une liste possible de jours de traitements pour traiter une rue, *triée en ordre croissant*. La liste des *ncomb* combinaisons rencontrées dans tout le réseau est notée *scomb* (*set of combinations*). Par exemple, *scomb* = ((1,4), (1,5), (1,2,3,4,5)) définit les combinaisons (lundi, jeudi), (mardi, vendredi) et "tous les jours". L'ensemble *scomb* peut être codé comme une matrice binaire $ncomb \times np$, avec $scomb(cb, p) = 1$ ssi la combinaison cb contient le jour p. Ce codage n'est pas le plus efficace pour un GA mais est bien pratique pour les programmes linéaires.

Dans le second mode, tout arc u a un sous-ensemble $comb(u)$ d'indices de combinaisons permises. Avec l'exemple précédent pour *scomb*, $comb(u) = \{1,2\}$ signifie que l'arc u est traitable au choix avec les combinaisons 1 et 2, c'est-à-dire lundi et jeudi ou mardi et vendredi. Nous excluons des combinaisons de fréquences différentes pour le même arc car, dans la réalité, la fréquence d'un arc est toujours fixée avant ses combinaisons de jours. *Nous supposons donc que toutes les combinaisons d'un arc ont la même fréquence.*

Nous avons retenu le second mode grâce à deux gros avantages :

- On peut toujours convertir des contraintes du premier mode en combinaisons de jour. Si np est petit comme en collecte des déchets, la transformation n'est pas coûteuse. A Troyes par exemple, $np = 7$ et les rues sont collectées tous les jours (centre ville) ou deux fois par semaine (banlieue) mais jamais les week-ends, ce qui donne 11 combinaisons possibles. Comme un espacement minimal de 3 jours est en plus imposé, seulement trois combinaisons sont en fait utilisées : (1,2,3,4,5), (1,4) et (2,5).

- Les demandes dépendant du temps dans les problèmes de la classe B peuvent être pré-calculées pour chaque tâche u, chaque combinaison de $comb(u)$ et chaque jour de cette combinaison : il suffit d'utiliser la formule 6.3 en remplaçant la liste de jours $days(S,u)$ par la combinaison considérée. Ce pré-traitement des données permet un traitement uniforme des problèmes des classes A et B

6.4.2.3 Nombre de véhicules

Le nombre *nva* de véhicules disponibles (taille de la flotte) peut être imposé si la flotte est déjà achetée ou être une variable de décision si elle n'est pas encore achetée. Dans une solution *S*, l'affectation des combinaisons de jours aux tâches est valide si elle respecte la capacité offerte par la flotte (condition 6.4). Notez que le respect de cette condition est déjà un problème NP-complet, puisqu'il équivaut à un problème de bin-packing pour *np* = 1.

$$\forall p \in \{1,...,np\}: \qquad \sum_{\substack{u \in R \\ p \in days(S,u)}} r(u,p) \leq nva \times Q \tag{6.4}$$

En simulation, pour répondre à un appel d'offres émis par une municipalité, une société comme Onyx va essayer d'investir dans des moyens minimaux (taille de flotte), puis de réaliser les tournées périodiques à coût minimal avec cette flotte. Nous avons donc décidé de *relaxer* la contrainte sur le nombre de véhicules et de l'incorporer dans la fonction objectif. Soit *M* une borne supérieure du coût total *cost* des tournées sur l'horizon et *nvu* le nombre de véhicules réellement utilisés (maximum des nombres de tournées de chaque période).

$$F_1(nvu, cost) = M.nvu + cost \tag{6.5}$$

$$F_2(nvu, cost) = M.\max(0, nvu\text{-}nva) + cost \tag{6.6}$$

L'objectif est de minimiser une fonction hiérarchique bicritère F_1 (6.5) si la flotte n'est pas imposée ou F_2 (6.6) si la flotte est fixée à *nva* véhicules. En minimisant F_1, on minimise hiérarchiquement la taille de la flotte nécessaire puis le coût total des tournées sur l'horizon. Dans F_2, peu importe que les *nva* véhicules disponibles ne soient pas tous utilisés. Par contre, surtout au début d'algorithmes itératifs comme les GA, cette fonction permet de tolérer, en les pénalisant, des solutions utilisant trop de véhicules.

6.5 Approches de résolution

Les travaux sur le PVRP résolvent en général deux sous-problèmes : un problème tactique, où il faut décider quelles tâches sont visitées pour chaque jour de l'horizon, puis un VRP pour les tâches affectées à chaque jour. On pourrait faire de même pour le PCARP et résoudre un CARP par période. Par analogie avec la gestion de production, rappelons que l'affectation aux jours correspond au *problème de planification* proprement dit, tandis que la résolution fine du CARP obtenu pour un jour donné correspond à *l'ordonnancement*, qui relève du niveau de décision dit *opérationnel*.

En gestion de production, une technique *d'horizon glissant* (*rolling horizon*) est souvent utilisée. Elle calcule un ordonnancement seulement pour les premières périodes du plan de production. En effet, il n'y a aucune périodicité et les quantités sont incertaines dans l'avenir : des nouvelles commandes vont encore arriver. En tournées sur arcs, en tout cas en collecte de déchets, il faut fixer les jours de visite sur tout l'horizon, puisqu'ils doivent être annoncés par avance à la population et rester stables pendant au moins un an.

On peut donc tenter de résoudre *simultanément* les niveaux planification et ordonnancement puisque les problèmes d'ordonnancement sont définis sur chaque jour de l'horizon, après affectation des tâches aux jours. Comme nous avons déjà un excellent algorithme génétique pour le CARP mono-période, la généralisation de ce GA à un horizon multipériode nous a paru prometteuse.

Dans ce qui suit, nous présentons deux méthodes de résolution du PCARP basées sur des algorithmes génétiques. La première est un GA multipériode appelé *PGA* ou *Periodic GA*, car il est capable de combiner des décisions tactiques (affectation d'une combinaison de jours à une tâche) et opérationnelles (construction des tournées d'un jour donné). Ce PGA se réduit au GA du chapitre 3 si $np = 1$. Une variante appelée *IPGA* ou *Improved PGA* consiste à effectuer une post-optimisation dans chaque période sur la meilleure solution du *PGA*.

L'autre méthode est un algorithme en deux phases qui appelle en sous-programme le GA monopériode du chapitre 3 : une heuristique affecte d'abord une combinaison de jours à chaque tâche, ce qui donne un CARP pour chaque jour, puis notre GA monopériode est appelé pour résoudre ces *np* CARP un par un.

Avant de présenter ces approches évolutionnistes, nous proposons deux heuristiques constructives qui peuvent être réutilisées dans la population initiale du PGA. Une solution S est désormais un tableau de listes de tournées, $S(p)$ désignant la liste des tournées construites pour le jour p, c'est-à-dire une solution du CARP défini par les tâches à traiter le jour p. Rappelons qu'avec nos notations du chapitre 2, $last(S(p))$ est la dernière tournée du jour p.

6.5.1 Heuristiques initiales

Nous avons développé deux heuristiques d'insertion qui partent d'une liste L de tâches triée par fréquences décroissantes et d'une solution initiale S contenant une tournée vide dans chaque période. Les deux heuristiques traitent chaque tâche $u = L(1), L(2), \ldots, L(\tau)$ et insèrent u de façon à minimiser la variation de la fonction-objectif choisie (F_1 ou F_2 au choix).

Par rapport à une méthode d'insertion classique, les heuristiques doivent insérer $f(u)$ copies de u dans $f(u)$ jours distincts, pour respecter la fréquence de la tâche. De plus, ces $f(u)$ jours doivent être permis, c'est-à-dire correspondre à une combinaison de $comb(u)$. En pratique, on teste donc chaque combinaison cb de $comb(u)$ et, pour chaque jour p de cb, on insère u dans une tournée de $S(p)$. Si aucune insertion n'est possible, les deux algorithmes concatènent à la fin de $S(p)$ une nouvelle tournée réduite à la tâche u.

La seule différence entre les deux heuristiques est le critère d'insertion. Pour chaque jour p de cb, la première heuristique nommée *LIH* ou *Last Insertion Heuristic* considère une seule position d'insertion, à la fin de $last(S(p))$ (la dernière tournée du jour p). La seconde heuristique, nommée *BIH* ou *Best Insertion Heuristic*, détermine la meilleure position d'insertion dans toutes les tournées de $S(p)$. Pour les deux heuristiques et une combinaison cb, la variation globale de la fonction-objectif se calcule facilement car les $f(u)$ copies de la tâche u sont insérées dans des jours distincts, par définition d'une combinaison : il n'y a donc pas d'interactions entre deux insertions.

6.5.2 Résolution globale du PCARP avec un GA périodique (PGA)

Nous développons ici un GA dit périodique (PGA), capable de combiner des solutions en modifiant simultanément des décisions tactiques (affectation des combinaisons de jours aux tâches) et des décisions opérationnelles (composition des tournées dans chaque période). Il réutilise certains composants du GA développé pour le CARP dans le chapitre 3, comme la procédure de découpage *Split* et la procédure de recherche locale. En fait, le PGA se comporte exactement comme le GA dans le cas où $np = 1$. Ci-dessous, nous décrivons les composants que nous avons étendus de façon non triviale. La structure générale du PGA reste par ailleurs la même que celle du GA du chapitre 3.

6.5.2.1 Chromosomes

Un chromosome code une solution du PCARP sous forme d'une longue séquence Λ d'arcs requis. Λ contient *ns* éléments (nombre total de services des tâches sur l'horizon) et est formée de *np* sous-listes successives (une par jour), séparées par des délimiteurs de jour. Chaque tâche *u* apparaît $f(u)$ fois dans Λ (éventuellement sous la forme $inv(u)$ pour une tâche-arête) et au plus une fois dans la sous-liste d'un jour donné. Les $f(u)$ jours où *u* apparaît doivent former une combinaison autorisée, c'est-à-dire contenue dans $comb(u)$. La figure 6.1 commentée plus loin avec le croisement donne deux exemples.

Un chromosome peut être interprété physiquement comme un ordre de priorité pour un véhicule traitant toutes les tâches sur l'horizon, jour après jour. Notons qu'une sous-liste pour un jour donnée correspond à un chromosome du chapitre 3 pour le CARP, sauf que des tâches peuvent être absentes certains jours. Comme dans notre meilleur GA du chapitre 3, il n'y a pas de délimiteurs de tournées dans les sous-listes. La procédure de découpage *Split* est utilisée pour découper optimalement en tournées chaque sous-liste et calculer le coût total du chromosome, comme expliqué en 6.5.2.2.

Un chromosome Λ peut être généré à partir d'une solution heuristique du PCARP (en concaténant ses tournées sur tous les jours de l'horizon), être généré aléatoirement comme expliqué en 6.5.2.3 ou résulter d'un croisement décrit en 6.5.2.4.

Notre codage des chromosomes est simple mais pertinent car *il existe au moins une séquence optimale*. En effet, soit une solution optimale du PCARP. Concaténons les tournées de chaque jour pour avoir une sous-liste par jour puis concaténons les sous-listes correspondant aux jours successifs 1 à *np*, en insérant un délimiteur entre les jours. On obtient un chromosome à partir duquel la procédure d'évaluation va retrouver la solution optimale.

6.5.2.2 Evaluation du chromosome

L'idée est de réutiliser la procédure *Split* décrite en 3.5 pour évaluer la sous-liste de tâches de chaque jour. Rappelons que cette procédure indique où découper la sous-séquence en tournées, avec un coût minimal. Pour le PCARP, elle peut être implémenté comme une procédure *Split* (Λ,*p*,*nt*,*ct*) évaluant dans Λ la sous-séquence pour un jour *p* et retournant le nombre de tournées *nt* pour ce jour et le coût total *ct*. Ainsi, la fonction-objectif bicritère F_1 peut être calculée par la procédure *EvaluateF1*(Λ,*cost*,*nvu*) de l'algorithme 6.1.

```
nvu  := 0
cost := 0
for p := 1 to np do
   split (Λ,p,nt,ct)
   nvu := max(nvu,nt)
   cost := cost + ct
endfor
F1 := M.nvu + cost
```

Algorithme 6.1 – *Procédure EvaluateF1*(Λ,*cost*,*nvu*) *pour un chromosome* Λ *du PCARP*.

