Université Paris-Dauphine
Laboratoire d'Analyse et Modélisation de Systèmes pour l'Aide à la Décision

THÈSE
pour l'obtention du grade de
DOCTEUR EN INFORMATIQUE

Résolution de problèmes d'optimisation combinatoire mono et multi-objectifs par énumération ordonnée

Candidat

Lyes BELHOUL

Jury

Daniel VANDERPOOTEN
Professeur à l'Université Paris-Dauphine (Directeur de thèse)

Lucie GALAND
Maître de Conférences à l'Université Paris-Dauphine (Co-encadrante)

André ROSSI
Maître de Conférences HDR à l'Université de Bretagne-Sud (Rapporteur)

Jacques TEGHEM
Professeur à la Faculté Polytechnique de l'Université de Mons (Rapporteur)

Olivier SPANJAARD
Maître de conférences HDR à l'Université Pierre et Marie Curie (Examinateur)

Date de soutenance prévisionnelle 13 Décembre 2014

Table des matières

Introduction **1**

1 Notions préliminaires **5**

1.1 Définitions des problèmes d'optimisation 5

1.1.1 Problèmes d'optimisation 6

1.1.2 Problèmes d'optimisation discrète 6

1.1.3 Problèmes d'optimisation combinatoire 7

1.1.4 Problèmes d'optimisation combinatoire multiobjectif 10

1.2 Schéma général de résolution par Branch and Bound 16

1.2.1 Séparation 16

1.2.2 Évaluation 17

1.3 Problème d'affectation et algorithmes de k-best 19

1.3.1 Problème d'affectation 19

1.3.2 Algorithmes de k-best pour le problème d'affectation 23

1.4 Contexte de la thèse 28

2 Principe général de l'énumération ordonnée **31**

2.1 Principe général de résolution 31

2.2 Énumération ordonnée et algorithmes de k-best 34

2.3 Énumération ordonnée et minoration variable 37

2.4 Amélioration des performances des algorithmes par énumération ordonnée 39

2.4.1 Algorithme de recherche de la $k^{\text{ème}}$ solution avec règles d'élagage . 40

2.4.2 Suppression des sous-ensembles vides 41

2.4.3 Suppression des sous-ensembles non-améliorants 41
2.5 Conclusion 44

3 Solution de compromis pour le problème d'affectation multiobjectif 47
3.1 Définitions préliminaires 48
3.2 Énumération ordonnée avec minoration unique 50
3.2.1 Fonction minorante 51
3.2.2 Recherche de la $k^{\text{ème}}$ meilleure solution avec minoration unique . . 56
3.2.3 Bornes $\hat{B}$ et $\tilde{B}$ 58
3.2.4 Ordre de séparation 61
3.3 Énumération ordonnée avec minoration variable 62
3.3.1 Recherche de la $k^{\text{ème}}$ meilleure solution avec minoration variable . . 63
3.3.2 Résolution du problème d'affectation associé à un nœud de l'arborescence dans la version avec minoration variable 64
3.4 Résultats expérimentaux 66
3.4.1 Conséquences des améliorations sur la procédure d'énumération ordonnée avec minoration unique 67
3.4.2 Résultats expérimentaux pour la procédure d'énumération ordonnée avec minoration variable 75
3.4.3 Instances difficiles 78
3.5 Conclusion 81

4 Problème du voyageur de commerce asymétrique 83
4.1 Définitions préliminaires 83
4.2 Énumération ordonnée pour **ATSP** 85
4.2.1 Algorithmes fondés sur la relaxation en problème d'affectation dans la littérature 86
4.2.2 Algorithmes par énumération ordonnée pour **ATSP** 88
4.3 Amélioration de l'énumération ordonnée pour **ATSP** 91
4.3.1 Recherche d'une bonne tournée 92
4.3.2 Critères de choix de la sous-tournée de séparation 92
4.3.3 Recherche de solutions équivalentes 93
4.3.4 Principe et mode de calcul de la borne $\tilde{B}$ 94

4.3.5 Gestion de l'ensemble des arêtes imposées 102
4.4 Résultats expérimentaux 102
4.4.1 Remarques préliminaires sur la partie expérimentale 102
4.4.2 Résultats expérimentaux 106
4.4.3 Synthèse des résultats expérimentaux 114
4.5 Conclusion 116

Conclusion **119**

Table des figures

1.1 Interprétation géométrique de la pseudo-distance de type point de référence 13
1.2 Graphe biparti complet 20
1.3 Problème d'affectation sous la forme du problème de flot maximum à coût minimum 21
1.4 Itération de l'algorithme hongrois avec graphe d'écart 23
1.5 Schéma de séparation de Murty-Lawler pour le problème d'affectation . . 24
1.6 Plus court chemin améliorant pour un nœud de l'arborescence de Murty-Lawler 25
1.7 Schéma de séparation de Chegireddy-Hamacher 27

2.1 Description d'un algorithme d'énumération ordonnée 33

3.1 Amélioration de la condition d'arrêtde l'énumération ordonnée dans le cas de coûts entiers 51
3.2 Principe de la borne $\hat{B}$ 59
3.3 Complémentarité entre les bornes $\hat{B}$ et $\tilde{B}$ 60
3.4 Principe des instances concaves 80
3.5 Instance concaves dans l'espace des objectifs 80

4.1 Relaxation de **ATSP** en problème d'affectation 85
4.2 Correspondances entre les solutions de **ATSP** et les solutions du problème d'affectation 85
4.3 Séparation de Murty naïve pour **ATSP** 87
4.4 Exemple d'un plus court chemin dans le graphe d'écart 95
4.5 Matrice des coûts réduits et solution du nœud séparé 100

4.6 Exemple de déroulement de l'algorithme 4.4 101
4.7 Plus courts chemins résultant de l'algorithme 4.4 101
4.8 Arc interdit induit par un chemin imposé 102

Liste des tableaux

2.1 Récapitulatif des bornes 44

3.1 Comparaison de différents modes d'initialisation (temps CPU en secondes) 68
3.2 Nombre de solutions énumérées (pourcentage des solutions énumérées après l'obtention de la solution optimale) 70
3.3 Taille de l'arborescence 71
3.4 Temps CPU moyens pour différentes variantes (secondes) 73
3.5 Temps CPU moyens de la variante $\hat{B}\tilde{B}R$ pour les grandes instances (secondes) 74
3.6 Comparaison des variantes avec minoration unique et variable 75
3.7 Temps CPU des versions unique et variable pour $7 \leq p \leq 14$ (secondes) . 77

4.1 Statistiques sur les instances de la TSPLIB 103
4.2 Temps CPU des variantes non-dominées pour les instance **ftv** (secondes) . 107
4.3 Résultats pour les instances **ftv** 108
4.4 Temps CPU pour les instances **rbg** (secondes) 110
4.5 Résultats pour les instances **ft** 111
4.6 Résultats pour les instances **ry48p** et **kro124p** 112
4.7 Temps CPU des variantes non-dominées pour toutes les familles d'instances (secondes) 115

Liste des Algorithmes

2.1 Résolution de $\mathcal{P}(X, f)$ par énumération ordonnée 33

2.2 $k^{\text{ème}}$ solution selon Murty-Lawler 35

2.3 $k^{\text{ème}}$ solution avec minoration choisie 38

2.4 $k^{\text{ème}}$ solution avec tests de bornes 40

3.1 Résolution du problème (3.2) par l'énumération ordonnée 50

3.2 $k^{\text{ème}}$ meilleure solution pour résoudre (3.2) par l'énumération ordonnée avec minoration unique 57

3.3 $k^{\text{ème}}$ meilleure solution pour résoudre (3.2) par l'énumération ordonnée avec minoration variable 64

4.1 Énumération ordonnée par Murty et al. (1962) pour le problème du voyageur de commerce asymétrique 86

4.2 Résolution de **ATSP** par énumération ordonnée des affectations 89

4.3 $k^{\text{ème}}$ meilleure affectation pour résoudre **ATSP** 89

4.4 Plus court chemin sortant de la sous-tournée $\mathcal{C}$ 98

4.5 Procédure de mise à jour des étiquettes 99

Introduction

L'optimisation combinatoire définit un cadre formel pour de nombreux problèmes de de l'industrie, de la finance ou de la vie quotidienne. Les problèmes d'optimisation combinatoire sont habituellement définis comme une problématique de choix d'une meilleure alternative dans un ensemble très grand mais fini d'alternatives. En raison du très grand nombre d'alternatives pour ces problèmes, l'ensemble des alternatives, dit aussi *ensemble de solutions réalisables*, est défini en compréhension, en d'autres termes les solutions réalisables se distinguent par un ensemble de propriétés ou de conditions, dites aussi *contraintes*, qu'elles doivent toutes remplir. Une évaluation est associée à toute solution réalisable à l'aide d'une fonction dite *fonction objectif*. Résoudre un tel problème consiste donc à trouver une solution *optimale*, c'est-à-dire trouver une solution réalisable qui minimise ou maximise, selon le contexte, la fonction objectif.

Les problèmes d'optimisation combinatoire peuvent s'avérer très difficiles à résoudre bien qu'ils soient généralement faciles à formaliser. La difficulté de ces problèmes a pour origine soit la structure de l'ensemble réalisable soit la nature de la fonction objectif. C'est pour cette raison que des approches de résolution proposées dans la littérature pour résoudre des problèmes d'optimisation combinatoire difficiles utilisent la technique de *relaxation*. L'approche la plus connue faisant appel à la relaxation est certainement la méthode par séparation et évaluation, dite aussi *Branch and Bound* . La relaxation associe un problème dit *relâché* au problème original par l'omission de contraintes difficiles et/ou par la minoration dans le cas de la minimisation (majoration dans le cas de la maximisation) de la fonction objectif avec une nouvelle fonction plus facile à optimiser. Le problème relâché est ainsi plus facile à résoudre que le problème original. Dans les approches de Branch and Bound la résolution de plusieurs problèmes relâchés, obtenus en divisant l'ensemble réalisable en plusieurs sous-ensembles, conduit à la solution optimale.

Nous nous intéressons particulièrement dans cette thèse à des problèmes d'optimisation combinatoire difficiles qui admettent des problèmes d'optimisation combinatoire faciles comme relaxation. Nous proposons une méthode qui procède par *énumération ordonnée*, qui consiste à associer un problème d'optimisation combinatoire relâché au

problème original et à énumérer des solutions du problème relâché dans un ordre non décroissant selon la fonction objectif du problème relâché, et ce jusqu'à l'obtention de la solution optimale du problème principal. La validité de cette approche repose sur le caractère fini des ensembles admissibles des problèmes d'optimisation combinatoire, puisqu'au pire cas toutes les solutions réalisables de la relaxation sont énumérées, et ainsi la solution optimale du problème difficile est forcément obtenue. L'efficacité de ces approches dépend de deux aspects, à savoir, le nombre de solutions énumérées et l'effort fourni pour l'énumération de chacune d'entre elles. Ces deux aspects sont conflictuels, et c'est pour cette raison que nous enrichissons le principe général de l'énumération ordonnée avec des perfectionnements qui tendent à améliorer les performances générales de nos procédures. Nous mettons en œuvre nos procédures sous la forme d'algorithmes de Branch and Bound, aussi nos améliorations interviennent à différents niveaux de la séparation et de l'évaluation.

Nous proposons aussi dans cette thèse deux applications de notre algorithme général pour des problèmes d'optimisation combinatoire difficiles qui admettent le *problème d'affectation* comme relaxation. Le premier de ces problèmes est *la recherche de solution de compromis pour le problème d'affectation multiobjectif*, où la relaxation consiste à minorer la fonction objectif non-linéaire par une fonction linéaire. Nous étudions aussi le *problème du voyageur de commerce asymétrique* que l'on peut relâcher pour obtenir un problème d'affectation, et ce en supprimant une famille de contraintes dites *contraintes de sous-tournées* . Les résultats expérimentaux occupent une part importante de cette thèse. Nous testons l'efficacité de nos algorithmes dans la partie dédiée à l'expérimentation, par l'implémentation informatique et la résolution d'instances aléatoires ou réelles lorsque celles-ci sont disponibles.

Hormis cette introduction et la conclusion, ce rapport contient quatre chapitres. Le chapitre 1 englobe quelques notions élémentaires de la théorie de l'optimisation. Nous rappelons les définitions de quelques classes de problèmes d'optimisation, de la théorie des graphes et de la théorie de la complexité. Puis nous abordons brièvement dans ce chapitre les méthodes les plus connues pour la résolution des problèmes d'optimisation combinatoire, en mettant l'accent sur quelques détails du Branch and Bound. Enfin nous rappelons différentes formulations du problème d'affectation et quelques-uns des algorithmes les plus connus pour le résoudre et pour énumérer ses solutions de manière ordonnée.