En fait, la procédure *Split* a besoin des demandes associées à chaque occurrence d'une tâche donnée pour savoir si elle peut découper une liste de tâches en respectant la capacité des véhicules. Comme expliqué en 6.4.2.2, le système des combinaisons de jours permet de pré-calculer, pour chaque tache *u* et chaque combinaison de *comb*(*u*), la demande pour chaque jour de la combinaison. *Split* peut par exemple appeler une fonction *amount*(Λ,*q*) qui lui renvoie la demande pour la tâche de rang *q* dans le chromosome Λ.

Le calcul de la fonction-objectif F_2 conçue pour une flotte limitée de *nva* véhicules est similaire. Dans ce cas, la demande totale dans chaque tournée peut excéder *nva*.*Q* puisque les véhicules en excès sont tolérés mais pénalisés dans l'expression de F_2, cf. 6.4.2.3. C'est le mécanisme de sélection du GA qui va éliminer des chromosomes de ce genre.

6.5.2.3 Population initiale

La population est un tableau de *nc* chromosomes de coûts distincts. Elle contient les solutions de nos deux heuristiques *BIH* et *LIH* du 6.5.1, complétées par des chromosomes aléatoires. L'algorithme 6.2 génère aléatoirement un chromosome réalisable en exploitant le fait que les combinaisons de jours permises pour une tâche respectent par définition les fréquences et les espacements. Le chromosome est ici évalué pour F_1.

```
préparer np listes vides L(1),L(2),…,L(np)
for  count := 1 to τ do
   tirer au sort un arc requis u non encore traité
   tirer au sort un index de combinaison q dans comb(u)
   for all p ∈ scomb(q) do
      enqueue(L(p),u)
   endfor
   marquer u et inv(u) comme arc traité
endfor
concaténer L(1),L(2),…,L(np) dans Λ
evaluateF1(Λ,cost,nvu)
```

Algorithme 6.2 – *Génération aléatoire d'un chromosome* Λ *pour le PCARP*

6.5.2.4 Croisement

Le GA mono-période du chapitre 3 utilise un croisement OX modifié. Nous avons vu que ce croisement conçu à l'origine pour des permutations cycliques donne de meilleurs résultats que LOX car il n'y a aucune raison pour distinguer une « première » ou une « dernière » tournée dans une solution du CARP.

Pour le PCARP, le croisement LOX conçu pour des chromosomes linéaires semble plus adapté car l'horizon temporel considéré a clairement un premier et un dernier jour. Rappelons que pour deux parents P1 et P2, LOX (algorithme 3.9 et figure 3.6) commence par choisir aléatoirement une sous-séquence dans P1 pour la copier aux mêmes positions dans l'enfant C1. Puis P2 est scanné de la gauche vers la droite et les éléments non encore présents dans C1 sont copiés pour compléter de gauche à droite les positions vides de C1.

Notre croisement pour le PCARP, nommé *ELOX* (*Extended LOX*), est une extension de LOX à des chromosomes multipériodes et dans lesquels les tâches apparaissent plus d'une fois. Comme dans la génération aléatoire, le fils C1 est préparé comme une table de *np* sous-listes vides C1(1), C1(2),…, C1(*np*), soit une sous-liste par jour. L'algorithme de LOX utilise aussi une variable temporaire *numb*(*u*) (nombre d'occurrences de *u* déjà copiées dans C1), une variable temporaire *done*(*u*) (ensemble des jours de traitement de *u* déjà choisis dans C1), ainsi qu'une fonction *day*(*S*,*k*) retournant le jour de la tâche de rang *k* dans un chromosome *S*.

ELOX commence par tirer au sort deux positions *a* et *b* dans P1, avec $1 \le a \le b \le ns$. Les tâches de P1(*a*) à P1(*b*) sont copiées dans C1 en gardant leurs jours de service. P2 est ensuite scanné pour compléter C1. Contrairement à LOX, les *f*(*u*) occurrences de chaque tâche *u* doivent être intégrées dans C1 pour satisfaire les contraintes de fréquences. Pour cela, ELOX copie la tâche courante *u* = P2(*q*) si et seulement si sa fréquence n'est pas encore satisfaite, c'est-à-dire si $numb(u) \le f(u)$.

Si c'est le cas, on tente d'abord de garder le même jour de service *day*(P2,*q*) dans C1, ce qui est impossible si C1 contient déjà une copie de *u* dans ce jour ou s'il n'existe aucune combinaison de jours *cb* contenant à la fois les jours déjà choisis pour *u* et *day*(P2, *q*), c'est-à-dire une combinaison $cb \in comb(u)$, telle que $done(u) \cup \{day(P2,q)\} \subseteq scomb(cb)$. Si on peut pas garder le jour *day*(P2,*q*), ELOX utilise le premier jour possible *p* qui ne contient pas *u* et tel que $done(u) \cup \{p\}$ soit inclus dans au moins une combinaison permise pour *u*.

L'algorithme de ELOX est présenté dans l'algorithme 6.3 avec un exemple dans la figure 6.2. Le traitement est décrit pour le premier enfant C1. Le second C2 peut être calculé en échangeant les rôles des deux parents. Soit *comb*(*u*,*q*) le $q^{ème}$ jour de traitement dans *comb*(*u*). L'algorithme utilise une fonction *dayOK*(*p*, *u*) qui retourne *true* si et seulement si *u* peut être copiée dans le jour *p*, c'est-à-dire :

- $p \notin done(u)$
- $\exists\, q \in \{1\ldots|comb(u)|\}$ tel que $done(u) \cup \{p\} \subseteq comb(u,q)$.

Le croisement réalise effectivement un héritage à partir des parents P1 et P2 : la portion de P1(*a*)-P1(*b*) est intégralement héritée de P1 (séquence et jours), et on s'efforce de respecter autant que possible l'ordre de P2. Il est facile de prouver que C1 est un enfant faisable si P1 et P2 sont faisables et que ELOX se réduit à LOX dans le cas d'une seule période : *np* = 1 et *f*(*u*) = 1 pour tout *u*.

N'oublions pas que l'enfant doit ensuite être évalué par l'algorithme 6.1, qui va découper optimalement la liste de tâches de chaque jour en tournées grâce à *Split*. Le nombre de tournées (et donc de véhicules) peut être très variable d'un jour à l'autre et, dans le cas d'une flotte limitée, utiliser trop de véhicules. Mais, grâce aux fonctions-objectifs F_1 et F_2 qui pénalisent ces phénomènes, le GA va progressivement les éliminer.

```
// 1. Initialisations
préparer np listes vides C1(1),C1(2),…,C1(np)
for each u in R do numb(u):=0; done(u)=∅ endfor
tirer aléatoirement deux positions a et b dans P1, a ≤ b
// 2. Copie P1(a) à P1(b) dans C1 (séquence et jours)
for q := a to b do
   u := P1(q)                                  //extrait l'arc requis courant
   p := day(P1,q)                              //récupère son jour
   enqueue (C1(p),u)                           //copie u à la fin de C1(p)
   incrémenter numb(u) et numb(inv(u))         //compte 1 jour pour la tâche
   ajouter p à done(u) et done(inv(u))         //ajuste ensembles des jours
endfor
// 3. Balaie P2 pour completer C1
for q := 1 to ns do
   u := P2(q)                                  //extrait l'arc requis courant
   p := day(P2,q)                              //récupère son jour
   if numb(u) < f(u) then                      //si fréquence non satisfaite
      if not dayOK(p,u) then                   //le jour doit être changé
         p := premier jour avec dayOK(p,u)     //calcule nouveau jour p
      endif
      enqueue (C1(p),u)                        //copie u à la fin de C1(p)
      incrementer numb(u) et numb(inv(u)       //compte 1 jour pour la tâche
      ajouter p à done(u) et done(inv(u))      //ajuste ensemble des jours
    endif
endfor
```

Algorithme 6.3 - *Croisement ELOX pour le PCARP*

u	***f*(*u*)**	***comb*(*u*)**
1	3	{L, Me, J}, {Ma, Me, J}
2	2	{L, Me}, {Ma, J}
3	1	{L}, {J}
4	1	{L}
5	1	{L}, {Ma}
6	3	{Ma, Me, J}, { L, Ma, J}, { L, Ma, Me}
7	2	{Ma, J}, {L, Me}
8	1	{Ma}, {J}

P1

Lundi				Mardi			Mercredi			Jeudi			
1	2	3	4	**5**	**6**	**7**	**1**	2	6	1	6	7	8

P2

Lundi				Mardi				Mercredi		Jeudi			
7	6	5	4	8	6	2	1	7	1	3	6	1	2

C1

Lundi		Mardi						Mercredi		Jeudi			
6	4	**5**	**6**	**7**	8	2	1	**1**	6	7	3	1	2

Figure 6.2 - *Exemple de croisement ELOX (les tâches copiées de P1 sont en gras souligné).*

6.5.3 Approche en deux phases

On procède comme dans la gestion de production en deux phases. Dans la première phase, une combinaison de jours est affectée à chaque tâche (niveau de décision tactique) de façon à minimiser la taille de la flotte. Dans la seconde, les *np* CARP résultant de l'affectation sont résolus en appelant le nouveau PGA avec *np* = 1 (niveau de décision opérationnel). Il reste donc à préciser la procédure d'affectation pour spécifier complètement la méthode en deux phases. Nous proposons dans cette section deux formulations linéaires pour ce problème, puis nous expliquons comment on peut aussi réutiliser les heuristiques ou le PGA.

6.5.3.1 Programmes linéaires simples pour la phase d'affectation

Si la flotte est limitée à *nva* véhicules, une condition nécessaire d'existence très faible d'une affectation réalisable est une somme des charges sur l'horizon ne dépassant pas *np*.*nva*.*Q*. Une condition nécessaire faible est que la somme des charges de chaque jour n'excède pas *nva*.*Q*. Enfin, une condition nécessaire et suffisante est que l'ensemble des tâches affectées à chaque jour soit effectivement partitionnable en *nva* sous-ensembles de charge inférieure ou égale à *Q*.

Nous proposons deux programmes linéaires en nombre entiers pour calculer une affectation consommant un nombre minimal de véhicules. Ils ne sont valables que dans le cas de demandes indépendantes de la période (PCARP de classe A). Le premier vérifie seulement la condition faible mais est de petite taille. Le second vérifie la condition nécessaire et suffisante mais est beaucoup plus gros.

Le premier programme minimise le nombre de véhicules utilisés *nvu*, tel que la charge totale sur chaque jour soit inférieure ou égale à *nvu*.*Q*. Si la flotte de véhicules est limitée à *nva*, il faudra constater *a posteriori* si *nvu* est inférieur, égal ou supérieur à *nva* et, en cas d'excès, à décider s'il faut par exemple louer les véhicules manquants.

Soit x_{uq} une variable binaire valant 1, si seulement si l'arc *u* utilise la combinaison de jours *q* et v_{up} une variable binaire valant 1 ssi *u* est visitée le jour *p*. L'ensemble *scomb* de toutes les combinaisons utilisées dans le problème est codé comme une matrice binaire *ncomb*×*np*, avec *scomb*(*q*, *p*) = 1 ssi la combinaison *q* contient le jour *p*.