Nous présentons nos contributions dans les chapitres suivants. Dans le chapitre 2, nous présentons un algorithme générique pour résoudre des problèmes d'optimisation combinatoire par énumération ordonnée avant de proposer deux variantes de cet algorithme basées sur le Branch and Bound et le schéma de séparation de Murty-Lawler. Nous présentons à la fin de ce chapitre des *règles d'élagage* de l'arborescence de recherche, utiles pour améliorer les performances de nos algorithmes.

Le chapitre 3 décrit les algorithmes que nous proposons pour la recherche d'une solution de *bon compromis* pour le problème d'affectation multiobjectif. Après avoir formalisé le problème, nous précisons les spécificités de la procédure générique d'énumération ordonnée lorsqu'elle est utilisée dans le contexte de la recherche de bon compromis et nous proposons ensuite différentes versions de cette procédure. Nous présentons à la fin de ce chapitre une synthèse des résultats expérimentaux que nous avons obtenus par la mise en œuvre informatique de nos algorithmes.

Nous proposons dans le chapitre 4 une méthode de résolution pour le problème du voyageur de commerce asymétrique par l'énumération ordonnée de solutions réalisables pour le problème d'affectation. Après une présentation du problème du voyageur de commerce asymétrique et de son lien avec le problème d'affectation, nous discutons des méthodes principales présentes dans la littérature qui se fondent sur la relaxation en problème d'affectation. Nous décrivons ensuite l'algorithme que nous proposons pour résoudre ce problème, puis nous apportons des précisions sur les améliorations que nous proposons d'introduire. Nous montrons à la fin de ce chapitre des résultats expérimentaux obtenus par la résolution d'instances issues de la banque d'instances TSPLIB.

En conclusion, nous récapitulons les avantages et les inconvénients des méthodes par énumération ordonnée, aussi bien dans le cadre général des problèmes d'optimisation combinatoire que dans les cas spécifiques aux applications que nous proposons. Diverses perspectives d'étude sont proposées pour clore ce chapitre.

Chapitre 1
Notions préliminaires

Résumé Ce chapitre englobe quelques notions élémentaires de la théorie de l'optimisation et de la recherche opérationnelle. Dans un premier temps nous rappelons les définitions de quelques classes de problèmes d'optimisation, de la théorie des graphes et de la théorie de la complexité dans la section 1.1. Ensuite nous abordons brièvement les méthodes de résolution des problèmes d'optimisation combinatoire en mettant l'accent sur quelques détails du Branch and Bound dans la section 1.2. Les applications que nous proposons dans cette thèse sont basées sur le problème d'affectation, aussi nous rappelons différentes formulations du problème d'affectation dans la section 1.3.1 et quelques-uns des algorithmes les plus connus pour le résoudre. Nous rappelons ensuite les deux schémas principaux des algorithmes de k-meilleures affectations dans la section 1.3.2. En guise de conclusion, la section 1.4 présente les problèmes auxquels nous nous intéressons et les outils que nous proposons pour les résoudre.

1.1 Définitions des problèmes d'optimisation

Nous nous sommes appuyés principalement sur trois ouvrages didactiques, à savoir Sakarovitch (1984), Ahuja et al. (1993), Gondran et Minoux (2009) et Teghem (2012) afin de réunir ces quelques éléments essentiels de la recherche opérationnelle. Loin d'être exhaustive, cette section a pour objectif d'introduire ou de rappeler les notions classiques utilisées dans cette thèse.

1.1.1 Problèmes d'optimisation

Un *problème d'optimisation*, noté $\mathcal{P}(X, f)$, est caractérisé par un *ensemble réalisable* ou *admissible* X non-vide et une *fonction objectif* f qui associe un scalaire dans $\mathbb{R}$ à chaque élément x de l'ensemble X. Les éléments de X sont dits *solutions réalisables*. Résoudre le problème $\mathcal{P}(X, f)$ revient à trouver parmi les solutions réalisables, une qui minimise ou maximise f, c'est-à-dire dans le cas d'un problème de minimisation, trouver une solution $x^* \in X$ telle que $f(x) \geq f(x^*)$ pour tout élément x dans X. Une telle solution est dite *optimale* et sera notée $x(X, f)$.

Nous nous contenterons de présenter les problèmes de minimisation ; les problèmes de maximisation obéissent aux mêmes règles, à quelques changements près. Nous ne considérons ici que les problèmes qui admettent au moins une solution optimale. Un problème d'optimisation peut avoir plusieurs fonctions objectif, il s'agit alors d'un problème d'*optimisation multicritère* ou *multiobjectif*. Ces problèmes seront abordés dans la section 1.1.4.

L'ensemble réalisable X est habituellement défini comme partie de $\mathbb{R}^n$ où n est un entier positif désignant la *taille du problème*. Les solutions réalisables peuvent alors être représentées comme des vecteurs dont les n composantes sont les *variables* du problème. L'ensemble X est délimité couramment par un système d'inégalités appelées *contraintes* du problème. Les contraintes sont construites à l'aide de combinaisons des variables, et permettent de caractériser les propriétés communes aux solutions de X afin de les distinguer parmi toutes les solutions de $\mathbb{R}^n$. La description de l'ensemble X est donc implicite. Les *programmes linéaires* sont sans doute les problèmes d'optimisation les plus connus. La fonction objectif et les contraintes de ces problèmes sont linéaires.

1.1.2 Problèmes d'optimisation discrète

Les problèmes d'*optimisation discrète*, par opposition à l'*optimisation continue*, forment une classe de problèmes d'optimisation particulièrement étudiée. Tout ou partie des variables de ce type de problèmes appartiennent à l'ensemble des entiers, autrement dit $X \subseteq \mathbb{Z}^m \times \mathbb{R}^{n-m}$, avec $0 \leq m \leq n$. Lorsque $m = n$, $\mathcal{P}(X, f)$ est dit problème en *nombres entiers*, sinon il fait partie des problèmes d'optimisation *mixtes en nombres entiers*. Bien que l'ensemble réalisable soit plus restreint dans le cas de l'optimisation discrète, ces problèmes sont souvent plus difficiles que leurs versions continues.

Les ensembles réalisables des problèmes d'optimisation et des problèmes mixtes peuvent être infinis, et ceux des problèmes en nombres entiers sont au plus dénombrables. Nous présentons dans la section 1.1.3 les problèmes auxquels nous nous intéressons, à savoir les problèmes d'*optimisation combinatoire*. Ces problèmes sont connus pour leur difficulté bien qu'ils soient définis sur des ensembles admissibles finis.

1.1.3 Problèmes d'optimisation combinatoire

Les problèmes d'optimisation combinatoire sont des problèmes d'optimisation dont les ensembles réalisables sont finis mais combinatoires. Aussi, le nombre de solutions réalisables des problèmes combinatoires augmente exponentiellement en fonction de la taille du problème, et c'est ce qui exclut des méthodes de résolution basées sur l'énumération de toutes les solutions réalisables.

Généralement l'ensemble admissible X d'un problème d'optimisation combinatoire est défini comme un sous-ensemble de l'ensemble des parties d'un ensemble fini d'éléments E, i.e. $E = \{e_1, \ldots, e_n\}$ et $X \subseteq 2^E$. Un problème d'optimisation combinatoire peut aussi être formalisé comme un problème d'optimisation en *variables binaires*. En associant une variable x_i à tout élément e_i de E, tout sous-ensemble de E peut être représenté par un vecteur $x = (x_1, \ldots, x_n)$, avec $x_i = 1$ si e_i appartient au sous-ensemble et $x_i = 0$ sinon.

Ce qui fait l'élégance des problèmes d'optimisation combinatoire, c'est qu'en plus des formulations en programmes mathématiques, il est souvent possible de les formuler comme des problèmes de la théorie des graphes. Nous rappelons donc ci-après quelques notions élémentaires de théorie des graphes, que nous utilisons dans cette thèse.

Rappel 1.1.a (Définitions élémentaires de la théorie des graphes)**.** Un *graphe simple orienté* $G = (V, E)$ est défini par un ensemble de *sommets* V et un ensemble d'*arcs* $E \subseteq V \times V$. Les arcs traduisent une relation binaire irréflexive $\mathcal{R}$ entre les sommets, i.e. il existe un arc entre deux sommets distincts v et w si et seulement si $(v\mathcal{R}w)$. Un arc (v, w) est défini par la donnée de ses *extrémités initiale* v et *finale* w. Le sommet v est appelé le *précédent* de w, et le sommet w est le *suivant* de v, ces sommets sont dits *voisins* ou *adjacents*. On dit aussi que l'arc (v, w) est *incident* aux sommets v et w. Des *valuations* sont habituellement associées aux arcs, mais il est possible que les valuations soient associées aux sommets.

Il est d'usage lorsque la relation $\mathcal{R}$ qui caractérise l'ensemble des arcs E est symétrique (i.e. $(v\mathcal{R}w) \Rightarrow (w\mathcal{R}v)$), de considérer que le graphe $G = (V, E)$ est *non-orienté*, où E est l'ensemble des *arêtes*.

Nous définissons ici les chaînes, les chemins, les cycles et les circuits qui représentent des *séquences d'arcs* qui tiennent compte ou non de l'orientation du graphe. Une *chaîne* $\mathcal{C}$ de longueur l est une séquence de l arcs, $\mathcal{C} = (e_1, e_2, \ldots, e_l)$ telle que chaque arc e_i (où $i = 2, \ldots, l-1$) possède une extrémité commune avec l'arc e_{i-1} ($e_i \neq e_{i-1}$) et avec l'arc e_{i+1} ($e_i \neq e_{i-1}$). Tout sommet qui n'est adjacent qu'à un seul sommet de la chaîne est dit *extrémité*. Un *chemin* est une chaîne dont tous les arcs ont la même orientation. Une chaîne qui ne possède pas d'extrémité est un *cycle*. Un chemin sans extrémité est un *circuit*. Un circuit peut aussi être considéré comme un cycle dont les arcs ont tous la même orientation.

La séquence $\mathcal{C}$ est dite *élémentaire* si elle passe une seule fois par chacun de ses sommets et elle est dite *hamiltonienne* si elle est élémentaire et parcours tous les sommets du graphe. La *valeur* (*poids*, *coût*, ...) d'un séquence d'arcs est la somme des valuations des arcs qui la composent.

Un graphe est dit *complet* si l'ensemble des arcs est maximal. Un graphe est *connexe* (respectivement *fortement connexe*) s'il existe une chaîne (respectivement un chemin) entre chaque couple de sommets.

Un graphe est dit *biparti* s'il existe une partition de son ensemble de sommets en deux sous-ensembles V et W tels qu'il n'existe pas d'arête ou arc ayant ses deux extrémités dans le même sous-ensemble de sommets, il est noté $G = (V, W; E)$. Un graphe est *biparti complet* si l'ensemble E est maximal.

Il est nécessaire de différencier la difficulté de résolution et la difficulté de formulation des problèmes. Nous avons fait référence à plusieurs reprises à des problèmes faciles ou difficiles et nous entendons par là, la *complexité liée à la résolution de ces problèmes*. La plupart des problèmes d'optimisation combinatoire sont connus pour être très difficiles bien qu'ils soient faciles à exprimer. Le meilleur exemple est sans doute celui du *problème de voyageur de commerce* **TSP** qui revient à rechercher un cycle hamiltonien de valeur minimale dans un graphe non-orienté valué. Une version asymétrique de ce problème **ATSP** cherche un circuit hamiltonien de valeur minimale dans un graphe orienté. En dépit de la simplicité de son énoncé, il n'en demeure pas moins que **TSP** est l'un des problèmes les plus difficiles de l'optimisation combinatoire. Nous ouvrons une parenthèse afin de rappeler quelques notions élémentaires de théorie de la complexité.

Rappel 1.1.b (Rudiments de théorie de la complexité)**.** Afin de définir rigoureusement la complexité, il aurait sans doute fallu présenter le formalisme des *machines de Turing* ou des *Random-access machines* et des différents modes d'encodage des données, mais notre objectif ici se limite à remettre la difficulté des problèmes dans le contexte de l'analyse des algorithmes.

Afin de mesurer l'efficacité d'un algorithme, il faut établir une relation entre le nombre d'*opérations élémentaires* réalisées par cet algorithme et la *taille des données* qui permettent d'encoder un exemple ou une *instance* du problème étudié.

Il existe plusieurs manières de coder une instance d'un problème. Il est préférable pour plus de simplicité de s'appuyer sur les paramètres de description du problème étudié. Aussi la taille du problème est souvent retenue pour représenter la taille des données d'une instance. Par ailleurs, une opération élémentaire est souvent définie comme une tâche qui se réalise en un temps constant par les calculateurs usuels. Les opérations arithmétiques ou logiques et les opérations informatiques telles que le stockage d'une donnée dans un

registre sont considérées comme des opérations élémentaires.

Un algorithme est dit de complexité $O(g(n))$ s'il se termine après un nombre d'opérations élémentaires ne dépassant pas *au pire cas* $c.g(n)$, avec c constant et n la taille d'une instance. On dira d'un algorithme de complexité $O(g(n))$ qu'il est efficace ou *polynomial* s'il existe un polynôme en n majorant $g(n)$ (i.e. il existe deux constantes $c, k \geq 0$ et $n_0 \in \mathbb{N}$, telles que pour tout $n \geq n_0$, $g(n) \leq c.n^k$).

La complexité d'un problème est la complexité du meilleur algorithme qui permet de le résoudre. Si cet algorithme est polynomial, le problème est dit facile, autrement le problème est difficile. Nous présentons ici la complexité en temps. Il existe aussi une mesure de la performance des algorithmes en termes d'espace mémoire dite complexité en *espace*.

Une classe des problèmes difficiles particulièrement étudiée est celle des problèmes *NP-difficiles*. Ces problèmes sont liés par une relation d'équivalence dans le sens où s'il existe un algorithme polynomial pour un des problèmes de cette classe alors tous les problèmes NP-difficiles pourront être résolus en un temps polynomial. Cette classe englobe les problèmes les plus étudiés de l'optimisation combinatoire, parmi lesquels les problèmes que nous étudions dans cette thèse.

On peut citer un ensemble restreint de problèmes d'optimisation combinatoire faciles. Cependant si des changements, même s'ils peuvent paraître insignifiants, affectent l'ensemble réalisable ou la nature de leur fonction objectif, il deviennent difficiles. Nous donnons dans les exemples suivants trois situations où des problèmes polynomiaux deviennent NP-difficiles suite à de légères modifications.

Exemple.

1. L'algorithme de Dijkstra (1959) permet de résoudre de manière efficace le *problème du plus court chemin* entre deux sommets dans un graphe orienté et valué par des coûts positifs. Il suffit d'inverser le sens de l'optimisation, c'est-à-dire chercher un chemin de valeur maximum, pour que ce problème devienne difficile (voir Garey et Johnson (1979)).

2. Un *arbre* est un graphe connexe sans cycles. La recherche d'un arbre de poids minimum qui couvre tous les sommets d'un graphe $G = (V, E)$ est un problème polynomial. Curieusement, si seulement un sous-ensemble des sommets du graphe devait être couvert par l'arbre minimum recherché (*arbre de Steiner*) alors le problème est NP-difficile.

3. Le problème d'affectation est connu pour être un problème polynomial si sa fonction objectif est linéaire, contrairement au problème d'affectation quadratique qui appartient à la classe des problèmes NP-difficiles.

Des modifications infimes sur un problème d'optimisation combinatoire peuvent induire une détérioration très importante de sa complexité. Nous nous intéressons à la

démarche inverse qui consiste à modifier un problème difficile pour en définir un nouveau, plus facile, qui sera utilisé afin de résoudre le problème original. Cette technique est connue sous le nom de *relaxation*. Nous la rappelons dans la section 1.2 dans laquelle nous rappelons aussi le schéma de résolution par séparation et évaluation ou le *Branch and Bound*.

1.1.4 Problèmes d'optimisation combinatoire multiobjectif

Nous définissons dans cette section le problème de recherche de solutions de bon compromis pour un problème d'optimisation combinatoire multiobjectif. Dans un premier temps nous rappelons les notions préliminaires de l'optimisation multiobjectif, ensuite nous précisons la notion de compromis et nous formalisons ces problèmes.

Un problème d'optimisation combinatoire multiobjectif peut s'écrire sous la forme du problème (1.1), où p représente le nombre d'objectifs et X un ensemble combinatoire.

$$\min_{x \in X} z(x) = (z_1(x), \ldots, z_p(x)) \tag{1.1}$$

Afin de ne pas confondre les solutions réalisables et leurs images dans $\mathbb{R}^p$, il est d'usage en optimisation multiobjectif de distinguer l'*espace des décisions* $\mathcal{X}$, qui contient l'ensemble admissible X, et l'*espace des objectifs* $\mathcal{Z} \subseteq \mathbb{R}^p$ qui lui contient l'image de l'ensemble admissible X dans $\mathbb{R}^p$. À toute solution réalisable x est associé un *vecteur critère* $z(x) = (z_1(x), \ldots, z_p(x))$ qui correspond à son image dans $\mathcal{Z}$, telle que z_k est une fonction objectif pour tout k dans $\{1, \ldots, p\}$. L'ensemble des points correspondant aux images des solutions admissibles dans $\mathcal{Z}$ est noté $Z = \{z(x) : x \in X\}$.

En raison de la nature conflictuelle des objectifs, il n'y a généralement pas une solution réalisable qui minimise simultanément tous les objectifs. L'optimalité dans un contexte multiobjectif est basée sur la notion de dominance et d'efficacité au sens de *Pareto* dont nous rappelons le formalisme dans les définitions 1 et 2.

Définition 1. Soient $z, z' \in \mathbb{R}^p$ deux vecteurs.

z *domine largement* z', noté $z \leqq z'$, si $z_k \leq z'_k$ pour tout k dans $\{1, \ldots, p\}$.

z *domine* z', noté $z \leq z'$, si $z_k \leq z'_k$ pour tout k dans $\{1, \ldots, p\}$ et $z \neq z'$.

z *domine strictement* z', noté $z < z'$, si $z_k < z'_k$ pour tout k dans $\{1, \ldots, p\}$.

Définition 2.

Un point z dans Z est (*faiblement*) *non-dominé* s'il n'existe pas un autre point z' dans Z tel que z' domine (strictement) z.

Une solution x dont l'image $z(x)$ dans Z est (*faiblement*) *non-dominé* est dite (*faiblement*) *efficace*.

On notera Z_N et Z_{WN} les ensembles de points non-dominés et faiblement non-dominés. Il est facile de voir que $Z_N \subset Z_{WN}$. Les ensembles des solutions efficaces et faiblement-efficaces seront notés respectivement X_E et X_{WE}.

1.1.4.a Solutions de bon compromis

Un point non-dominé est l'image d'une ou de plusieurs solutions réalisables, tel que la performance d'un objectif ne peut être améliorée sans détériorer au moins un autre objectif. Les solutions efficaces ne sont donc pas comparables entre elles et représentent de bons compromis potentiels. Cependant, un décideur peut vouloir exprimer des préférences qui lui sont propres. Ces préférences seraient alors de nature à privilégier un certain type de solutions, par exemple il est d'usage de préférer des solutions équilibrées, c'est-à-dire des solutions avec des valeurs moyennes sur les objectifs.

Le problème de recherche d'une solution de bon compromis revient généralement à agréger les critères à l'aide d'une fonction *scalarisante* f_γ, dite aussi fonction d'*agrégation*. L'indice γ de la fonction scalarisante fait référence aux paramètres qui modélisent les préférences du décideur. Une solution de meilleur compromis est donc obtenue en résolvant le problème (1.2).

$$\min_{x \in X} f_\gamma(z_1(x), \ldots, z_p(x)) \tag{1.2}$$

La recherche de solution de compromis trouve sa principale application dans les *méthodes interactives*. Une méthode interactive consiste en une alternance d'étapes de calculs et d'étapes de dialogue entre un analyste et un décideur. La première étape de calcul fournit une première solution optimale pour le problème (1.2). Cette solution est présentée au décideur, qui réagit en apportant des informations supplémentaires sur ses préférences que l'analyste transcrit généralement en modifiant les paramètres γ. Une nouvelle solution de compromis est calculée et présentée à nouveau au décideur. Le processus interactif s'arrête lorsque le décideur est ou satisfait de la solution proposée, ou bien fatigué. Plus de détails sur ces méthodes peuvent être consultés dans Vanderpooten (1989) ou Vanderpooten et Vincke (1989). Nous nous intéressons dans cette thèse à la phase de calcul des méthodes interactives.

Wierzbicki (1986b) énumère des conditions qui doivent être prises en considération lors du choix d'une fonction scalarisante, à savoir :

C1. toute solution optimale du problème (1.2) correspond à un point non-dominé,

C2. tout point non-dominé peut être obtenu en résolvant le problème (1.2),

C3. le problème (1.2) est raisonnablement facile.

Autrement dit, la condition **C1.** impose que la résolution du problème (1.2) conduise à une solution efficace pour n'importe quels paramètres admissibles γ, et assure qu'aucun

point dominé ne soit présenté au décideur. La condition **C2.** permet de garantir qu'aucun point non-dominé ne soit écarté, et que toutes les solutions efficaces soient candidates pour représenter le compromis. Enfin, on souhaite par la condition **C3.** choisir des fonctions d'agrégation f_γ telles que le problème (1.2) soit au plus aussi difficile que sa version linéaire mono-objectif. Ces conditions sont conflictuelles, néanmoins en faisant quelques concessions, des fonctions scalarisantes réalisant un équilibre entre ces exigences peuvent être identifiées.

Le même auteur classifie les fonctions scalarisantes dans des familles dont dépend la nature des solutions optimales du problème (1.2). Nous rappelons les deux classes de fonctions scalarisantes qui nous intéressent.

Théorème 1 (Wierzbicki (1986a,b))**.** Soit le problème (1.2) avec la fonction objectif $f_\gamma : Z \longrightarrow \mathbb{R}$.

1. Si f_γ est *fortement monotone*, i.e. pour $z, z' \in \mathbb{R}^p$, $z \leq z'$ implique $f_\gamma(z) < f_\gamma(z')$, alors toute solution optimale du problème (1.2) est efficace.
2. Si f_γ est *strictement monotone*, i.e. pour $z, z' \in \mathbb{R}^p$, $z < z'$ implique $f_\gamma(z) < f_\gamma(z')$, alors toute solution optimale du problème (1.2) est faiblement efficace.

L'utilisation de fonctions scalarisantes fortement monotones permet de satisfaire la condition **C1.**.

Somme pondérée La fonction scalarisante la plus intuitive est sans doute la *somme pondérée* $f_\lambda(z(x)) = \sum_{k=1}^{p} \lambda_k z_k(x)$. Cette fonction est fortement monotone lorsque le jeu de poids λ est strictement positif, i.e. $\lambda \in \mathbb{R}^p_>$. La condition **C1.** est donc satisfaite. Le problème (1.3) formalise la recherche de la solution de bon compromis lorsque celui-ci est modélisé par une somme pondérée, et est équivalent dans notre cas à la version mono-objectif linéaire du problème (1.1), ce qui satisfait la condition **C3.**.

$$\min f_\lambda(z(x)) : x \in X \tag{1.3}$$

Malheureusement la somme pondérée ne remplit pas l'exigence **C2.**. En effet, on peut distinguer deux types de points non-dominés : les points qui se trouvent sur la frontière de l'enveloppe convexe de Z, qui sont dits *supportés*, et les points *non-supportés*. Les points supportés correspondent aux solutions optimales du problème (1.3) pour au moins un jeu de poids $\lambda \in \mathbb{R}^p_>$, ce qui n'est pas le cas des non-supportés. Par conséquent, la condition **C2.** n'est pas satisfaite puisque toutes les solutions correspondant à des points non-supportés ne peuvent pas être optimales pour le problème (1.3), et cela pour tous les jeux de poids $\lambda \in \mathbb{R}^p_>$.