Minimiser *nvu* (6.7)

$$\forall u \in R, inv(u) < u : \sum_{q \in comb(u)} x_{uq} = 1 \quad (6.8)$$

$$\forall u \in R, inv(u) < u, \forall p \in \{1,...,np\}: v_{up} = \sum_{q \in comb(u)} scomb(q, p) \times x_{uq} \quad (6.9)$$

$$\forall p \in \{1,...,np\}: \sum_{\substack{u \in R \\ inv(u) < u}} r_u \times v_{up} \leq nvu \times Q \quad (6.10)$$

$$\forall u \in R, inv(u) < u, \forall q \in comb(u) : x_{uq} \in \{0,1\} \quad (6.11)$$

$$\forall u \in R, inv(u) < u, \forall p \in \{1,...,np\}: v_{up} \in \{0,1\} \quad (6.12)$$

$nvu \in$ IN (6.13)

La fonction-objectif (6.7) consiste à minimiser le nombre de véhicules réellement utilisés. Les contraintes (6.8) garantissent qu'une seule combinaison de jours est utilisée pour chaque arc. Les contraintes (6.9) positionnent les variables binaires des jours de visite en fonction de la combinaison choisie. Les contraintes (6.10) bornent *nvu* pour chaque jour. On aurait pu mettre une contrainte $nvu \leq nva$, mais le modèle pourrait être infaisable. Notre version va toujours trouver une solution : le décideur devra trancher si $nvu > nva$.

Le problème d'affectation modélisé par ce premier programme est *NP-difficile*, car il se réduit à un *problème d'ordonnancement sur machines parallèles* si toutes les fréquences sont égales à 1 et si chaque tâche peut être placée dans n'importe quel jour : il faut alors placer les tâches sur les *np* machines (les jours), pour minimiser un makespan multiple de *Q*.

Le programme précédent ne garantit pas que la charge affectée à chaque jour soit découpable en au plus *nva* véhicules de charge au plus *Q*. Cet inconvénient est levé par le second modèle. On ajoute des variables y_{upk} valant 1 si la tâche *u* est visitée le jour *p* par le véhicule *k*, et des variables binaires z_{qp} valant 1 si le véhicule *q* est utilisé le jour *p*. Nous ne commentons pas les contraintes, qui ne posent pas de difficulté.

Minimiser *nvu* (6.14)

$$\forall u \in R, inv(u) < u : \sum_{q \in comb(u)} x_{uq} = 1 \quad (6.15)$$

$$\forall u \in R, inv(u) < u, \forall p \in \{1,...,np\} : v_{uq} = \sum_{q \in comb(u)} scomb(q,p) \times x_{uq} \quad (6.16)$$

$$\forall p \in \{1,...,np\}, \forall u \in R, inv(u) < u : \sum_{k=1}^{nva} y_{upk} = v_{up} \quad (6.17)$$

$$\forall p \in 1..np, \forall k \in \{1,...,nva\} : \sum_{\substack{u \in R \\ inv(u) < u}} r_u \times y_{upk} \leq Q \times z_{kp} \quad (6.18)$$

$$\forall p \in \{1,..,np\} : \sum_{q=1}^{nva} z_{qp} \leq nvu \quad (6.19)$$

$$\forall u \in R, inv(u) < u, \forall q \in comb(u) : x_{uq} \in \{0,1\} \quad (6.20)$$

$$\forall u \in R, inv(u) < u, \forall p \in \{1,...,np\} : v_{uq} \in \{0,1\} \quad (6.21)$$

$nvu \in$ IN (6.22)

$$\forall u \in R, inv(u) < u, \forall p \in \{1,...,np\}, \forall k \in \{1,...,nva\} : y_{upk} \in \{0,1\} \quad (6.23)$$

$$\forall k \in \{1,...,nva\}, \forall p \in \{1,...,np\} : z_{kp} \in \{0,1\} \quad (6.24)$$

On a une sorte de bin-packing emboîté, par jour puis par véhicule. Ce problème est évidemment NP-difficile. Si ce deuxième modèle trouve une solution, on peut résoudre ensuite le CARP de chaque jour par n'importe quelle heuristique (Path-Scanning, Augment-Merge, etc.). On obtient ainsi une bonne heuristique pour le PCARP.

6.5.3.2 Réutilisation de LIH, BIH ou PGA pour l'affectation par BIH et PGA

Cette méthode consiste d'abord à exécuter l'heuristique BIH du 6.5.1 ou le GA périodique du 6.5.2 pour calculer une bonne solution du PCARP. On ignore ensuite les tournées obtenues pour conserver seulement les combinaisons affectées aux tâches. Finalement, on exécute le PGA en mode mono-période (ou le GA du chapitre 3) pour résoudre les CARP définis par l'ensemble des tâches de chaque jour.

Ce genre de méthode peut être vue comme une phase de post-optimisation avec un algorithme génétique, appliquée jour par jour à une solution du PCARP obtenue par une heuristique qui peut en fait être quelconque. Nous appelons *DGA* (*Daily GA*), la méthode qui utilise l'affectation calculée par BIH et *IPGA* (*Improved PGA*) celle qui utilise l'affectation par PGA. Cette dernière méthode peut être vue comme une double application de notre GA périodique, d'abord sur l'ensemble des périodes, puis sur chaque jour, ce qui constitue une sorte d'intensification.

6.6 Evaluation numérique

6.6.1 Implémentation et jeux de tests

Les algorithmes ont été programmés en Delphi 5 et exécutés sur un PC sous Windows 98 avec un Pentium-III à 1 GHz. Pour l'évaluation numérique, nous avons adapté les 23 problèmes de DeArmon (1981), qui possède de 7 à 27 nœuds et de 11 à 55 arêtes, pour construire des fichiers pour le PCARP appelés *gdbf*.

Pour imiter des situations réalistes comme à Troyes, nous avons ajouté un horizon cyclique d'une semaine (7 jours) mais avec 5 jours de travaille. Les demandes initiales des arcs dans les fichiers gdb sont interprétées maintenant comme des productions journalières constantes sur les 7 jours. Nous avons défini un ensemble de combinaisons par fréquence valables pour toutes les tâches, *cf.* table 6.1. Pour chaque arête u, on tire aléatoirement une fréquence $f(u)$ entre 1 et 5.

Les coûts de traversées $c(u)$ ne changent pas. Pour chaque tâche u et chaque jour p de chaque combinaison, on peut calculer $r(u, p)$ par la formule 6.3 sur l'horizon cyclique de 7 jours. Le coût de service est calculé par la formule $w(u,p) = w(u) \times sp(p)$, avec $sp(p)$ le délai entre la visite précédant p et p. Les instances résultantes ont une taille allant de ns = 33 à 132 services sur l'horizon, avec une moyenne de 70.

fréquence	*nombre de combinaisons*	*liste des combinaisons*
1	5	{1}, {2}, {3}, {4}, {5}
2	6	{1,3}, {1,4}, {1,5}, {2,4}, {2,5}, {3,5}
3	7	{1,2,4}, {1,2,5}, {1,3,4}, {1,3,5}, {1,4,5}, {2,3,5},{2,4,5}
4	5	{1,2,3,4}, {1,2,3,5}, {1,3,4,5}, {2, 3 ,4,5}, {2,4,5,1}
5	1	{1,2,3,4,5}

Table 6.1- *Définition des combinaisons*.

6.6.2 Résultats des évaluations numériques

Nous avons étudié la minimisation de la fonction-objectif F_1. Nous avons comparé les heuristiques LIH et BIH et les approches évolutionnistes PGA, DGA et IPGA. Nous donnons les résultats dans la table 6.3 et les détails par période pour PGA, DGA et IPGA respectivement dans les tables 6.4, 6.5 et 6.6. Ces quatre tables sont présentées en fin du chapitre. Une comparaison globale des performances est donnée dans la table 6.2.

La table 6.3 contient les colonnes suivantes. La colonne *file* donne le nom du problème. Celles libellées *n* et *ns* indiquent respectivement le nombre de nœuds et le nombre total de tâches à traiter sur l'horizon. Les colonnes *nvu*, *cost*, *time* donnent respectivement le nombre de véhicules utilisés, le coût total sur l'horizon et le temps de calcul en minutes. La ligne *average* indique les moyennes en terme de nombre de véhicules, de coût total et de temps de calcul. En gras, les meilleurs résultats en terme de coût total sur l'horizon et, sur fond gris, les meilleures solutions pour la minimisation de F_1.

En ce qui concerne les heuristiques, notons que LIH1 est une version rustique de LIH qui utilise en entrée une liste de tâches ordonnée aléatoirement tandis que LIH2 utilise une liste triée par ordre décroissant des fréquences. On note que LIH2 est meilleure que LIH1, avec en moyenne 7.35 véhicules au lieu de 8.00. Cela s'explique, parce qu'une tâche *u* de grande fréquence est difficile à placer en dernier : il est peu probable qu'il reste assez de capacité dans tous les $f(u)$ jours où il faut insérer une copie de la tâche. BIH est encore meilleure que LIH2, avec 7.00 véhicules en moyenne. L'avantage de ces heuristiques est leur rapidité : les temps de calcul, non mentionnés pour ne pas surcharger les tables, sont en moyenne inférieurs à une seconde.

En ce qui concerne les approches évolutionnistes, les meilleurs résultats sont obtenus par le IPGA avec une moyenne de 3.56 véhicules, suivi du PGA avec une moyenne de 3.65 puis du DGA avec une moyenne de 4.04. Le prix à payer pour le IPGA est un temps de calcul plus élevé que les autres méthodes. La meilleure solution est trouvée dans 19 cas par le IPGA, 2 cas par le PGA et 5 cas par le DGA. Le PGA est le meilleur compromis entre temps de calcul et qualité des résultats.

Comparés aux heuristiques constructives, les GA sont bien meilleurs. Ces résultats montrent l'intérêt de prendre en parallèle compte les deux niveaux tactique et opérationnel dans la résolution des problèmes de planification en tournées sur arcs. L'examen des tables 6.4 à 6.5 montre que le PGA donne aussi des solutions mieux équilibrées que les autres méthodes sur l'ensemble des périodes : l'écart entre la meilleure moyenne et la mauvaise moyenne des résultats pour les cinq périodes est de 0.48 pour le PGA et 1.04 pour le DGA.

valeur moyenne	**BIH**	**DGA**	**PGA**	**IPGA**
nvu	7.00	4.04	3.65	3.56
cost	968.4	689.7	706.3	688.9
temps par instance (*min*)	< 1 s	4.63	1.18	9.65

Table 6.2 *– Synthèse des résultats sur nos* 23 *instances gdbf.*

6.7 Conclusion

Dans ce chapitre, nous avons présenté une nouvelle extension temporelle du CARP, nommée PCARP. Nous avons développé deux heuristiques constructives très rapides et nous avons présenté une généralisation multipériode du GA du chapitre 3, baptisée PGA. Ce PGA traite les décisions du niveau opérationnel et du niveau tactique en parallèle. Cette approche globale est novatrice en planification de tournées. Par exemple, les méthodes développées pour les problèmes de planification en tournées sur nœuds procèdent en général en deux phases, et donc traitent les niveaux opérationnel et tactique séparément.

Nous avons vu comment certains composants développés pour le GA pouvaient être réutilisés dans le PGA, comme la structure du chromosome et la procédure d'évaluation Split. Le PGA a par contre nécessité un croisement très sophistiqué, ELOX. Malgré ces différences, notre PGA peut quand même être vu comme une généralisation du GA puisqu'il calcule exactement les mêmes solutions dans le cas mono-période $np = 1$.

Les tests réalisés montrent que le PGA permet d'obtenir un gain significatif comparé à des heuristiques constructives et des méthodes en deux phases consistant à affecter les tâches aux périodes avant de résoudre le CARP pour chaque période. Une post-optimisation du PGA par application récursive de lui-même sur chaque période améliore encore les résultats.