Fonction scalarisante de type point de référence La condition **C2.** est la plus contraignante pour le choix d'une fonction d'agrégation. Wierzbicki propose d'utiliser *les*

fonctions scalarisantes de type point de référence. La forme la plus courante et la plus simple de telles fonctions correspond à la fonction objectif du problème (1.4), avec $\lambda \in \mathbb{R}^p_>$ la *direction de recherche* et $\bar{z}$ le *point de référence*, les paramètres qui modélisent les préférences du décideur.

$$\begin{array}{ll} \min & f_{(\lambda,\bar{z})}(z(x)) = \max\limits_{k=1,\dots,p} \{\lambda_k(z_k(x) - \bar{z}_k)\} \\ \text{s.c.} & x \in X \end{array} \tag{1.4}$$

Géométriquement, si on représente les point réalisables du problème (1.4) dans l'espace des objectifs, résoudre ce problème revient à chercher une solution réalisable dont l'image est l'un des premiers points que touche le cône qui se déplace du point de référence $\bar{z}$ dans la direction définie par λ. Comme le montre la figure 1.1 les points z^1, z^2 et z^3 sont optimaux, avec z^1 et z^3 non-dominés et z^2 faiblement non-dominé.

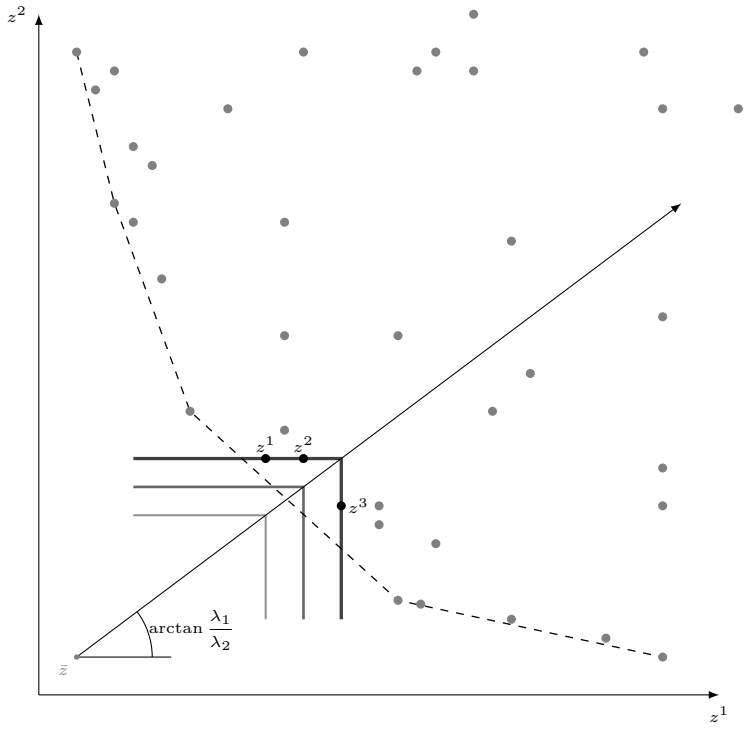


FIGURE 1.1 – Interprétation géométrique de la pseudo-distance de type point de référence

Un cas spécial de ces fonctions est *la distance de Tchebychev* (Bowman Jr, 1976), avec le point de référence fixé au *point idéal* z^*, qui correspond aux meilleures valeurs possibles pour chacun des objectifs, considéré indépendamment des autres, i.e. $z^* = (\min_{x\in X} z_1(x), \dots, \min_{x\in X} z_p(x))$.

Ces fonctions, dites aussi *pseudo-distances de type point de référence*, sont strictement monotones vérifient la condition **C2.**. Comme on peut le voir sur la figure 1.1, le cône peut

pénétrer à l'intérieur de l'enveloppe convexe des points réalisables. Toutefois, quelques concessions sur les conditions **C1.** et **C3.** sont nécessaires.

En effet, la condition **C1.** n'est pas réalisée. Les solutions optimales du problème (1.4) ont seulement la garantie d'être faiblement efficaces. Parmi les solutions optimales du problème (1.4), une au moins est efficace (voir Ehrgott (2005)). Aussi lors de la résolution du problème (1.4), il faut s'assurer que l'image de la solution optimale dans l'espace des objectifs est bien non-dominée.
La condition **C3.** est aussi violée, puisque le problème (1.4) est difficile même si le problème mono-objectif linéaire correspondant est polynomial. En effet, le problème (1.4) est une généralisation d'un problème d'optimisation combinatoire avec le critère *min max regret*. Ces problèmes ont été particulièrement étudiés par Kouvelis et Yu (1997) et Aissi et al. (2009) ; la quasi-totalité des problèmes classiques d'optimisation combinatoire avec le critère min max regret sont difficiles même si leurs versions linéaires sont polynomiales.

La somme d'une pseudo-distance et d'une somme pondérée donne la *pseudo-distance augmentée* (1.5).

$$f_{(\lambda,\bar{z},\rho)}(z(x)) = \max_{k=1,\dots,p} \{\lambda_k(z_k(x) - \bar{z}_k)\} + \rho \sum_{k=1}^{p} \lambda_k(z_k(x) - \bar{z}_k) \tag{1.5}$$

avec $\rho > 0$ et suffisamment petit.

La fonction $f_{(\lambda,\bar{z},\rho)}$ est fortement monotone, donc elle garantit une solution optimale efficace. De plus, la fonction $f_{(\lambda,\bar{z},\rho)}$ ne semble pas plus difficile à optimiser que la pseudo-distance $f_{(\lambda,\bar{z})}$. Cependant, nous préférons la fonction $f_{(\lambda,\bar{z})}$ à la distance augmentée $f_{(\lambda,\bar{z},\rho)}$, et ce pour deux raisons. La première est liée à la difficulté du choix du paramètre ρ. En effet, le terme linéaire de la fonction $f_{(\lambda,\bar{z},\rho)}$ s'interprète géométriquement par le remplacement du cône en angle droit de la figure 1.1 par un cône obtus. Plus le paramètre ρ est grand, plus ce cône est ouvert. C'est grâce à ce principe que le cône atteint le point non-dominé z^1 avant le point faiblement non-dominé z^2 dans l'exemple de la figure 1.1. Le choix du paramètre ρ est donc crucial. Une quantité ρ trop grande entraîne le risque de ne pas atteindre les solutions optimales du problème (1.4), i.e. les solutions optimales du problème (1.5) ne le sont pas pour le problème (1.4). À l'inverse, quand ρ est trop petit nous avons constaté lors de la mise en œuvre informatique que des solutions faiblement efficaces sont retournées, et ce vraisemblablement à cause de problèmes liés aux erreurs d'arrondis dans les calculs. Nous discuterons de ces points dans la section 3.4. Dächert et al. (2012) consacrent un article pour l'étude du choix du paramètre ρ pour le cas des problèmes discrets avec deux objectifs. La seconde raison pour laquelle nous utilisons les pseudo-distances strictement monotones, définies par problème (1.4), est liée au fait que les approches que nous proposons permettent de garantir l'efficacité de la solution optimale que nous retournons, ainsi l'introduction de l'augmentation n'est plus nécessaire.

D'autres fonctions scalarisantes strictement monotones ont été proposées dans la littérature, nous citons notamment Luque et al. (2012), Miettinen et Mäkelä (2002),

Nikulin et al. (2012) et Ruiz et al. (2008).

1.1.4.b Approche de résolution

Le problème (1.4) est non-linéaire, mais il peut s'écrire sous la forme du programme linéaire mixte en nombres entiers (1.6).

$$\begin{array}{ll} \min & \mu \\ \text{s.c.} & \mu \geq \lambda_k(z_k(x) - \bar{z}_k) \quad k = 1, \ldots, p \\ & x \in X \end{array} \tag{1.6}$$

Le problème (1.6) peut être résolu à l'aide des méthodes génériques ou des outils existants pour les programmes linéaires mixtes en nombres entiers tels que les *solveurs*. Cependant il faut s'assurer que la solution optimale est bien efficace. Généralement cela se fait par la résolution d'un ou de plusieurs nouveaux problèmes d'optimisation, ce qui n'est pas toujours facile.

Afin de contourner cette difficulté, il est possible de procéder en *deux phases*, à savoir, générer explicitement toutes les solutions efficaces dans un premier temps, puis choisir la solution qui minimise la fonction d'agrégation parmi ces solutions efficaces. Cette approche présente une faiblesse majeure due au fait que la plupart des problèmes d'optimisation combinatoire multiobjectif souffrent d'*intraitabilité*. Cela signifie que le nombre de points non-dominés peut être exponentiel en fonction de la taille de l'instance du problème. Par conséquent, il serait au moins aussi difficile de lister toutes les solutions efficaces que de chercher directement la solution optimale parmi elles. De plus les méthodes de génération des points non-dominés proposées dans la littérature ne sont pas efficaces lorsque le problème est à plus de deux critères (i.e. $p > 2$). Nous discutons de ces méthodes dans le chapitre 3.

C'est pour cela que nous procédons par la recherche directe d'une solution optimale et efficace, à l'aide des algorithmes d'énumération ordonnée que nous présentons dans le chapitre 2. Nous montrons par les tests expérimentaux du chapitre 3 que ces algorithmes sont plus performants que les solveurs, et nous montrons qu'il est possible de pallier les inconvénients des pseudo-distances (non-respect de la condition **C1.**) en garantissant l'efficacité de la solution optimale que nous retournons.

Le problème de la recherche de solutions de compromis pour les problèmes d'optimisation combinatoire multiobjectif a été étudié dans la littérature pour différents problèmes et différentes fonctions scalarisantes. Nous citons par exemple Paixão et al. (2003) pour *la distance euclidienne*, Hamacher et Ruhe (1994) ou Ehrgott et Skriver (2003) pour l'opérateur max, Galand et Perny (2006) pour la norme de Tchebychev et Galand et al. (2010) pour *l'intégrale de Choquet*.

1.2 Schéma général de résolution par Branch and Bound

Certaines propriétés des problèmes d'optimisation combinatoire donnent des indications quant aux méthodes à mettre en œuvre pour les résoudre. Ainsi, par exemple, la structure de matroïde de certains problèmes, comme l'arbre couvrant de poids minimum, conduit à envisager des stratégies gloutonnes. Citons également les problèmes, tels que le problème du plus court chemin, pour lesquels le principe d'optimalité de Bellmann s'applique ("dans une séquence optimale, toute sous-séquence est optimale") qui conduisent à mettre en œuvre des approches par programmation dynamique.

Nous nous intéressons particulièrement aux méthodes par *séparation et évaluation*, dites aussi *Branch and Bound* , qui sont basées sur l'idée de l'énumération des solutions réalisables. L'avantage certain de ces méthodes est qu'elles sont exploitables pour tout problème d'optimisation combinatoire puisqu'elles tirent leur validité du seul fait que l'ensemble admissible d'un problème est fini. Bien évidemment, le principe de ces méthodes n'est pas d'énumérer toutes les solutions réalisables mais de restreindre progressivement l'ensemble des solutions réalisables, et ce en alternant une étape dite de *séparation* et une étape d'*évaluation* jusqu'à obtention de la preuve de l'optimalité d'une solution. C'est la raison pour laquelle elles sont aussi appelées méthodes d'*énumération implicite*.

La séparation consiste en la division de l'ensemble des solutions réalisables en plusieurs sous-ensembles et l'évaluation repose sur la résolution des sous-problèmes engendrés par la séparation et à la suppression des sous-ensembles qui ne contiennent pas la solution optimale.

À proprement parler, le Branch and Bound est un schéma de résolution général, non un algorithme. Les algorithmes construits sur la base de ce schéma sont définis par la donnée des outils utilisés dans la séparation et l'évaluation, outils dont dépend la performance de ces algorithmes. Dans cette thèse, nous proposons des procédures basées sur le Branch and Bound afin de résoudre, le plus efficacement possible, des problèmes difficiles. Nous présentons brièvement les différentes étapes du Branch and Bound.

1.2.1 Séparation

Comme nous l'avons déjà évoqué, les méthodes de Branch and Bound reposent sur la réduction progressive de l'ensemble des solutions potentiellement optimales. Du fait de la difficulté des problèmes d'optimisation combinatoire, il est souvent plus facile d'aborder leur résolution en décomposant de manière itérative l'ensemble admissible en plusieurs sous-ensembles et ainsi définir des *sous-problèmes*. À chaque itération d'un algorithme de Branch and Bound, un sous-ensemble est choisi dans l'ensemble des parties de l'ensemble réalisable, ensuite cet ensemble est remplacé par un recouvrement. Notons qu'il est souhaitable d'utiliser des partitions afin d'éviter les redondances.