C'est travaux ont fait l'objet de deux conférences internationales que nous citons ci-dessous. La communication présentée à *PMS* 2002 est la première publication sur les problèmes de planification en tournées sur arcs. Elle a présenté le PCARP, la classification de la figure 6.1 et des ébauches de GA. Le GA complet et des évaluations numériques ont fait l'objet d'un second article présenté à IPMU 2002.

A travers cette étude se dégagent des points à approfondir comme le développement de grands jeux d'essai pertinents pour le PCARP, la généralisation de la recherche locale qui est actuellement appliquée seulement période par période et le développement de bonnes bornes inférieures. Cette dernière tâche est certainement très complexe puisqu'aucune bonne borne n'est actuellement publiée pour l'analogue du PCARP en tournées sur nœuds, le PVRP.

Références

P. Lacomme, C. Prins, W. Ramdane-Chérif, Planning problems in arc routing, actes de *PMS* 2002 (*EURO Working Group on Project Management and Scheduling*), p. 232-235. Actes coordonnés par V. Valls et al, édités par Fundacion Universidad-Empresa de Valencia, Valencia, Espagne, 03-05/04/2002.

P. Lacomme, C. Prins, W. Ramdane-Chérif, Evolutionary Algorithms for Multiperiod Arc Routing Problems, actes d'*IPMU* 2002 (*9th Int. Conf. On Information Processing and Management of Uncertainty in Knowledge-Based Systems*), p. 845-852, édités par ESIA–Université de Savoie, Annecy, France, 01-05/07/2002.

File	n	ns	LIH1		LIH2		BIH		DGA		Time	PGA		Time	IPGA		time
			nvu	cost	nvu	cost	nvu	cost	nvu	cost		nvu	cost		nvu	cost	
gdbf1	12	40	5	1315	4	1417	5	1085	3	730	5.9	2	**718**	1.14	2	**718**	33.44
gdbf2	12	46	6	1245	5	1357	5	1026	3	714	4.29	3	732	1.10	3	**707**	4.13
gdbf3	12	40	6	1209	4	1190	5	916	3	618	5.56	2	608	1.13	2	**597**	6.21
gdbf4	11	33	4	1225	4	1212	4	950	3	**632**	4.54	2	650	1.48	2	646	44.34
gdbf5	13	46	6	1505	5	1535	5	1238	3	824	5.11	3	828	1.52	3	**788**	26.55
gdbf6	12	40	5	1307	4	1343	5	1047	3	743	6.52	2	743	1.12	2	**730**	2.03
gdbf7	12	40	6	1292	5	1329	5	1041	3	712	3.38	2	682	1.08	2	**666**	17.69
gdbf10	27	83	9	1831	8	1643	8	1297	6	838	2.48	4	818	1.21	4	**803**	1.95
gdbf11	27	96	11	1629	12	1503	10	1235	6	**764**	2.61	5	790	1.22	5	770	1.95
gdbf12	12	45	4	949	4	1098	3	829	2	671	4.41	2	673	1.12	2	**659**	55.06
gdbf13	22	132	10	2877	9	2746	8	2030	4	1357	2.63	4	1448	1.30	4	**1322**	2.29
gdbf14	13	75	11	3405	11	3029	10	2614	6	**1806**	2.67	6	1895	1.12	5	1843	2.38
gdbf15	10	87	12	2592	11	2547	11	2135	5	**1561**	3.12	6	1709	1.14	5	1710	2.23
gdbf16	7	57	8	566	7	518	6	429	4	329	2.55	3	335	1.09	3	**320**	1.87
gdbf17	7	73	7	319	7	309	7	279	3	227	3.77	3	**225**	1.06	3	**225**	2.14
gdbf18	8	73	9	585	8	602	8	495	5	**389**	6.02	4	401	1.12	4	393	2.21
gdbf19	8	93	9	438	10	436	9	382	4	**318**	6.55	4	328	1.22	4	324	2.16
gdbf20	9	100	9	898	8	877	8	643	4	**516**	5.11	4	538	1.18	4	526	2.59
gdbf21	11	37	6	375	5	367	5	299	3	225	6.91	3	225	1.05	3	**223**	1.81
gdbf22	11	68	8	695	8	619	8	547	4	424	5.13	4	**417**	1.11	4	419	1.93
gdbf23	11	87	11	736	10	707	9	596	5	481	5.81	5	478	1.13	5	**471**	3.25
gdbf24	11	98	11	697	10	708	8	617	6	**527**	5.30	6	530	1.21	6	**527**	1.84
gdbf25	11	105	11	669	10	644	9	543	5	**457**	6.23	5	475	1.25	5	**457**	1.97
Average		70	8.00	1233	7.35	1205.91	7.00	968,39	4.04	689,69	4.63	3.65	706.35	1.18	3.56	688,87	9.65

Table 6.3 – *Résultats du PGA sur les 23 fichiers gdbf.*

	PERIODE1		PERIODE2		PERIODE3		PERIODE4		PERIODE5	
file	nvu	cost	nvu	cost	nvu	cost	nvu	cost	nvu	cost
gdbf1	2	103	2	134	2	129	2	155	2	197
gdbf2	3	218	3	170	2	81	2	146	2	117
gdbf3	2	166	2	108	2	145	2	115	2	74
gdbf4	2	157	2	165	2	125	1	83	1	120
gdbf5	3	145	3	209	3	273	1	69	2	132
gdbf6	2	145	2	148	2	138	2	136	2	176
gdbf7	2	173	2	135	2	177	2	108	1	89
gdbf10	4	169	4	139	4	176	4	177	3	157
gdbf11	4	169	4	150	4	154	5	171	5	146
gdbf12	1	102	2	158	1	118	2	125	2	170
gdbf13	3	277	4	309	4	323	4	268	3	271
gdbf14	5	338	5	339	6	428	6	355	5	435
gdbf15	5	317	6	453	6	404	5	308	4	227
gdbf16	2	49	3	68	3	84	3	59	3	75
gdbf17	2	29	3	55	3	53	2	38	3	50
gdbf18	4	78	4	90	4	89	4	78	3	66
gdbf19	4	59	3	65	4	73	4	68	3	63
gdbf20	4	122	4	143	2	66	3	117	3	90
gdbf21	3	44	2	51	2	53	3	48	2	29
gdbf22	4	93	3	91	4	95	4	88	2	50
gdbf23	4	82	4	94	5	99	5	114	4	89
gdbf24	3	79	4	86	4	112	6	126	5	127
gdbf25	4	72	5	121	5	101	5	113	4	68
Average	3.13	138.52	3.30	151.35	3.30	152	3.35	133.26	2.87	131.22

Table 6.4 – Détail des résultats du PGA par période.

	PERIODE1		PERIODE2		PERIODE3		PERIODE4		PERIODE5	
File	nvu	cost	nvu	cost	nvu	cost	nvu	cost	nvu	cost
gdbf1	3	156	2	140	2	154	2	126	2	154
gdbf2	3	223	2	124	1	101	1	75	3	191
gdbf3	2	144	2	107	2	113	1	60	3	194
gdbf4	2	129	2	143	1	106	1	61	3	193
gdbf5	3	166	2	168	3	203	1	111	2	176
gdbf6	2	154	2	133	2	161	1	79	3	216
gdbf7	3	169	2	140	2	183	1	58	2	162
gdbf10	3	131	3	168	6	247	3	119	4	173
gdbf11	3	134	4	112	6	227	3	107	5	184
gdbf12	2	129	2	119	2	129	2	127	2	167
gdbf13	3	230	4	324	3	277	3	198	4	328
gdbf14	4	303	5	351	5	345	5	401	6	406
gdbf15	3	139	4	301	4	382	5	376	5	363
gdbf16	2	49	3	60	3	78	2	43	4	99
gdbf17	2	38	3	52	3	45	2	38	3	54
gdbf18	3	73	3	65	3	84	3	64	5	103
gdbf19	3	51	4	78	3	53	3	62	4	74
gdbf20	3	110	3	116	3	89	2	82	4	119
gdbf21	2	36	2	39	3	53	2	46	3	51
gdbf22	3	69	4	93	3	75	3	86	4	101
gdbf23	3	93	4	84	5	99	4	91	5	114
gdbf24	3	78	4	86	4	113	5	107	6	143
gdbf25	3	69	3	65	5	110	5	113	5	100
Average	2.74	124.91	3.00	133.39	3.22	149	2.61	114.38	3.78	168.04

Table 6.5 – *Résultats du DGA par période (affectation par BIH).*

File	PERIODE1		PERIODE2		PERIODE3		PERIODE4		PERIODE5	
	nvu	cost	Nvu	cost	nvu	cost	nvu	cost	nvu	cost
gdbf1	2	103	2	134	1	129	2	155	2	197
gdbf2	3	216	3	152	2	81	2	141	2	117
ddbf3	2	165	2	108	2	141	2	109	2	74
gdbf4	2	157	2	165	2	125	1	83	1	116
gdbf5	3	145	3	206	3	236	1	69	2	132
gdbf6	2	137	2	148	2	138	2	136	2	171
gdbf7	2	173	2	134	2	177	2	108	1	74
gdbf10	4	168	4	135	4	187	4	166	3	147
gdbf11	4	161	4	152	4	145	5	174	5	138
gdbf12	1	102	2	158	1	118	2	123	2	158
gdbf13	3	259	4	283	4	272	3	258	3	250
gdbf14	5	327	5	311	5	407	5	385	5	413
gdbf15	4	334	5	445	5	408	5	302	4	221
gdbf16	2	46	3	70	3	78	3	57	3	69
gdbf17	2	29	3	55	3	53	2	38	3	50
gdbf18	4	76	4	92	4	87	4	72	3	66
gdbf19	3	59	3	65	4	73	4	66	3	61
gdbf20	3	122	4	143	2	68	3	108	3	85
gdbf21	2	44	2	49	2	53	3	48	2	29
gdbf22	4	93	3	93	4	95	4	88	2	50
gdbf23	4	81	4	92	5	99	4	112	4	87
gdbf24	3	78	4	86	4	113	6	126	5	124
gdbf25	3	72	5	118	5	98	5	103	3	66
Average	2.91	139.77	3.18	148.91	3.09	149.23	3.14	132.91	2.82	128.59

Table 6.6 – *Résultats du IPGA (affectation par PGA).*

Conclusion générale et perspectives

Nous nous sommes intéressés dans ce mémoire aux problèmes de tournées où les entités à traiter sont les « arcs » (rues, routes) d'un réseau. Ce problème a été beaucoup moins étudié que les problèmes de tournées sur nœuds et pourtant il permet de modéliser plusieurs applications réelles comme la collecte des ordures ménagères, le nettoyage des rues, le fauchage des bas-côtés, l'inspection de lignes à haute tension, etc.

La revue de la littérature que nous avons effectuée sur ce domaine nous a permis de centrer nos recherches sur le CARP (ou *Capacitated Arc Routing Problem*). En effet c'est le problème le plus général en tournées sur arcs et le plus proche des applications réelles. Par contre, il est encore trop simple pour pouvoir proposer des solutions industriellement exploitables. De nombreuses contraintes réelles comme un graphe mixte, les interdictions de tourner, l'incertitude sur les données et l'aspect planification ont été totalement négligés.

Cette thèse peut être considérée comme une extension d'un problème théorique de base, le CARP, vers de nouveaux problèmes généraux intégrant plus de contraintes réelles. Nous avons proposé d'étendre le CARP vers trois problèmes : le *ECARP* (*Extended CARP*), le *SCARP* (*Stochastic CARP*) et le *PCARP* (*Periodic CARP*). Bien que nous ayons choisi comme application de référence *l'optimisation dans la collecte des déchets*, les modèles et les méthodes de résolution que nous avons proposés sont assez généraux et flexibles pour s'adapter à d'autres applications en tournées sur arcs.

Nous avons commencé nos travaux par la définition d'un codage du graphe et des structures de données efficaces supportant ainsi les extensions souhaitées. Ces aspects relatifs au codage des données sont très importants mais souvent négligés dans la littérature. Outre la facilité de maintenance, ils nous permettent de présenter de façon homogène tous les chapitres.

En plus de la proposition de nouveaux problèmes en tournées sur arcs, la contribution la plus opérationnelle de cette thèse porte sur les premiers algorithmes génétiques pour le CARP, qui sont devenus les meilleures méthodes de résolution actuellement disponibles pour ce problème. Ces derniers se sont montrés très flexibles dans le processus de généralisation du CARP et ont permis de constituer une plate-forme logicielle puissante pour le CARP, le SCARP, le ECARP et le PCARP. Chaque problème étendu a bénéficié donc d'au moins un algorithme génétique comme méthode de résolution.

Les algorithmes génétiques que nous avons développés pour le CARP sont des *algorithmes mémétiques* (MA), c'est-à-dire qu'ils sont hybridés avec une recherche locale. Ils ont été testés sur 81 instances de la littérature ayant jusqu'à 140 nœuds et 190 arêtes. Le meilleur MA est au moins aussi bon que les meilleures métaheuristiques publiées. 30 meilleures solutions connues sont améliorées et 3 nouveaux optima ont été découverts.

Pour compléter les résultats théoriques du MA, qui sont très satisfaisants, nous avons analysé sa robustesse face à des variations sur les demandes à traiter. De telles variations peuvent engendrer des coûts supplémentaires non prévus dans un modèle déterministe comme le CARP. Nous avons proposé le *premier modèle stochastique en tournée sur arcs*, appelée *SCARP*. Ce dernier permet de prendre en compte le cas où les quantités à collecter sont stochastiques et qui sont modélisées sous la forme de variables aléatoires. Dans un premier temps une étude *a posteriori* a été réalisée, en perturbant aléatoirement les quantités collectées dans des solutions trouvées par notre algorithme génétique. A l'issue de l'analyse des résultats obtenus, nous avons suggéré des techniques pour intégrer la robustesse *a priori*, pendant le calcul des solutions par l'algorithme génétique. Ces travaux permettent de traiter des applications où les quantités ne sont que des estimations.

La deuxième extension que nous avons proposée est appelée ECARP. Le ECARP utilise un réseau plus complexe et intègre plus de contraintes réelles comme les interdictions de tourner, un lieu de vidage distinct du dépôt et plusieurs lieux de vidage. Nous avons proposé une formulation linéaire du ECARP dérivée d'un modèle linéaire de Golden et Wong (1981) et deux approches de résolution : une méthode de coupes pour les problèmes de petite à moyenne taille et des heuristiques rapides pour les problèmes de grande taille. Le GA a pu être généralisé pour traiter le ECARP et un cas bicritère, dans lequel des surcharges de véhicules sont tolérées. Des jeux d'essai de taille moyenne (*gdbe*) ont été construits en convertissant en ECARP des instances classiques (*gdb*) bien connues pour le CARP de base. Pour les problèmes de grandes tailles, nous avons généralisé trois heuristiques du CARP de base (Path-Scanning, Augment-Merge et la méthode d'Ulusoy). Un générateur aléatoire sophistiqué a été développé pour créer des instances pour le ECARP de grande taille imitant les réseaux réels. Une comparaison des caractéristiques et du comportement de ces heuristiques ainsi que leur implémentation détaillée ont été réalisés. Un résultat surprenant est l'inutilité de la phase Augment dans Augment-Merge. La méthode "Merge" obtenue en enlevant "Augment" devient en fait la meilleure heuristique sur les grandes instances.

Enfin, la dernière contribution de cette thèse est un prolongement naturel de nos travaux vers la planification à moyen terme des tournées sur arcs, donnant un problème appelé *PCARP* ou *Periodic CARP*. Notons que quelques travaux existent pour les problèmes de tournées sur nœuds périodiques, mais rien sur le *PCARP*. Nous avons proposé un modèle général permettant de gérer des espacements entre visites et des demandes ou coûts dépendant du temps. Nous avons conçu des heuristiques constructives et deux approches de résolution. La première consiste à résoudre d'abord le problème d'affectation (niveau tactique), puis à résoudre le CARP obtenu dans chaque période avec notre GA pour le CARP (niveau opérationnel). La deuxième approche, très novatrice, consiste à traiter simultanément les niveaux tactiques et opérationnels par une version étendue du GA, nommée *PGA*. Nous avons montré que les concepts développés initialement pour le GA pouvaient être réutilisés pour le PGA comme la structure du chromosome et les croisements. Les tests réalisés montrent que le PGA permet d'obtenir des gains significatifs comparés aux heuristiques constructives et à la méthode en deux phases. Une post-optimisation du PGA améliore encore considérablement les résultats.

Nos travaux ont trouvé une première voie d'application en collecte des déchets dans le cadre d'une collaboration avec *Onyx* (Pôle Propreté du groupe *VIVENDI*). En fait, suite à la présentation de nos travaux au siège social d'Onyx à Nanterre, les responsables se sont montrés très intéressés par nos recherches et ils ont voulu être notre partenariat technique. Cette collaboration est enrichissante par l'expérience de cette société leader du domaine. Elle nous a permis de comprendre le métier et de descendre sur le terrain de la pratique pour toucher à la complexité des contraintes réelles dans les problèmes de collecte des déchets. Nous avons pu visiter le Centre d'Enfouissement Technique (CET) de Montreuil-sur-Barse prés de Troyes, un dépôt, deux centres de transfert ainsi qu'une agence de déchets industriels à Emerainville. Ces différentes visites ont permis de renforcer nos recherches et d'intégrer plus de réalisme dans nos modèles. Des tests sont en cours de préparation sur des données réelles de la ville de Limoges.

Nos recherches ont fait l'objet de 3 articles dans des revues internationales (1 accepté et 2 soumis), 7 conférences avec actes à ISBN (dont 6 internationales) et 5 conférences sans actes ou avec actes à diffusion restreinte (dont 2 internationales). La liste de nos publications est donnée à la fin de cette conclusion.

Compte tenu de l'intérêt des résultats obtenus, nous souhaitons poursuivre cette recherche dans plusieurs directions :

- valider nos modèles sur d'autres applications réelles en tournées sur arcs ;
- prendre en compte d'autres contraintes dans la résolution du ECARP comme celles liées aux secteurs de collecte, aux flottes hétérogènes et aux multitours ;
- approfondir les aspects tactiques comme suite à nos travaux sur le PCARP, avec par exemple la planification de l'approvisionnement des décharges, des centres de tri et de recyclage. Une thèse sur ce sujet a démarré il y a une année au sein du laboratoire ;
- étendre l'algorithme génétique pour une résolution directe du SCARP ;
- étudier les aspects stratégiques dans les problèmes de collecte de déchets, comme par exemple la reconfiguration de réseaux (emplacement des dépôts et des usines de traitement) ou la conception et le dimensionnement de réseaux de collecte sélective ;
- optimiser des tournées mixtes sur nœuds et arcs, ce qui permettrait d'avoir un algorithme unique pour traiter le VRP, le CARP et des problèmes plus généraux en collecte de déchets, avec des rues à traiter mais aussi des accumulations ponctuelles de déchets (hôpitaux, restaurants).

Toutes ces pistes de recherches très prometteuses, montrent l'importance croissante des problèmes de tournées sur arcs, aussi bien sur le plan académique que sur le plan économique. Nous espérons avoir apporté une contribution utile à ce domaine passionnant.

Liste récapitulative des publications

Article de revues

[1] P. Lacomme, C. Prins, **W. Ramdane-Chérif**, Competitive Memetic Algorithms for Arc Routing Problems, accepté pour *AOR* (*Annal of Operations Research*).

[2] P. Lacomme, C. Prins, **W. Ramdane-Chérif**, Solving general routing problems by a cutting plane algorithm and a genetic algorithm, en révision pour *IIE Transactions*.

[3] G. Fleury, P. Lacomme, C. Prins, **W. Ramdane-Chérif**, Improving sensitivity of Capacitated Arc Routing solutions, soumis à *JORS* (*Journal of the Operational Research Society*).

Communications internationales avec actes à ISBN et comité de lecture

[4] P. Lacomme, C. Prins, **W. Ramdane-Chérif**, A genetic algorithm for the capacitated arc routing problem and its extensions, actes d'*EuroGP*-2001 (4*th European Conf. on Genetic Programming*), p.473-483, publiés dans Applications of evolutionary computing, coordonné par Egbert J.W. Boers et *al*., Lecture Notes in Computer Science 2037, Springer, Côme, Italie, 18-20/04/2001. Prix du «*Best Paper of EvoStim*».

[5] P. Lacomme, C. Prins, **W. Ramdane-Chérif,** Algorithmes génétiques pour le CARP, actes de *MOSIM*'01 (3*ème Conférence Francophone de Modélisation et Simulation*), p. 117-124, A. Dolgui et F. Vernadat (Ed.), SCS Publishing Group, Troyes, France, 25-27/04/2001.

[6] P. Lacomme, C. Prins, **W. Ramdane-Chérif**, An integer linear model for general arc routing problems, actes de ESS 2001 (13*th European Simulation Symposium*), p. 427-432, SCS Publishing Group, Marseille, France, 18-20/10/2001.

[7] G. Fleury, P. Lacomme, C. Prins, **W. Ramdane-Chérif**, Robustness Evaluation of Solutions for the Capacitated Arc Routing Problem, actes de *AIS* 2002 (*AI, Simulation and planning in high autonomy systems*), p. 290-295, édités par Fernando J. Barros et Norbert Giambiasi. Society for modeling and simulation international, Lisbon, Portugal, 07-10/04/2002.

[8] P. Lacomme, C. Prins, **W. Ramdane-Chérif**, Planning problems in arc routing, actes de *PMS* 2002 (*EURO Working Group on Project Management and Scheduling*), p. 232-235. Actes coordonnés par V. Valls et *al*., édités par Fundacion Universidad-Empresa de Valencia, Valencia, Espagne, 03-05/04/2002.

[9] P. Lacomme, C. Prins, **W. Ramdane-Chérif**, Evolutionary Algorithms for Multiperiod Arc Routing Problems, actes d'*IPMU* 2002 (9th *Int. Conf. On Information Processing and Management of Uncertainty in Knowledge-Based Systems*), p. 845-852, édités par ESIA–Université de Savoie, Annecy, France, 01-05/07/2002.

[10] P. Lacomme, C. Prins, **W. Ramdane-Chérif**, General Arc Routing Problems Solved by a Cutting Plane Algorithm and a Genetic Algorithm, actes d' *IFAC*'2002, (15th *Triennial World Congress of the International Federation of Automatic Control*), p. 1195-1200, Barcelona, Spain, 21-26/07/2002.

Communications internationales et nationales sans actes ou avec actes à diffusion restreinte

[11] P. Lacomme, C. Prins, **W. Ramdane-Chérif**, Les problèmes de tournées sur arcs : application à la collecte des déchets ménagers, Conférence invitée, *Journée Technique Transport Urbain et Multimodalité*, Université de Technologie de Belfort-Montbéliard, France, 07/06/2001.

[12] G. Fleury, P. Lacomme, C. Prins, **W. Ramdane-Chérif,** Étude de la robustesse des solutions pour le CARP, *ROADEF* 2002 (*Congrès de l'Association Française de Recherche Opérationnelle et d'Aide à la Décision*), p 178-179, Paris, France, 20-22/02/2002.

[13] **W. Ramdane-Chérif**, Vers des problèmes plus généraux en tournées sur arcs, *ROADEF* 2002 (*Congrès de l'Association Française de Recherche Opérationnelle et d'Aide à la Décision*), p 253-254, Paris, France, 20-22/02/2002.

[14] P. Lacomme, C. Prins, **W. Ramdane-Chérif**, Realistic arc routing problems: fast heuristics and instance generator, actes de *CO* 2002 (*International Sympsium on Combinatorial Optimization*), p. 112, Paris, France, 08 -10/04/2002.