Les nouveaux sous-ensembles sont définis en compréhension. Généralement des con-

traintes sont ajoutées à la description de l'ensemble séparé. Par exemple, l'ensemble admissible d'un problème avec des variables binaires est divisé en imposant qu'une variable vaille 0 pour les solutions appartenant à un sous-ensemble et que la même variable soit égale à 1 pour les solutions d'un second sous-ensemble, ou bien en imposant et en interdisant une arête dans une formulation en graphe. On parle alors de séparation *binaire*. Il existe aussi d'autres types de séparations tels que le schéma de séparation de Murty-Lawler que nous présentons en détail dans la section 2.2.

Il est pratique de représenter l'évolution du Branch and Bound par une arborescence. L'ensemble réalisable est représenté par la racine de l'arborescence et les sous-ensembles non-séparés sont symbolisés par les nœuds feuilles de l'arborescence. Chaque nœud *père* est relié par un arc à chacun des nœuds *fils* qu'il a engendré. C'est vraisemblablement à cause de cette représentation que la séparation est aussi dite *branchement* et le Branch and Bound *méthode arborescente*.

Toutes les solutions intéressantes sont contenues par les feuilles de l'arborescence, par conséquent seuls ces nœuds présentent un intérêt dans les méthodes de Branch and Bound. Notons aussi qu'un nœud rassemble plusieurs informations liées à un sous-ensemble de solutions réalisables telles que les contraintes qui ont permis de le définir et la solution optimale du sous-problème qui lui est associé.

1.2.2 Évaluation

Sans l'étape d'évaluation, un algorithme basé sur le Branch and Bound engendrerait une arborescence dont chacune des feuilles contient une solution réalisable. Au lieu d'être implicite, l'énumération serait donc exhaustive.

L'évaluation permet de calculer une borne inférieure (dans le cas de la minimisation) sur la valeur de la solution optimale des sous-problèmes issus de la séparation et d'éliminer les nœuds qui ne peuvent contenir la solution optimale du problème original. Dans un premier temps, nous présentons la *relaxation*, qui est le moyen le plus utilisé pour calculer des bornes inférieures.

Définition 3 (Relaxation des problèmes d'optimisation -cas minimisation)**.** On désigne par *relaxation*, ou *problème relâché*, associée au problème d'optimisation $\mathcal{P}(X, f)$, un problème $\mathcal{P}(\hat{X}, \hat{f})$ tel que X est un sous-ensemble de $\hat{X}$ (i.e. $X \subseteq \hat{X}$) et $\hat{f}$ une *fonction minorante* de f sur X (i.e. $f(x) \geq \hat{f}(x)\ \forall x \in X$).

Le recours aux techniques de relaxation est très utile lors de la résolution de problèmes difficiles, par conséquent le problème $\mathcal{P}(\hat{X}, \hat{f})$ est supposé être plus facile que $\mathcal{P}(X, f)$.

En plus de son usage principal, qui est le calcul de bornes inférieures, la relaxation est aussi utilisée dans la séparation. Un sous-ensemble de la partition de l'ensemble admissible X du problème principal est défini alors par $\hat{X}(v) \cap X$, tel que $\hat{X}(v)$ est le sous-ensemble de solutions de l'ensemble admissible $\hat{X}$ du problème relâché $\mathcal{P}(\hat{X}, \hat{f})$ associé au nœud v.

La relaxation dite *continue* est certainement la plus utilisée. Elle consiste en la transformation d'un problème d'optimisation discrète en un problème d'optimisation continue, notamment la relaxation d'un programme linéaire en nombres entiers (appelé aussi **PLNE**) connu difficile en un programme linéaire continu polynomial par la suppression des contraintes d'*intégrité*.

La *relaxation lagrangienne* est basée sur la suppression de contraintes considérées difficiles. Soulagé de ces contraintes, l'ensemble $\hat{X}$ contient plus de solutions que X. Les contraintes supprimées sont ensuite traduites et introduites dans la fonction objectif afin de pénaliser les solutions de $\hat{X} \setminus X$.

Dans cette thèse, nous relâchons des problèmes d'optimisation combinatoire en leur associant d'autres problèmes d'optimisation combinatoire plus faciles afin de les résoudre par des algorithmes de Branch and Bound. En effet, il est toujours possible d'associer une telle relaxation à un problème d'optimisation combinatoire en enrichissant son ensemble réalisable par la suppression de certaines contraintes et/ou par la minoration de sa fonction objectif.

Comme nous l'avons mentionné précédemment, l'évaluation se compose de deux principes complémentaires qui sont le calcul de bornes et la suppression des nœuds inintéressants, dans le sens où ils ne contiennent pas de solution optimale.

Par définition, la résolution d'une relaxation permet de produire une borne inférieure pour la valeur de la meilleure solution d'un nœud. Il est souvent possible de produire des bornes inférieures par d'autres mécanismes liés aux propriétés de certains problèmes. À force de séparation, les sous-ensembles obtenus deviennent de plus en plus restreints, aussi une solution optimale pour certains sous-problèmes peut être obtenue en temps polynomial. Cette situation se produit par exemple lorsqu'un sous-ensemble ne contient qu'une seule solution réalisable, ou lorsque la solution optimale de la relaxation est réalisable pour le problème principal.

Pendant la progression d'un algorithme de Branch and Bound, il est d'usage de stocker la meilleure solution réalisable rencontrée du problème original $\mathcal{P}(X, f)$. Cette solution permet de définir une borne supérieure pour la valeur de la solution optimale. À la fin de la procédure, cette solution est retournée comme solution optimale, et peut être considérée, avant la fin de l'algorithme, comme une solution approchée du problème.

La seconde partie de l'évaluation, dite d'*élagage*, consiste en la suppression des nœuds qui ne contiennent pas la solution optimale et à arrêter la séparation des nœuds dont la solution optimale est connue. En effet, lorsque la borne inférieure associée à un nœud est supérieure à la valeur d'une solution réalisable connue, le sous-ensemble correspondant ne peut pas contenir la solution optimale, un tel nœud est *coupé*. Par ailleurs, un nœud dont la solution optimale est connue ne nécessite aucun développement supplémentaire, et peut donc être *stérilisé*, dans le sens où seule sa solution optimale est retenue.

1.3 Problème d'affectation et algorithmes de k-best

Le problème d'affectation joue un rôle important dans cette thèse, puisque toutes les applications que nous proposons dans les chapitres suivants reposent sur l'énumération de solutions du problème d'affectation. C'est la raison pour laquelle nous dédions cette section au rappel de quelques définitions autour de ce problème. Nous rappelons différentes formulations du problème d'affectation dans la section 1.3.1 et quelques-uns des algorithmes les plus connus pour le résoudre, puis nous rappelons les deux schémas principaux des algorithmes de k-meilleures affectations dans la section 1.3.2.

Nous nous appuyons dans cette section essentiellement sur l'ouvrage spécialisé dans le problème d'affectation de Burkard et al. (2009).

1.3.1 Problème d'affectation

1.3.1.a Formulation

Nous entendons par problème d'affectation le problème qui consiste à associer chaque élément d'un ensemble de n items à un seul élément d'un autre ensemble de n items avec un coût minimal.

Il existe plusieurs formulations du problème d'affectation parmi lesquelles nous en retenons trois qui présentent un intérêt dans le contexte de notre travail. La première est la formulation par la programmation linéaire (1.7).

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & & \\
\text{s.c.} \quad & \sum_{j=1}^{n} x_{ij} = 1 & i = 1,\dots,n & \quad (1.7.a)\\
& \sum_{i=1}^{n} x_{ij} = 1 & j = 1,\dots,n & \quad (1.7.b)\\
& x_{ij} \geq 0 & i,j = 1,\dots,n &
\end{aligned}$$

Une solution réalisable pour le problème d'affectation dans ce cas est un vecteur ou une matrice carrée de n^2 éléments, où :
$x_{ij} = \begin{cases} 1 & \text{si } i \text{ est associé à } j \\ 0 & \text{sinon} \end{cases}$ pour $i,j = 1,\dots,n$.

Cette formulation est très intéressante du fait qu'elle montre une particularité bien connue du problème d'affectation. En effet, bien que les solutions du problème d'affectation soient entières, celui-ci peut être énoncé sous la forme d'un programme linéaire continu, et cela grâce au fait que la matrice du programme linéaire primal (1.7) est *totalement unimodulaire* (voir par exemple Burkard et al. (2009)). Le problème d'affectation

est donc dit à *saut de dualité nul* et le problème dual qui lui est associé s'écrit sous la forme du programme linéaire (1.8).

$$\begin{array}{lll} \max & \displaystyle\sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j & \\ \text{s.c.} & u_i + v_j \leq c_{ij} & i,j = 1,\ldots,n \end{array} \tag{1.8}$$

Par le théorème des *écarts complémentaires*, les solutions x, u et v du primal et du dual sont optimales si et seulement si $x_{ij}(c_{ij} - u_i - v_j) = 0$ pour $i,j = 1,\ldots,n$. Les quantités $\bar{c}_{ij} = c_{ij} - u_i - v_j$ pour $i,j = 1,\ldots,n$ sont les *coûts réduits* et sont nécessairement positifs lorsque la solution est dual-réalisable.

La deuxième formulation du problème d'affectation qui nous intéresse est définie en théorie des graphes comme *le problème du couplage parfait de coût minimum dans un graphe biparti complet*. Un *couplage* dans un graphe non orienté est un ensemble d'arêtes qui n'ont pas de sommets en commun. Un couplage est dit *parfait* si chaque sommet du graphe est une extrémité d'une des arêtes du couplage. Le problème d'affectation peut donc être considéré comme le problème du couplage parfait de coût minimum dans un graphe biparti complet $G = (U \cup V, E)$ avec $|U| = |V| = n$. Chaque arête $(i,j) \in E$ est munie de la valuation c_{ij}. La figure 1.2 représente une instance du problème d'affectation formulée comme un problème de couplage de coût minimum.

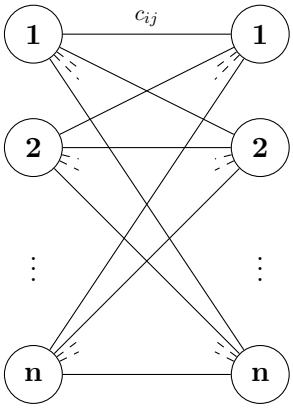


FIGURE 1.2 – Graphe biparti complet

Une solution réalisable pour cette formulation peut être représentée par l'ensemble des n arêtes qui forment le couplage. Elle sera notée $E(x) = \{\{i_1, j_{i_1}\},\ldots,\{i_n, j_{i_n}\}\}$, lorsque j_{i_k} est l'élément de V affecté à l'élément i_k de U, pour $k = 1,\ldots,n$.

Finalement, la formulation en *problème de flot maximum à coût minimum* est un cas général du problème d'affectation. Une grande partie des approches de résolution et des résultats du problème d'affectation sont à l'origine développée pour les problèmes de flot. Ce problème consiste à faire circuler dans un graphe de la matière, à partir d'un sommet

source s vers un sommet puits t, à travers les arcs d'un graphe orienté munis à la fois de capacités q_{ij} et de coûts unitaires c_{ij}. Un flot réalisable respecte les contraintes de capacité sur les arcs et la loi de *conservation de la matière*, à savoir pour chaque sommet du graphe différent de la source s et du puits t, la somme des flux des arcs entrants est égale à la somme des flux sur les arcs sortants du sommet. Le problème d'affectation peut être formulé comme un problème de flot maximum à coût minimum comme le montre la figure 1.3. Les capacités des arcs sortant de s et ceux entrant en t sont d'une unité et les coûts qui leur sont associés sont nuls. Les autres arcs sont non contraints et sont munis des coûts c_{ij} du problème d'affectation.

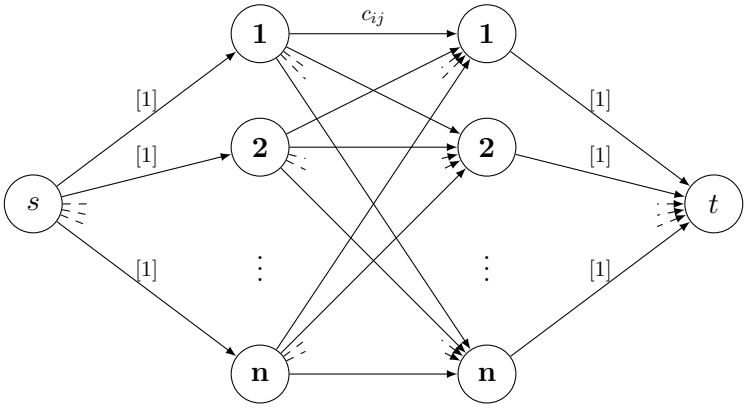


FIGURE 1.3 – Problème d'affectation sous la forme du problème de flot maximum à coût minimum

Comme pour le problème d'affectation, la matrice des contraintes de la formulation en programme linéaire du problème de flot est totalement unimodulaire, donc l'intégralité des solutions est garantie. Par conséquent, les contraintes de capacité sur les arcs vont imposer des valeurs bivalentes pour les valeurs des flux. Il est évident que les arcs non-adjacents à s ou à t et dont le flux vaut 1 correspondent aux arêtes d'une affectation ou d'un couplage.