[15] P. Lacomme, C. Prins, **W. Ramdane-Chérif**, Fast algorithms for general arc routing problems, *IFORS* 2002 (*The sixteenth triennial conference of the International Federation of Operational Research Societies*), Edinburgh, Scotland, 08-12/07/2002.

Séminaires

[16] W. **Ramdane-Chérif**, Les problèmes de tournées sur arcs. *Groupe de travail BERMUDES*, bulletin de liaison n° 21, LAMIH / Université de Valenciennes, France, 2/02/2001.

[17] P. Lacomme, C. Prins, **W. Ramdane-Chérif**, Genetic Algorithms for the Capacitated Arc Routing Problem, Université de Valencia, Espagne, 20/12/2001.

[18] P. Lacomme, C. Prins, **W. Ramdane-Chérif**, Algorithmes génétiques pour les problèmes de tournées de véhicules, *journée industrielle Ordonnancement, Transport et Logistique*, Université de technologie de Compiègne, France, 30/11/2002.

BIBLIOGRAPHIE

[1] Alfa A. S., Liu D. Q., 1988. Postman routing problem in a hierarchical network. *Eng. Optim*, 14, 127-138.

[2] Alvarez-Valdes R., Benavent E., Campos V., Corberan A., Mota E., Tamarit J. M., Valls V., 1993. ARC: a computerized system for urban garbage collection. *TOP*, 1 (1), 89-105.

[3] Amberg A., Domschke W., Voβ S., 2000. Multiple center capacitated arc routing problems: a tabu search algorithm using capacitated trees. *European Journal of Operational Research*, 124, 360-376.

[4] Amberg A., Voβ S., 2002. A hierarchical relaxations lower bound for the capacitated arc routing problem. In: Sprague, R. H. (Ed.), *Proceedings of the 35th Annual Hawaii International Conference on System Sciences*, IEEE, Piscataway, DTIST02, 1-10.

[5] Anily S., Gendreau M., Laporte G., 2000. Optimal sequencing of tasks on a tree shaped structure, *Ricerca Operativa*, 29, 3-14.

[6] Assad A., Golden B. L., 1995. Arc routing Methods and Applications. In: M. O. Ball, T. L. Magnanti, C. L. Monma and G. L. Nemhauser (Ed.), *Network Routing, Handbooks in Operations Research and Management Science*, Amsterdam, North-Holland, 8, 375-483.

[7] Assad A., Pearn W. L., Golden B. L., 1987. The capacitated Chinese postman problem: lower bounds and solvable cases. *American Journal of Mathematical and Management Science*, 7, 63-88.

[8] Ball M. O., Magazine M. J., 1988. Sequencing of insertions in printed circuit board assembly. *Operations Research*, 36 (2), 192-201.

[9] Barr R. S., Golden B. L., Kelly J. P., Resende M. G. C., Stewart Jr. W. R., 1995. Designing and reporting on computational experiments with heuristic methods. *Journal of Heuristic*, 1, 9-32.

[10] Belenguer J. M, Benavent E., 1992. Polyhedral results on the capacitated arc routing problem. *Working Paper*, Department of Statistics and Operations Research, University of Valencia, Spain.

[11] Belenguer J. M, Benavent E., 1994. A branch and cut algorithm for the capacitated arc routing problem. *Workshop on Algorithmic Approaches to Large and Complex Combinatorial Optimization Problems*. Giens, France.

[12] Belenguer J. M, Benavent E., 1998. The capacitated arc routing problem: valid inequalities and facets. *Computational Optimization and Applications*, 10, 165-187.

[13] Belenguer J. M, Benavent E., 2002. A cutting plane algorithm for the capacitated arc routing problem. *Working Paper*, Department of Statistics and Operations Research, University of Valencia, Spain. Paru ensuite dans *Computers and Operations Research*, 30 (5), 705 – 728, 2003.

[14] Belenguer J. M., Benavent E., Cognata F., 1997. A tabu heuristic for the capacitated arc routing problem. *Working Paper*, Department of Statistics and Operations Research, University of Valencia, Spain.

[15] Beltrani E. J., Bodin L. D., 1974. Networks and vehicle routing for municipal waste collection. *Networks*, 4, 65-94.

[16] Benavent E., Campos V., Corberan A., 1992. The capacitated arc routing problem: lower bounds. *Networks*, 22, 669-690.

[17] Benavent E., Campos V., Corberan A., Mota E., 1990. The capacitated arc routing problem: a heuristic algorithm. *Questiio*, 14 (1-3), 107-122.

[18] Benavent E., Carrota A., Corberan A., Sanchis J. M., Vigo D., 2002a. Heuristic and lower bound problem for the windy rural postman problem. *Technical Report*, Department of Statistics and Operations Research, University of Valencia.

[19] Benavent E., Corberan A., Piñana E., Plana I., Sanchis J. M., 2002b. A scatter search algorithm for the windy rural postman problem. *Proceedings of the First International Workshop on Heuristics*, Beijing, China, 104-109.

[20] Benavent E., Corberan A., Sanchis J. M., 2000. Linear programming based methods for solving arc routing problems. In: M. Dror (Ed.), *Arc Routing: Theory, Solutions and Applications*, Kluwer Academic Publishers, Boston, 231-275.

[21] Benavent E., Soler D., 1999. The directed rural postman problem with turn penalties. *Transportation Science*, 33 (4), 408-466.

[22] Blanton J. L., Wainwright R. L., 1993. Multiple vehicle routing with time and capacity constraints using genetic algorithms. *Proceeding of the Fifth International Conference on Genetic Algorithms*, Champaign, IL, 452-459.

[23] Bodin L. D., Fagin G., Webno R., Green Beeg J., 1989. The design of computerized sanitation vehicle routing and scheduling system for the town of Oyster Bay. *Computers and Operations Research*, 16, 45-54.

[24] Bodin L. D., Golden B. L., Assad A. A., Ball M., 1983. Routing and scheduling of vehicules and crews. *Computers and Operations Research*, 10 (2), 63-211.

[25] Bodin L. D., Kursh S. J., 1978. A computer-assisted system for the routing and scheduling of street sweepers. *Operations Research*, 26, 525-537.

[26] Bodin L. D., Kursh S. J., 1979. A detailed description of computer systems for the routing and scheduling of street sweepers. *Computers and Operations Research*, 6, 181-198.

[27] Bostel N., Dejax P., Langevin A., Wizere A., 2002. Planification et optimisation de la gestion des déchets : Application au plan d'un département français, *Technical Report* G-2002-02, GERAD, Montréal, Canada.

[28] Branke J., 1998. Creating robust solutions by means of evolutionary algorithms. In: A. Eiben E., Bäck T., Schoenauer M., H-P Schwefel (Ed.), *Parallel Problem Solving from Nature, PPSN'*98, LNCS 1498, Springer Verlag, 119-128.

[29] Brucker P., 1981. The Chinese postman problem for mixed graphs. In: H. Noltemeier (Ed.), *Graph Theoretic Concepts in Computer Science*, Springer, Berlin, 100, 354-366.

[30] Campbell J. F., Langevin A., 1995a. Operations management for urban snow removal and disposal. *Transportation Research*, 29A, 359-370.

[31] Campbell J. F., Langevin A., 1995b. The snow disposal assignment problem. *Journal of the Operational Research Society*, 46, 919-929.

[32] Campbell J. F., Langevin A., 1999. A survey of arc routing applications for roadway snow and ice control. *Technical Report* G-99-03, GERAD, Montréal, Canada.

[33] Campos V., Savall J. V., 1995. A computational study of several heuristics for the DRPP. *Computational Optimization and Applications*, 4, 67-77.

[34] Cerf R., 1994. Une théorie Asymptotique des algorithmes génétiques. *PhD thesis*, Université Montpellier II, France.

[35] Chapleau L., Ferland J. A., Lapalme G., Rousseau J. M., 1984. A parallel insert method for the capacitated arc routing problem. *Operations Research Letters*, 3 (2), 95-99.

[36] Cheung B. K. S., Langevin A., Villeneuve B., 2001. High performing evolutionary techniques for solving complex location problems in industrial system design. *Journal of Intelligent Manufacturing*, 12 (5-6), 455-466.

[37] Christofides N., 1973. The optimum traversal of a graph. *Omega*, 1 (6), 719-732.

[38] Christofides N., 1975. *Graph theory. An algorithmic approach*. Academic Press, London.

[39] Christofides N., 1985. Vehicle Routing. In: E. L. Lawler et *al*. (Ed.), *The Traveling Salesman Problem*, John Wiley and Sons, New York, 431-448.

[40] Christofides N. Beasley J. E., 1984. The period routing problem. *Networks*, 14, 237-256.

[41] Christofides N., Benavent E., Campos V., Corberan A., Mota E., 1984. An Optimal Method for the Mixed Postman Problem. In: P. Thoft-Christensen (Ed.), *System Modelling and Optimisation, Lecture Notes in Control and Inf. Sciences*, Springer, Berlin, 59.

[42] Christofides N., Benavent E., Campos V., Corberan A., Mota E., 1986. An algorithm for the rural postman problem on a directed graph. *Mathematical Programming Study*, 26, 155-166.

[43] Christofides N., Campos V., Corberan A., Mota E., 1981. An algorithm for the rural postman Problem. *Technical Report* ICOR-81.5, Imperial College Report, London.

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

[45] Clossey J., Laporte G., Soriano P, 2001. Solving arc routing problems with turn penalties. *Journal of the Operational Research Society*, 52, 433-439.

[46] Corberan A., Letchford A., Sanchis J. M., 1998. A cutting plane algorithm for the general routing problem. *Working Paper*, Department of Statistics and Operations Research, University of Valencia, Spain.

[47] Corberan A., Marti R., Martinez E., Soler D., 2000b. The rural postman problem on mixed graphs with turn penalities. Soumis à *Computers and Operations Research*.

[48] Corberan A., Marti R., Romero A., 1997. A tabu search algorithm for the mixed rural postman problem. *Working Paper*, Department of Statistics and Operations Research, University of Valencia, Spain.

[49] Corberan A., Marti R., Romero A., 2000c. Heuristics for the mixed rural postman problem. *Computers and Operations Research*, 27 (2), 183-203.

[50] Corberan A., Marti R., Sanchis J. M., 2000a. The general routing problem on a mixed graph. *Working Paper*, Department of Statistics and Operations Research, University of Valencia, Spain.

[51] Corberan A., Romero A., Sanchis J. M., 1997a. Facets of the mixed rural postman problem polyhedron. Communication presented at the *Euro/Informs Conference*. Barcelona, Spain.

[52] Corberan A., Sanchis J. M, 1998. The general routing problem polyhedron: facets from RPP and GTSP polyhedra. *European Journal of Operational Research*, 108, 538-550.

[53] Corberan A., Sanchis J. M., 1994. A polyhedral approach to the rural postman problem. *European Journal of Operational Research*, 79, 95-114.

[54] Cordeau J. F., Gendreau M., Laporte G., 1997. A tabu search heuristic for periodic and multi-depot vehicle routing problems, *Networks*, 30, 105-119.

[55] Cormen T. H., Leiserson C. E., Rivest R. L., 1990. *Introduction to algorithms*. The MIT Press, Cambridge, MA.

[56] Dantzig G. B., Fulkerson D. R., Johnson S. M., 1954. Solution of large scale travelling salesman problem. *Operations Research*, 2, 393-410.

[57] Davis L. D., 1991. *Handbook of genetic algorithms*, Van Nostrand Reinhold, New-York.

[58] Dawkins R., 1976. *The selfish gene*, Oxford University Press.

[59] DeArmon, J. S., 1981. A comparaison of heuristics for the capacitated chinese postman problem. *Master's thesis*, University of Maryland, College Park, MD.

[60] DeJong K., Spears W., 1989. Using genetic algorithms to solve NP-Complete problems. *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufman Publishers, California, 124-132.

[61] Dror M. (Ed.), 2000. *Arc Routing: Theory, Solutions and Applications*. Kluwer Academic Publishers.

[62] Dror M., Houari M., 2000. Generalized Steiner problems and other variants. *Journal of Combinatorial Optimization*, 4 (4), 415-436.

[63] Dror M., Langevin A., 1997. A generalized travelling salesman approach to the directed clustered rural postman problem. *Transportation Science*, 31 (2).

[64] Dror M., Stern H., Trudeau P., 1987. Postman tour on a graph with precedence relation on arcs. *Network*, 17, 283-294.

[65] Edmonds J., Johnson E. L., 1973. Matching Euler tours and the chinese postman problem. *Mathematical Programming*, 5, 88-124.

[66] Eglese R. W., 1994. Routing winter gritting vehicles. *Discrete Applied Mathematics*, 48 (3), 231-244.

[67] Eglese R. W., Letchford A., 2000. Polyhedral theory for arc routing problems. In: M. Dror (Ed.), *Arc Routing: Theory, Solutions and Applications*. Kluwer Academic Publishers, Boston, 199-230.

[68] Eglese R. W., Li L. Y. O., 1992. Efficient routeing for winter gritting. *Journal of the Operational Research Society*, 43 (11), 1031-1034.

[69] Eglese R. W., Li L. Y. O., 1996. A tabu search based heuristic for arc routing with a capacity constraint and time deadline. In: I. H. Osman and J. P. Kelly (Ed.), *Metaheuristics: Theory and Applications*, Kluwer, 633-650.

[70] Eglese R. W., Murdock H., 1991. Routing road sweepers in a rural area. *Journal of the Operational Research Society*, 42, 281-288

[71] Eilselt H. A., Gendreau M., Laporte G., 1995a. Arc routing problems. Part I: The Chinese postman problem. *Operations Research*, 43 (3), 231-242.

[72] Eilselt H. A., Gendreau M., Laporte G., 1995b. Arc routing problems. Part II: The rural postman problem. *Operations Research*, 43 (3), 399-414.

[73] Euler L., 1736. Solutio problematis ad geometriam situs pertinentis, Commentarii academiae scientarum Petropolitanae, 8, 128-140.

[74] Ferland J. A., Chapleau L., Lapalme G., Rousseau J. M., 1984. A parallel insert method for the capacitated arc routing problem. *Operations Research Letters*, 3 (2), 95-99.

[75] Fernandez de Cordoba P., Garcia Raffi L. M., Sanchis J. M., 1998. A heuristic algorithm based on monte carlo methods for the rural postman problem. *Computers and Operations Research*, 25 (12), 1097-1106.

[76] Fisher M., Jaikumar R., 1981. A generalized assignment heuristic for vehicle routing. *Networks*, 11, 109-124.

[77] Filipec M., Škrlec D., Krajcar S., 2000. Genetic algorithm approach for multiple depot capacitated vehicle routing problem solving with heuristic improvements. *International Journal of Modelling and Simulation*, 20 (4), 320-328.

[78] Fleischner H., 1991. Eulerian graphs and related topics (Part I). *Annals of Discrete Mathematics*, North Holland, Amsterdam, 50 (2).

[79] Fleurent C, J.A. Ferland, 1996. Algorithmes génétiques hybrides pour l'optimisation combinatoire. *RAIRO-Operations Research*, 30 (4), 373-398.

[80] Fleury G., 1993. Méthodes stochastiques et déterministes pour les problèmes NP-difficiles. *PhD Thesis*, Université Blaise Pascal – Clermont II, France.

[81] Fogel L. J., Owens A. J., Walsh M. J., 1966. *Artificial intelligence through simulated evolution*. John Wiley.

[82] Ford L. R., Fulkerson D. R., 1962. Flows in networks, *Princeton University Press*, Princeton, NJ.

[83] Frederickson G. N., 1979. Approximation algorithms for some postman problems. *Journal of the Association for Computing Machinery*, 26 (3), 538-554.

[84] Frederickson G. N., Hecht M. S., Kim C. E., 1978. Approximation algorithms for some routing problems. *SIAM Journal on Computing*, 7 (2), 178-193.

[85] Freidlin M. I., Wentzell A. D., 1983. *Random perturbations of dynamical systems*. Springer Verlag, New-York.

[86] Gaudioso M., Paletta G., 1992. A heuristic for the periodic vehicle routing problem. *Transportation Science*, 26 (2), 86–92.

[87] Gélinas E., 1992. Le problème du postier chinois avec contraintes générales de préséance. *M. Sc. A. dissertation*, Ecole Polytechnique de Montréal, Canada.

[88] Gendreau M., Laporte G., Séguin R., 1995. An exact algorithm for the vehicle routing problem with stochastic customers and demands, *Transportation Science*, 29, 143-155.

[89] Gendreau M., Laporte G., Zhao Y., 1990. The windy postman problem on general graphs. *Technical Report* CRT-698, Centre de Recherche sur les Transports.

[90] Ghiani G., Improta G., 2000a. An algorithm for the hierarchical chinese postman problem. *Operations Research Letters*, 26 (1), 27-32.

[91] Ghiani G., Improta G., 2000b. Optimizing Laser-plotter beam movement. *Journal of the Operational Research Society*, Forthcoming.

[92] Ghiani G., Improta G., Laporte G., 2001. The capacitated arc routing problem with intermediate facilities. *Networks*, 37 (3), 134-143.

[93] Ghiani G., Laporte G., 1999. Eulerian location problems. *Networks*, 34, 291-302.

[94] Ghiani G., Laporte G., 2000. A branch and cut algorithm for the undirected rural postman problem. *Mathematical Programming*, 87 (3), 467-481.

[95] Ghiani G., Laporte G., 2001. Location arc routing problems. *Technical Report* G-2001-12, GERAD, Montréal, Canada.

[96] Glover, F., 1989. Tabu search, part I. *ORSA Journal of Computing*, 1, 190-206.

[97] Glover, F., 1990. Tabu search, part II. *ORSA Journal of Computing*, 2, 4-32.

[98] Goldberg D. E., 1989a. *Genetic algorithms*. Addison Wesley.

[99] Goldberg D. E., 1989b. *Genetic algorithms in search, optimization and machine learning*. Reading MA Addison Wesley.

[100] Golden B. L., Dearmon J. S., Bakes E. K., 1983. Computational experiments with algorithms for a class of routing problems. *Computers and Operation Research*, 10 (1), 47-59.

[101] Golden B. L., Wong R. T., 1981. Capacitated arc routing problems. *Networks*, 11, 305-315.

[102] Grefenstette J. J., Gopal R., Rosmaita B. R., Gucht D. V., 1985. Genetic algorithms fort the traveling salesman problem. In: J. J Grefenstette (Ed.), *Proceeding of the First International Conference on Genetic Algorithms*, Lawrence Erlbaum Associates, New Jersey, 160-168.

[103] Greistorfer P., 2003. A tabu scatter search metaheuristic for the arc routing problem. A paraitre dans *Computers and Industrial Engineering.*.

[104] Grötschel M., Holland O., 1985. Solving matching problems with linear programming. *Mathematical Programming*, 33, 243-259.

[105] Grötschel M., Win Z., 1988. On the windy postman polyhedron. *Technical Report* 75, Schwerpunkt-Program der Deutschen Forschungsgeneinschaft, Universitat Augsburg.

[106] Grötschel M., Win Z., 1992. A cutting plane algorithm for the Windy Postman Problem. *Mathematical Programming*, 55, 339-358.

[107] Guan H., 1993. Polyhedral structure and efficient algorithms for certain classes of directed rural postman problem. *PhD dissertation*, Applied Math. Program, University of Maryland at College Park, Md.

[108] Guan M., 1962. Graphic programming using odd and even points. *Chinese Mathematics*, 1, 273-277.

[109] Guan M., 1984. On the windy postman problem. *Discrete Applied Mathematics*, 9, 41-46.

[110] Gueguen C., 1999. Méthode de résolution exacte pour les problèmes de tournées de véhicules. *Thèse de doctorat*. Informatique et Recherche Opérationnelle. Ecole Centrale Paris, France.

[111] Hartmann S., 1998. A competitive genetic algorithm for resource-constrained project scheduling. *Naval Research Logistics*, 45 (7), 733-750.

[112] Haslam E., Wright J. R., 1991. Applications of routing technologies to rural snow and ice control. *Transportation Research*, 1304, 202-394.

[113] Hertz A., Laporte G., Mittaz M., 2000. A tabu search heuristic fort the capacitated arc routing problem. *Operations Researh*, 48 (1), 129-135.

[114] Hertz A., Laporte G., Nanchen-Hugo P., 1999. Improvement procedures for the undirected rural postman problem. *INFORMS Journal on Computing*, 11 (1), 53-62.

[115] Hertz A., Mittaz M., 1998. Arc routing problems: heuristic algorithms. *Working Paper* 98/09. Ecole Polytechnique Fédérale de Lausanne, Suisse.

[116] Hertz A., Mittaz M., 2000. Heuristic algorithms. In: M. Dror (Ed.), *Arc Routing: Theory, Solution and Apllications,* Kluwer Academic Publishers, chapter 9, 327-386.

[117] Hertz A., Mittaz M., 2001. A variable neighbourhood descent algorithm for the undirected capacitated arc routing problem. *Transportation Science,* 35 (4), 425-434.

[118] Hierholzer, 1873. Uber die möglichkeit einen linienzug ohne widerholung und ohne unterbrechung zu umfahren, *Mathematische Annalen*, 6, 30-32.

[119] Hirabayashi R., Saruwatari Y., Nishida N., 1992. Tour construction algorithm for the capacitated arc routing problem. *Asia-Pacific Journal of Operational Research,* 9, 155-175.

[120] Holland J. H., 1975. *Adaptation in natural and artificial systems*. University of Michigan press.

[121] Kapauf C. H., Koehler G. J., 1979. The mixed postman problem. *Discrete Applied Mathematics,* 1, 89-103.

[122] Kaufmann. Graphs, 1967. *Dynamic programming and finite graphs*. Academic press, New York.

[123] Kirkpatrick S., Gelatt C. D., Vecchi M., 1983. Optimization by simulated annealing. *Science,* 220, 671-680.

[124] Kiuchi M., Shinano Y., Hirabayashi R., Saruwatari Y., 1995. An exact algorithm for the capacitated arc routing problem using parallel branch and bound method. Abstracts of the *1995 Spring National Conference of the Oper. Res. Soc. of Japan*, Japanese, 28-29.

[125] Labelle A., Langevin A., Campbell J. F., 2000. Sector design for snow removal and disposal in urban areas. *Technical Report* G-2000-17, GERAD, Montréal, Canada.

[126] Laporte G., 1992a. The travelling salesman problem: an overview of exact and approximate algorithms. *European Journal of Operational Research,* 59 (2), 231-247.

[127] Laporte G., 1992b. The vehicle routing problem: an overview of exact and approximate algorithms. *European Journal of Operational Research,* 59 (3), 345-358.

[128] Laporte G., 1995. Modeling and solving several classes of arc routing problems as traveling salesman problems. *Technical Report* 95-81, Centre de Recherche sur les Transports, Montréal, Canada.

[129] Laporte G., 1996. Selected readings on vehicle routing. *Technical Report* 96-10, Centre de Recherche sur les Transports, Montréal, Canada.

[130] Laporte G., 1998. Exact algorithms for the travelling salesman problem and the vehicle routing problem. *Technical Report* G-98-37, GERAD, Montréal, Canada.

[131] Laporte G., Louveaux F. V., 1999. Solving stochastic routing problems with the integer L-shaped method. In: Grainic T. G. and Laporte G. (Ed.), *Fleet Management and Logistics*, Kluwer, 159-167.