1.3.1.b Quelques algorithmes pour le problème d'affectation

Burkard et al. (2009) présentent une comparaison minutieuse des différentes méthodes de résolution du problème d'affectation, allant du premier algorithme exponentiel proposé pour ce problème jusqu'aux dernières implémentations en programmation parallèle. S'il fallait faire une classification vulgarisée de la plus grande partie de ces méthodes, nous pouvons distinguer deux types d'approches. Une première classe de méthodes vise à construire une solution réalisable puis à l'améliorer progressivement jusqu'à l'obtention d'une preuve d'optimalité. La seconde classe de méthodes est constituée de celles qui construisent une *solution partielle* et qui l'augmentent pas à pas, jusqu'à l'affectation de tous les éléments de l'ensemble U à ceux de l'ensemble V.

Notre objectif est d'énumérer de manière efficace des solutions réalisables du problème d'affectation ; le second type de méthodes convient mieux que le premier à cet exercice. Nous rappelons brièvement l'*algorithme hongrois* et plus précisément son implémentation à l'aide de la recherche de plus courts chemins car cela permet de réduire la complexité de l'énumération ordonnée.

Dans toutes les méthodes par construction progressive d'une solution optimale pour le problème d'affectation, il convient dans un premier temps d'affecter de manière optimale quelques objets, avec un minimum d'effort de calcul. Cette étape vise à construire une *solution partielle*. Une telle solution, qui a souvent moins de n objets affectés, est non-réalisable mais correspond à une solution *duale-réalisable*. Cette étape n'est pas nécessaire pour la validité de l'algorithme hongrois, mais elle permet d'améliorer ses performances.

L'algorithme hongrois à l'aide des plus courts chemins se base essentiellement sur la formulation du problème d'affectation en problème de flot et sur l'algorithme par les *plus courts chemins successifs* pour le problème de flot maximum à coût minimum (voir Roy (1970) ou Ahuja et al. (1993)), avec quelques simplifications rendues possibles en raison de la spécificité du problème d'affectation. Un *graphe d'écart* $G = (U \cup V; E_d \cup E_a)$ est associé à une solution partielle $\tilde{x}$, avec E_d l'ensemble des arc directs tel que $E_d = \{(i,j) \in U \times V : \tilde{x}_{ij} = 0\}$ et l'ensemble des arcs arrières $E_a = \{(j,i) \in V \times U : \tilde{x}_{ij} = 1\}$. La valuation c_{ij} est associée aux arcs directs $(i,j) \in E_d$, et la valuation $-c_{ij}$ est associée aux arcs arrières $(j,i) \in E_a$.

À chaque itération de l'algorithme hongrois, on construit un graphe d'écart puis on cherche un plus court chemin dans ce graphe allant d'un sommet non-affecté de l'ensemble U vers un sommet non affecté de l'ensemble V. Un tel chemin permet d'augmenter la taille de la solution partielle d'une arête, en introduisant les arêtes correspondant aux arcs directs dans la solution partielle et en retirant les arêtes des arcs inverses. La nouvelle solution partielle ainsi obtenue est duale-réalisable et le graphe d'écart qui lui est associé ne contient pas de circuits négatifs. L'algorithme s'arrête une fois que les n sommets de U sont affectés aux n sommets de V.

La complexité de cet algorithme est en $O(n^4)$. Elle correspond à n recherches de plus court chemin dans un graphe contenant des valuations négatives et sans circuits négatifs. La recherche d'un tel plus court chemin s'effectue avec une complexité en $O(n^3)$ lorsque le graphe ne contient pas de circuit négatif.

La complexité de cet algorithme a pu être ramenée à $O(n^3)$ grâce aux travaux de Tomizawa (1971) et de Edmonds et Karp (1972), et ce en remplaçant les valuations du graphe d'écart par les coûts réduits positifs associés à la solution partielle courante. En effet, la recherche d'un plus court chemin dans un tel graphe muni de valuations positives peut se faire avec une complexité en $O(m)$, où m est le nombre des arcs, à l'aide de l'implémentation par les tas de l'algorithme de Dijkstra. Pour plus de précision dans l'implémentation de l'algorithme de plus court chemin, l'arité du tas, qu'on notera d, doit être ajustée à la densité du graphe, à savoir $d = \max\{2, \lceil m/2n \rceil\}$ (voir Ahuja et al. (1993)). Ici le nombre d'arcs est au plus de $m = n^2$, ce qui donne la complexité

de l'algorithme hongrois en $O(n^3)$ (n plus courts chemins dans le pire cas). Le coût de la nouvelle solution partielle est la somme du coût de l'ancienne et de la valeur du plus court chemin dans le graphe partiel.

La figure 1.4 illustre la construction d'un graphe d'écart (1.4b) à partir d'une solution partielle (1.4a) et la mise à jour de celle-ci (1.4d) grâce au plus court chemin en ligne double allant du sommet 3 au sommet 4 dans la sous-figure 1.4b.

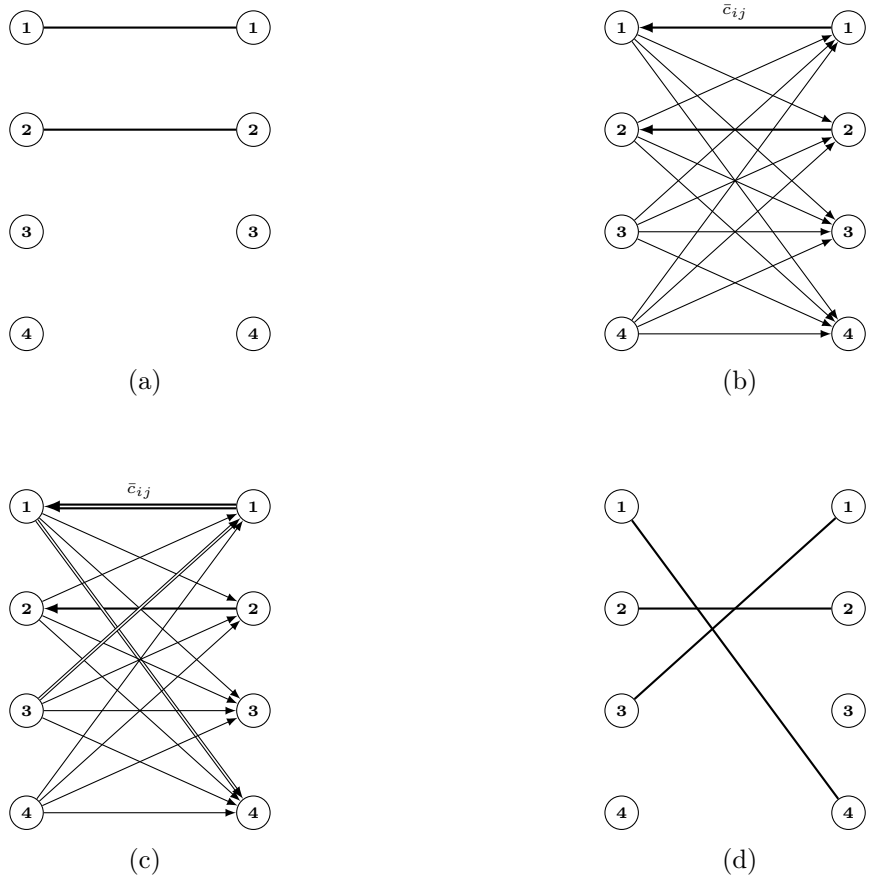


FIGURE 1.4 – Itération de l'algorithme hongrois avec graphe d'écart

1.3.2 Algorithmes de k-best pour le problème d'affectation

Tous les algorithmes de recherche des k meilleures solutions (k-best) pour le problème d'affectation sont basés sur des méthodes arborescentes, où à chaque nœud un problème d'affectation avec des contraintes supplémentaires est résolu (Murty (1968), Chegireddy et Hamacher (1987), Miller et al. (1997), Pascoal et al. (2003), Pedersen et al. (2008)). Pascoal et al. (2003) puis Pedersen et al. (2008) ont comparé ces algorithmes et il semble que le schéma de séparation proposé par Murty (1968) soit le plus efficace. Nous pré-

sentons dans la section 1.3.2.a quelques détails du mode de séparation de Murty-Lawler pour le problème d'affectation et nous discutons dans la section 1.3.2.b l'algorithme de k-best pour le problème d'affectation de Chegireddy-Hamacher.

1.3.2.a Algorithme de Murty

Nous décrivons dans la section 2.2 le schéma de séparation de Murty-Lawler généralisé aux problèmes d'optimisation combinatoire en {0,1}. Ce schéma a été proposé par Murty (1968) initialement pour le problème d'affectation. L'auteur tient compte des spécificités du problème d'affectation ce qui lui permet de simplifier la phase de séparation. En effet, imposer une variable $x_{i'j'}$ à valoir 1 force implicitement toutes les variables $x_{i'j}$ pour $j \neq j'$ et $j = 1, \ldots, n$, et toutes les variables $x_{ij'}$ pour $i \neq i'$ et $i = 1, \ldots, n$ à valoir 0, pour satisfaire les contraintes 1.7.a et 1.7.b du problème (1.7).

Formellement, à tout nœud v de l'arborescence est associé un sous-ensemble de solutions $X(v)$ défini à l'aide de deux ensembles disjoints d'arêtes $F(v)$ et $I(v)$ représentant respectivement l'ensemble des arêtes interdites et l'ensemble des arêtes imposées. Ainsi, les arêtes de l'ensemble $I(v)$ font partie des solutions réalisables du sous-ensemble $X(v)$, contrairement aux arêtes de l'ensemble $F(v)$ qui n'appartiennent pas à ces solutions. Tout nœud v de l'arborescence contient aussi la solution optimale $x(v)$ du sous-problème qui lui est associé.

Soit $E(x^k) = \{(i_1, j_{i_1}), \ldots, (i_n, j_{i_n})\}$ la meilleure solution du nœud v^k correspondant au sous-ensemble de solutions $X(v^k)$ contenant la $k^{\text{ème}}$ meilleure solution x^k. Soit $I(v^k) = \{(i_1, j_{i_1}), \ldots, (i_s, j_{i_s})\}$ l'ensemble des arêtes imposées pour le nœud v^k ; les arêtes $E(x^k) \setminus I(v^k) = \{(i_{s+1}, j_{i_{s+1}}), \ldots, (i_n, j_{i_n})\}$ sont dites *libres*. Les nœuds engendrés par la séparation du nœud v^k héritent de ses contraintes $I(v^k)$ et $F(v^k)$. Le branchement du nœud v^k consiste en la création de $n - s - 1$ nœuds v_l^k, où $l = 1, \ldots, n - s - 1$ tels que :

$$I(v_1^k) = I(v^k) \text{ et } F(v_1^k) = F(v^k) \cup \{(i_{s+1}, j_{i_{s+1}})\}$$

$$I(v_l^k) = I(v_{l-1}^k) \cup \{(i_{s+l-1}, j_{i_{s+l-1}})\} \text{ et } F(v_l^k) = F(v^k) \cup \{(i_{s+l}, j_{i_{s+l}})\}, \text{ pour } l = 2, \ldots, n - s - 1$$

La figure 1.5 montre le branchement par le mode de séparation de Murty-Lawler du nœud v^k de l'arborescence de recherche.