[132] Laporte G., Nobert Y., 1984. *Exact algorithms for the vehicle routing problem*. North-Holland, Amsterdam.

[133] Lawler E. L., 1976. *Combinatorial Optimization: Networks and Matroid*s. Holt, Rinehart and Winston, New-York.

[134] Lemieux P. F., Campagna L., 1984. The snow plowing problem solved by a graph theory algorithm. *Civil Eng. Systems*, 1, 337-341.

[135] Lenstra J. K., Rinnooy K., 1976. On general routing problems. *Networks*, 6, 273-280.

[136] Letchford A. N., 1996. Polyhedral results for some constrained arc-routing problems. *PhD thesis*, Department of Management Science, Lancaster University, United Kingdom.

[137] Letchford A. N., 1997a. New inequalities for the general routing problem. *European Journal of Operational Research,* 96, 317-322.

[138] Letchford A. N., 1997b. The general routing polyhedron: a unifying framework. *Forthcoming in European Journal of Operational Research.*

[139] Letchford A. N., Eglese R. W., 1998. The rural postman problem with deadline classes. *European Journal of Operational Research,* 105, 390-400.

[140] Levy L., Bodin L., 1988. Scheduling the postal carriers for the United States Postal Service: an application of arc partitioning and routing. In: B. Golden and A. Assad (Ed.), *Vehicle Routing: Methods and Studies*, North-Holland, New York, N. Y., 359-394.

[141] Li L. Y. O., 1992. Vehicle Routeing for Winter Gritting. *PhD dissertation,* Department of OR et OM. Lancaster University, United Kingdom.

[142] Li L. Y. O., Eglese R., W., 1996. An interactive algorithm for vehicle routeing for winter gritting. *Journal of the Operational Research Society*, 47, 217-228.

[143] Liebling T. M., 1970. Graphentheorie in planungs – und Tourenproblemen. *Lecture Notes in Operations Research and Mathematical System,* Springer, Berlin, 21.

[144] Liebling T. M., 1973. Routing problems for street cleaning and snow removal. In: R.A. Deininger (Ed.), *Models for Environmental Pollution Control*, Ann Arbor, Michigan, 363-375.

[145] Lin Y., Zhao Y., 1988. A new algorithm for the directed Chinese postman problem. *Computers and Operation Research*. 15 (6), 577-584.

[146] Lotan T., Carttrysse D., Van Oudheusden D., 1996. Winter gritting in the province of Antwerp - a combined location and routing problem. *JORBEL Belgian Journal of Operations Research, Statistics and Computer Science*, 36, 141-157.

[147] Lukka A., Salminen L., 1987. Comparison of three heuristics for an arc routing problem. *Research Report* 6/1987. Department of information technology, Lappeenranta University of Technology, Finland.

[148] Male J. W., 1973. A heuristic solution to the m-postmen's problem. *PhD dissertation*, The Johns-Hopkins University, Baltimore, Md.

[149] McBride R., 1982. Controlling left and U-turns in the routing of refuse collection vehicles. *Computers and Operations Research*, 9, 145-152.

[150] Metropolis N., Rosenbluth A., Rosenbluth M., Teller A., Teller E., 1953. Equation of state calculations by fast computing machines. *Journal of Chem. Physics*, 21 (66), 1087-1092.

[151] Michalewicz Z., 1992. *Genetic algorithms + Data structures = Evolution programs*. Springer Verlag.

[152] Minieka E., 1979. The Chinese postman problem for mixed networks. *Management Science*, 25 (7), 643-648.

[153] Mingozzi A., Valletta A., 2002. A lower bound for the period vehicle routing problem. Communication orale, *International Symposium on Combinatorial Optimization* (CO'02), Paris, France.

[154] Mittaz M., 1999. Problèmes de cheminements optimaux dans des réseaux avec contraintes associées aux arcs. *PhD thesis*, Department of Mathematics, Ecole Polytechnique Fédérale de Lausanne, Switzerland.

[155] Mladenovic N., Hansen P., 1997. Variable neighbourhood arc routing problem. *Computers and Operations Research*, 34, 1097-1100.

[156] Moscato P., 1999. Memetic algorithms: a short introduction. In: Corne D., Dorigo M., Glover F. (Ed.), *New Ideas in Optimization*, McGraw-Hill, 219–234.

[157] Mourão M. C, Almeida M. T., 2000. Lower Bounding and Heuristic methods for a refuse collection vehicle routing problem. *European Journal of Operational Research*, 121, 420-434.

[158] Muhlenbein H., George-Schleuter M., Kramer O., 1988. Evolution algorithms in combinatorial optimization. *Parallel Computing*, 7, 65-85.

[159] Mullaseril P. A., Dror M., 1996. A set covering approach for directed node and arc routing problems with split deliveries and time windows. *Technical Report*, MIS Department, University of Arizona.

[160] Muralidharan B., Pandit R., 1991. Routing and scheduling of trucks to minimise the cost of collection of post-consumer recyclable products. *Working Paper*, Department of Industrial and Manufacturing System Engineering, Iowa State University.

[161] Nobert Y., Picard J. C., 1996. An optimal algorithm for the mixed chinese postman problem. *Networks*, 27, 95-108.

[162] Orloff C. S., 1974. A fundamental problem in vehicle routing. *Networks*, 4, 35-64.

[163] Papadimitriou C. H., 1976. On the complexity of edge traversing. *Journal of the Association for Computing Machinery*, 23, 544-554.

[164] Pearn W. L., 1988. New lower bounds for the capacitated arc routing problem. *Networks*, 18 (3), 181-191.

[165] Pearn W. L., 1989. Approximate solutions for the capacitated arc routing problem. *Computers and Operations Research*, 16 (6), 589-600.

[166] Pearn W. L., 1991. Augment-insert algorithms for the capacitated arc routing problem. *Computers and Operations Research*, 18 (2), 189-198.

[167] Pearn W. L., 1994. Solvable cases of the k-person Chinese postman problem. *Operations Research Letters*, 16 (4), 241-244.

[168] Pearn W. L., Assad A. A., Golden B. L., 1987. Transforming arc routing into node routing problems. *Computers and Operations Research*, 14 (4), 285-288.

[169] Pearn W. L., Chou J. B, 1999. Improved solutions for the Chinese postman problem on mixed networks. *Computers and Operations Research*, 26, 819-827.

[170] Pearn W. L., Liu C. M., 1995. Algorithms for the Chinese postman problem on mixed networks. *Computers and Operations Research*, 22 (5), 479-489.

[171] Pearn W. L., Liu C. M., 1999. Improvement solutions for the chinese postman problem on mixed networks. *Computers and Operations Research*, 26 (8), 819-827.

[172] Pearn W. L., Wu T. C., 1995. Algorithms for the rural postman problem. *Computers and Operations Research*, 22, 819-828.

[173] Potvin J. Y., Bengio S., 1993. A genetic approach to the vehicle routing problem with time windows. *Technical Report* CRT-953, Centre de Recherche sur les Transports, Université de Montréal, Québec.

[174] Potvin J. Y., Bengio S., 1996. The vehicle routing problem with time windows - Part II: Genetic Search. *Informs Journal Computing*, 8 (2), 165-172.

[175] Prins C., 1993. *Algorithmes de graphes*. Eyrolles, Paris.

[176] Prins C., 2000. Competitive genetic algorithms for the open-shop scheduling problem. *Mathematical Methods of Operations Research*, 51, 540-564.

[177] Reeves C.R., 1995. A genetic algorithm for flowshop sequencing. *Computers and Operations Research*, 22 (1), 5-13.

[178] Reeves C. R., 1996. *Modern heuristic techniques for combinatorial problems*. Blackwell Scientific, Oxford, United Kingdom.

[179] Romero A., 1997. On mixed rural postman problem (in Spanish). *PhD thesis*, Department of Statistics and Operations Research, University of Valencia, Spain.

[180] Roy S., Rousseau J. M., 1989. The capacitated canadian postman problem. *INFOR*, 27 (1), 58-73.

[181] Sanchis J. M., 1990. El Poliedro del problema del cartero rural. *PhD thesis*, University of Valencia, Spain.

[182] Saruwatari Y., Hirabayashi R., Nishida N., 1992. Node duplication lower bound for the capacitated arc routing problems. *Journal of the Operations Research Society of Japan*, 35 (2), 119-133.

[183] Savall J. V., 1990. Polyedral results and approximate algorithms for the directed rural postman problem. *PhD dissertation* (in Spanish), Department of Statistics and Operations Research, University of Valencia, Spain.

[184] Shuster K. A., Schur D. A., 1974. Heuristic routing for solid waste collection vehicles. U.S. *Environmental Protection Agency Publication SW-113*, U. S. Government Printing Office, Washington, D. C.

[185] SPEC, 2001. Standard Performance Evaluation Corporation. *http://www. spec.org*.

[186] Starkweather T., McDaniel S., Mathias K., Whitley D., Whitley A., 1991. A comparaison of genetic sequencing operators. In: R. K. Belew and L. B. Booker (Ed.), *Proceedings of the Fourth International Conference on Genetic Algorithms*, Morgan Kaufman, California, 69-76.

[187] Stern H., Dror M., 1979, Routing electric meter readers. *Computers and Operations Research*, 6, 209-223.

[188] Stricker R., 1970. Public sector vehicle routing: the chinese postman's problem. *Masters thesis*, Massachusetts Institute of Technology, Cambridge, Mass.

[189] Thangiah S. R., 1993. Vehicule routing with time windows using genetic algorithms.In: L. Chambers (Ed.), *Application Handbook of Genetic Algorithms*: New *Frontiers*, CRC Press, Florida, 2.

[190] Tsutsui A. Ghosh A., 1997. Genetic algorithms with a robust solution searching scheme. *IEEE Transactions on Evolutionary Computation*, 1(3), 201-208.

[191] Tung D. V., Pinnoi A., 2000. Vehicle routing-scheduling for waste collection in Hanoi. *European Journal of Operational Research*, 125, 449-468.

[192] Ulusoy G., 1985. The fleet size and mixed problem for capacitated arc routing. *European Journal of Operational Research*, 22 (3), 329-337.

[193] Van Aardenne-Ehrenfest T., De Bruijn N. G., 1951. Circuits and trees in oriented linear graphs. *Simon Stevin*, 28 (3), 203-217.

[194] Van Laarhoven P. J. M., Aarts E. H. L., 1987. *Simulated annealing: theory and applications*. Kluwer Academic Publishers.

[195] Verweij B., Ahmed S., Kleywegt A., Nemhauser G., Shapiro A., 2001. The sample average approximation method applied to stochastic routing problems: a computational study. *Technical Report*, The Logistic Institute, Georgia Institute of Technology.

[196] Welz S. A., 1994. Optimal solutions for the capacitated arc routing problem using integer programming. *PhD thesis*, Department of QT and OM, University of Cincinnati.

[197] Win Z., 1987. Contributions to routing Problems. *PhD thesis*, University of Augsburg, Germany.

[198] Win Z., 1989. On the windy postman problem on eulerian graphs. *Mathematical Programming*, 44, 97-112.

[199] Witucki M, Dejax P, Haouari M., 1997. Un modèle et un algorithme de résolution exacte pour le problème de tournées de véhicules multipériodique : une application à la distribution des gaz industriels, *Deuxième Congrès International Fronco-Québécois de Génie Industriel*, Albi, France.

[200] Wunderlich J., Collette M., Levy L., Bodin L. D., 1992. Scheduling meter readers for Southern California Gas Company. *Interfaces*, 22 (3), 22-30.

[201] Zhang L., 1992. Polynomial algorithms for the k-chinese postman problem. In: Jan van Leeuwen (Ed.), *Proceeding of the IFIP 12th World Computer Congress*: Algorithms, Software, Architecture. Elsevier, 1, 430-435.