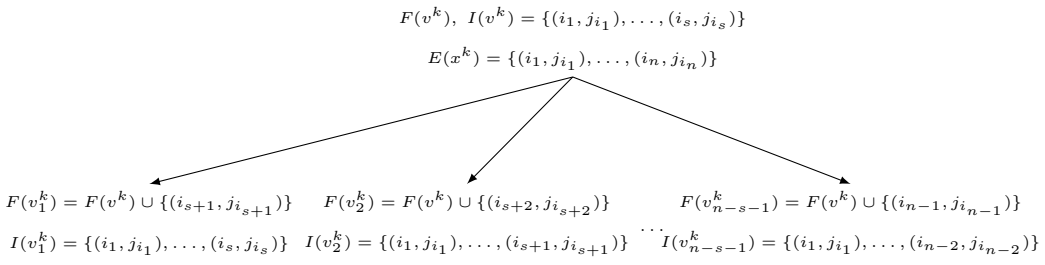


FIGURE 1.5 – Schéma de séparation de Murty-Lawler pour le problème d'affectation

Le mode de branchement de Murty-Lawler permet de diminuer la complexité de la recherche de la $k^{\text{ème}}$ meilleure solution de $O(n^4)$ à une complexité en $O(n^3)$. En effet, dans le pire cas $n-1$ problèmes d'affectation sont engendrés par cette séparation. Ces problèmes correspondent aux $n-1$ nouveaux nœuds. La résolution de chacun de ces nouveaux problèmes se ramène à la recherche d'un seul plus court chemin dans le graphe d'écart associé au problème. Pour ce faire, un graphe d'écart est construit sur la base de la solution optimale et des coûts réduits du nœud père v^k et des nouvelles contraintes du nœud fils. Comme on le montre dans la figure 1.6, la solution du père diminuée de la nouvelle arête interdite (arête en pointillés $(2,2)$ de la figure 1.6a) est une solution partielle du nouveau fils. Les arêtes imposées (arête en gras $(1,1)$ de la figure 1.6a) feront certainement partie de la solution optimale, aussi les extrémités des arêtes imposées et les arcs qui leur sont adjacents sont supprimés du graphe d'écart comme le montre la figure 1.6b.

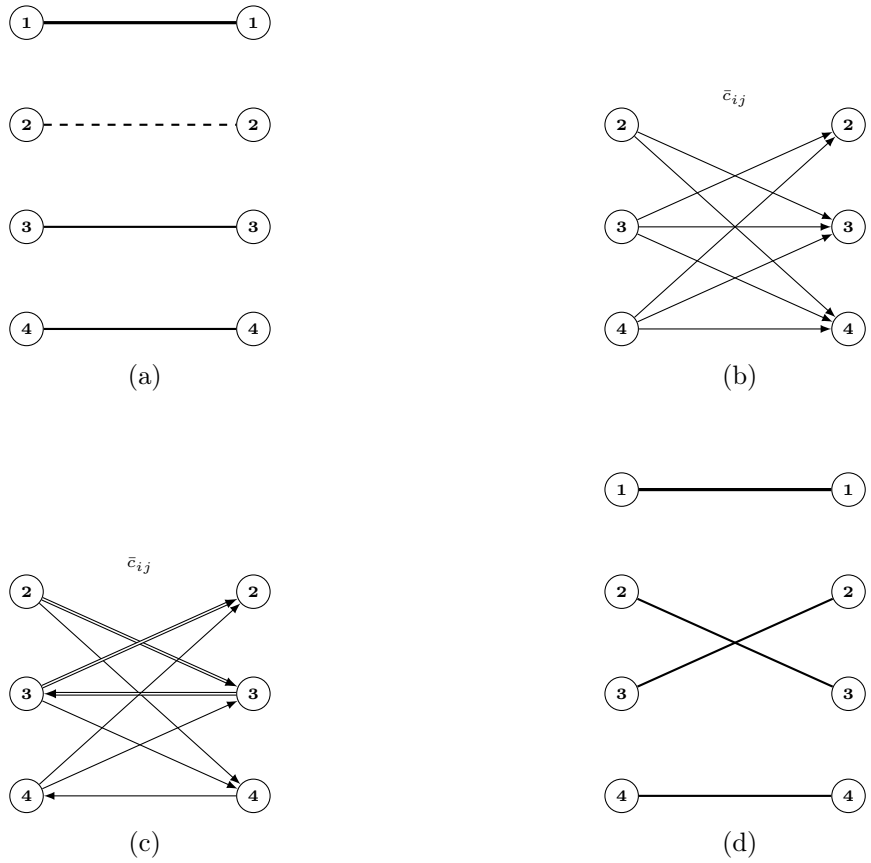


FIGURE 1.6 – Plus court chemin améliorant pour un nœud de l'arborescence de Murty-Lawler

Les extrémités de la nouvelle arête interdite sont donc les seuls sommets non affectés. Il suffit donc de trouver un seul chemin améliorant pour obtenir la solution optimale

avec une complexité en $O(n^2)$ comme le montrent les figures 1.6c et 1.6d. Comme pour l'algorithme hongrois, la valeur de la solution du fils est obtenue en sommant la valeur de la solution partielle, donc la valeur de la solution optimale du nœud père, et la valeur du plus court chemin dans le graphe d'écart valué par les coûts réduits. Finalement les variables duales sont modifiées afin de respecter la condition des écarts complémentaires. Ces améliorations ont été apportées aux algorithmes les plus efficaces de k-best pour le problème d'affectation. Pour plus de détails nous suggérons de consulter Pascoal et al. (2003), Miller et al. (1997) ou Pedersen et al. (2008).

Après la séparation du nœud v^k et la résolution des sous-problèmes associés aux nœuds fils, la $k+1^{\text{ème}}$ meilleure solution est la meilleure solution des nœuds feuilles de l'arborescence de recherche, et le nœud v^{k+1} qui contient cette solution sera à son tour séparé.

Améliorations de l'algorithme de Murty

Miller et al. (1997) font remarquer que plus on impose d'arêtes plus le sous-problème obtenu par la séparation de Murty-Lawler est petit, donc les nœuds créés en dernier (nœud fils à droite dans la figure 1.5) contiennent moins de solutions réalisables que les fils gauches. Les auteurs proposent de choisir la nouvelle arête interdite lors de la création d'un nouveau nœud, de façon à engendrer la plus grande détérioration sur la fonction objectif. Un tel choix permettrait de mettre les mauvaises solutions dans les problèmes de plus grande taille, et par conséquent les meilleures solutions tendraient à appartenir aux sous-ensembles de petite taille. Nous utilisons cette idée dans quelques-unes des variantes de nos algorithmes d'énumération ordonnée. Nous définirons plus en détail la façon d'évaluer la détérioration de la fonction objectif dans la section 3.2.3.

Exploitant la même remarque sur la taille des problèmes, Pascoal et al. (2003) proposent de résoudre les problèmes des fils droits avant ceux des fils gauches. De plus, ils proposent de considérer la solution optimale du nœud *frère*[1] comme solution partielle pour résoudre le sous-problème associé à un nœud nouvellement créé, et non la solution optimale du nœud père comme on l'a décrit précédemment.

1.3.2.b Algorithme de Chegireddy-Hamacher

Chegireddy et Hamacher (1987) proposent une approche alternative à la méthode de Murty qui est basée sur une séparation binaire qui nécessite le calcul de la première et de la seconde meilleure solution pour chaque nœud de l'arborescence. Comme pour l'algorithme de Murty un nœud v est défini par la donnée de l'ensemble des arêtes imposées $I(v)$ et de l'ensemble des arêtes interdites $F(v)$. Chaque nœud contient la meilleure solution $x(v)$ et la seconde meilleure solution $x'(v)$ du sous-problème qui lui est associé.

1. issu de la séparation du même nœud

Initialement, la première et la seconde meilleures solutions sont calculées et associées au nœud racine v^2 qui ne possède pas de contraintes supplémentaires (i.e. les ensembles $I(v^2)$ et $F(v^2)$ sont vides). Soit v^k le nœud dont la seconde meilleure solution $x'(v^k)$ est la $k^{\text{ème}}$ meilleure solution x^k du processus d'énumération. Une arête $e \notin I(v^k)$ est choisie dans $E(x(v^k)) \backslash E(x'(v^k))$ et deux nœuds fils v_e^k et $v_{\bar{e}}^k$ sont créés tels que : $(I(v_e^k) = I(v^k) \cup \{e\},\ F(v_e^k) = F(v^k))$ et $(I(v_{\bar{e}}^k) = I(v^k),\ F(v_{\bar{e}}^k) = F(v^k) \cup \{e\})$.

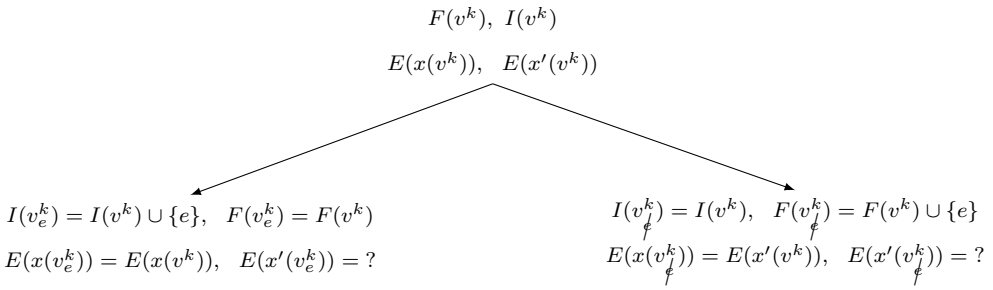


FIGURE 1.7 – Schéma de séparation de Chegireddy-Hamacher

La figure 1.7 montre la séparation d'un nœud v^k selon la méthode de Chegireddy-Hamacher. La meilleure solution du nœud fils v_e^k est la meilleure solution du nœud père v^k et la meilleure solution du nœud fils $v_{\bar{e}}^k$ est la seconde meilleure solution de son père v^k. Après le calcul de la seconde meilleure solution pour chacun des deux nouveaux nœuds fils $x'(v_e^k)$ et $x'(v_{\bar{e}}^k)$, la $k+1^{\text{ème}}$ meilleure solution est la meilleure solution parmi les secondes meilleures solutions des nœuds feuilles.

Les auteurs ont prouvé que le problème de la recherche de la deuxième meilleure solution pour le problème du couplage parfait dans un graphe biparti se ramène à la recherche d'un circuit de coût minimum dans un graphe d'écart similaire à celui décrit dans la section 1.3.2.a et la figure 1.6. La recherche d'un circuit de coût minimum peut se faire avec une complexité en $O(n^3)$. Techniquement, cela correspond à la recherche d'un plus court chemin entre les extrémités de tous les arcs inverses, ce qui correspond aux étapes de l'algorithme de Murty.

Les résultats expérimentaux obtenus par Pascoal et al. (2003) montrent que leur algorithme, qui rappelons-le est basé sur le schéma de Murty, est plus performant que leur implémentation de l'algorithme de Chegireddy-Hamacher. Cela est probablement dû au fait qu'une solution peut être calculée plusieurs fois dans l'algorithme de Chegireddy-Hamacher. En effet, l'algorithme de Chegireddy-Hamacher ne stocke pas les solutions obtenues par le calcul successif des plus courts chemins lors de la recherche du circuit de coût minimum, et c'est ce qui le rend moins performant que l'algorithme de Murty. C'est pour cette raison que nous écartons l'approche de Chegireddy-Hamacher dans nos algorithmes d'énumération ordonnée.

1.4 Contexte de la thèse

Ce chapitre était destiné à circonscrire les problèmes qui nous intéressent et les méthodes que nous envisageons pour les résoudre, à savoir les problèmes d'optimisation combinatoire et les méthodes de Branch and Bound. Nous nous intéressons particulièrement à des problèmes difficiles qui admettent des problèmes d'optimisation combinatoire faciles comme relaxations.

Nous décrirons dans le chapitre 2 une méthode générique pour la résolution des problèmes d'optimisation combinatoire. Ensuite, nous nous focaliserons sur certains problèmes d'optimisation combinatoire connus pour être difficiles auxquels il est possible d'associer le problème d'affectation comme relaxation. Les problèmes que nous nous proposons d'étudier peuvent être classés en trois catégories :

1. Les problèmes d'affectation avec une fonction objectif non-linéaire. La recherche d'une solution de compromis pour le problème d'affectation multiobjectif correspond à cette description. En effet, le recours à des techniques d'agrégation ramène les problèmes multiobjectifs à des problèmes mono-objectifs souvent non-linéaires. Le défi pour cette famille de problèmes est de minorer la fonction objectif non-linéaire par une fonction linéaire. De plus, des exigences liées à la nature multicritère du problème original conduisent au rejet de certaines solutions dites *dominées*. Nous présentons une procédure pour la recherche d'un bon compromis dans le chapitre 3.

2. Les problèmes avec fonction objectif linéaire et dont l'ensemble admissible est contenu dans l'ensemble réalisable du problème d'affectation. Nous proposons une procédure pour le problème de voyageur de commerce asymétrique dans le chapitre 4.

3. Les problèmes non-linéaires avec un ensemble réalisable plus restreint que celui du problème d'affectation. Nous citons cette famille de problèmes comme perspective. Les résultats que nous avons obtenus pour les deux premières familles sont prometteurs en vue d'une éventuelle application pour la recherche d'une solution de compromis pour le problème du voyageur de commerce asymétrique multiobjectif.

Les algorithmes de Branch and Bound ont été beaucoup étudiés, notamment dans le contexte de la résolution de problèmes d'optimisation combinatoire. Nous nous intéressons à une classe d'algorithmes de Branch and Bound fondés sur l'énumération ordonnée. Nous insérons des compléments à différents niveaux de la structure brute des algorithmes de Branch and Bound. Ces ajouts se présentent sous la forme de choix dans les détails de la séparation et des règles d'élagage. Les possibilités sont quasiment infinies quant au choix de ces compléments, aussi le défi dans l'implémentation des algorithmes de Branch and Bound est de trouver une combinaison raisonnable de ces perfectionnements afin d'améliorer les performances de l'algorithme et de résoudre des instances de plus en plus grandes.

Nous nous appuyons dans l'évaluation des performances de nos algorithmes sur l'analyse de la complexité théorique, mais nous accordons aussi un l'intérêt particulier aux études expérimentales. Les algorithmes que nous proposons sont mis à l'épreuve de l'implémentation informatique et à la résolution de plusieurs instances représentatives. Divers résultats sont analysés, parmi ceux-ci les temps de calcul, la taille des arborescences et le nombre de coupes résultant d'une règle d'élagage. Ces résultats sont interprétés, critiqués et comparés à des résultats connus s'ils sont disponibles, et ce n'est qu'après une analyse théorique et pratique que nous tirons des conclusions sur l'apport véritable d'une amélioration dans les performances d'un algorithme.

Chapitre 2
Principe général de l'énumération ordonnée

Résumé Dans ce chapitre nous présentons dans un premier temps un algorithme générique pour résoudre des problèmes d'optimisation combinatoire. Ensuite, deux variantes de cet algorithme basées sur le Branch and Bound sont discutées dans les sections 2.2 et 2.3. Enfin, nous présentons dans la section 2.4 des règles d'élagage de l'arborescence de Branch and Bound.

2.1 Principe général de résolution

Nous proposons dans cette section un algorithme pour résoudre des problèmes d'optimisation combinatoire basé sur l'énumération des solutions d'une relaxation. Cet algorithme découle de la proposition 1 qui décrit une condition suffisante d'optimalité pour un problème de minimisation.

Proposition 1 (Condition suffisante d'optimalité)**.** Soit le problème de minimisation $\mathcal{P}(X, f)$ et soit $\mathcal{P}(\hat{X}, \hat{f})$ un problème relâché associé à $\mathcal{P}(X, f)$. Soit $\hat{Y}$ un sous-ensemble de $\hat{X}$ et soit $x^* \in \arg \min_{x \in (X \cap \hat{Y})} f(x)$.

Si l'ensemble $\hat{Y}$ satisfait les conditions suivantes :

$$\forall x \in \hat{X} \setminus \hat{Y}, \forall y \in \hat{Y} : \hat{f}(x) \geq \hat{f}(y) \tag{2.1}$$

$$\exists y \in \hat{Y} : \hat{f}(y) \geq f(x^*) \tag{2.2}$$

alors x^* est une solution optimale pour le problème $\mathcal{P}(X, f)$.

Preuve. D'une part, la solution x^* est définie de telle sorte que :

$$\forall x \in X \cap \hat{Y} : f(x) \geq f(x^*) \tag{2.3}$$

D'autre part, la définition de la relaxation : $\forall x \in X : f(x) \geq \hat{f}(x)$, et l'inégalité (2.1) donnent :

$$\forall x \in X \cap (\hat{X} \setminus \hat{Y}), \forall y \in \hat{Y} : f(x) \geq \hat{f}(y) \tag{2.4}$$

Or, puisque $X \subseteq \hat{X}$ on a $X \cap (\hat{X} \setminus \hat{Y}) = X \setminus \hat{Y}$. L'inégalité (2.4) est donc équivalente à l'inégalité (2.5) :

$$\forall x \in X \setminus \hat{Y}, \forall y \in \hat{Y} : f(x) \geq \hat{f}(y) \tag{2.5}$$

D'après les inégalités (2.2) et (2.5), il vient :

$$\forall x \in X \setminus \hat{Y} : f(x) \geq f(x^*) \tag{2.6}$$

Les inégalités (2.3) et (2.6) impliquent que $\forall x \in X : f(x) \geq f(x^*)$, garantissant ainsi l'optimalité de la solution x^*. □

Exploiter la proposition 1 afin d'élaborer un mécanisme de résolution se heurte à la difficulté de trouver un ensemble $\hat{Y}$ qui satisfait les conditions (2.1) et (2.2). Néanmoins, on s'intéresse à des problèmes d'optimisation combinatoire définis sur des ensembles finis. Un problème d'optimisation combinatoire relâché $\mathcal{P}(\hat{X}, \hat{f})$ peut toujours être associé à $\mathcal{P}(X, f)$, avec $X \subseteq \hat{X}$, et $\forall x \in X : f(x) \geq \hat{f}(x)$.

Dans ce contexte, un ensemble $\hat{Y}$ satisfaisant la condition (2.1) de la proposition 1 peut être construit de manière itérative, à l'aide des k meilleures solutions du problème $\mathcal{P}(\hat{X}, \hat{f})$. En effet, ces solutions seraient énumérées dans un ordre non décroissant suivant la fonction $\hat{f}$ et ajoutées progressivement dans l'ensemble $\hat{Y}$. Notons la $k^{\text{ème}}$ meilleure solution x^k et remarquons qu'à ce stade x^k a la plus grande valeur selon la fonction $\hat{f}$ parmi les solutions de l'ensemble $\hat{Y}$, i.e. $\max_{y \in \hat{Y}} \hat{f}(y) = \hat{f}(x^k)$. Par ailleurs, la meilleure solution courante x^* est initialisée avec une solution quelconque x^0 de X, et mise à jour à chaque fois qu'une nouvelle solution de l'ensemble X est énumérée. La solution x^0 peut être obtenue par une heuristique pour le problème $\mathcal{P}(X, f)$. Ces étapes sont répétées jusqu'à l'obtention de la condition (2.2) ou bien jusqu'à l'énumération de toutes les solutions de $\hat{X}$.

Sur la base de la proposition 1 nous proposons une méthode générique, dont les étapes principales sont résumées dans l'algorithme 2.1, pour résoudre un problème d'optimisation combinatoire $\mathcal{P}(X, f)$.

Algorithme 2.1 : Résolution de $\mathcal{P}(X, f)$ par énumération ordonnée

Données : $\mathcal{P}(X, f), \mathcal{P}(\hat{X}, \hat{f}), x^0$
$x^* \leftarrow x^0$, $k \leftarrow 0$, $\hat{Y} \leftarrow \emptyset$
répéter
  $k \leftarrow k + 1$
  Déterminer $x^k \in \arg\min_{x \in \hat{X} \setminus \hat{Y}} \hat{f}(x)$ `// Résoudre` $\mathcal{P}(\hat{X} \setminus \hat{Y}, \hat{f})$
  $\hat{Y} \leftarrow \hat{Y} \cup \{x^k\}$
  si $(x^k \in X \ et \ f(x^k) < f(x^*))$ **alors** $x^* \leftarrow x^k$
jusqu'à $(\hat{f}(x^k) \geq f(x^*) \ ou \ \hat{X} \setminus \hat{Y} = \emptyset)$
retourner x^*

La figure 2.1 montre un exemple de l'algorithme 2.1. Les solutions de $\hat{X}$ énumérées sont représentées sur l'axe des abscisses. L'axe des ordonnées donne les valeurs des solutions énumérées, à la fois selon la fonction objectif de la relaxation $\hat{f}$ avec les points creux, et selon la fonction objectif du problème principal f avec les points pleins : noirs pour les solutions de X et gris pour les solutions réalisables uniquement pour le problème relâché $\mathcal{P}(\hat{X}, \hat{f})$, c'est-à-dire les solutions de $\hat{X} \setminus X$. La ligne continue en forme d'escalier ascendant qui suit les points creux représente l'évolution de la fonction $\hat{f}$ qui croît durant l'énumération. La valeur de la meilleure solution énumérée selon la fonction f est donnée par la ligne en tiretés qui décroît à chaque fois qu'une meilleure solution selon f de l'ensemble X est énumérée. Le processus s'arrête dès que ces deux courbes se croisent.

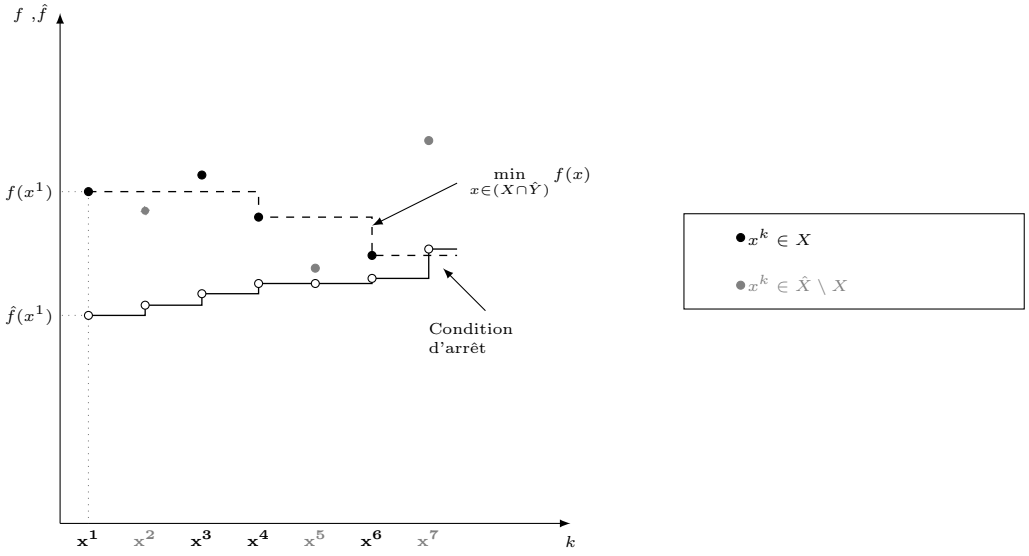


FIGURE 2.1 – Description d'un algorithme d'énumération ordonnée

L'efficacité de l'algorithme 2.1 dépend du choix de la relaxation à savoir, l'ensemble $\hat{X}$ et la fonction minorante $\hat{f}$. Si l'ensemble $\hat{X}$ est petit (i.e. $\hat{X} \backsimeq X$), alors le nombre

d'itérations au pire cas de l'algorithme 2.1 est forcément réduit. De même que si $\hat{f}$ est très proche de f, on peut raisonnablement s'attendre à ce que la condition d'arrêt soit obtenue rapidement. D'autre part, cet algorithme est proposé afin de résoudre des problèmes difficiles, par conséquent utiliser une relaxation trop proche de $\mathcal{P}(X, f)$ n'est pas une démarche pertinente. Un compromis est donc à trouver entre les approximations de X et f et la difficulté du problème relâché $\mathcal{P}(\hat{X}, \hat{f})$.

Le problème $\mathcal{P}(\hat{X} \setminus \hat{Y}, \hat{f})$ résolu à l'étape 4 de l'algorithme 2.1 n'est pas forcément trivial, même si le problème $\mathcal{P}(\hat{X}, \hat{f})$ est facile. C'est cette difficulté qui nous amène à utiliser des algorithmes de Branch and Bound.

Nous présentons dans les sections 2.2 et 2.3 deux algorithmes basés sur le même type de séparation mais qui divergent par leurs modes d'évaluation.

2.2 Énumération ordonnée et algorithmes de k-best

Les solutions x^k obtenues successivement en résolvant les problèmes $\mathcal{P}(\hat{X} \setminus \hat{Y}, \hat{f})$ et ajoutées dans l'ensemble $\hat{Y}$ sont ordonnées selon $\hat{f}$ dans un ordre non décroissant. En choisissant $\hat{f}$ linéaire et identique à chaque itération de l'algorithme 2.1, on peut tirer profit de la littérature existante sur les algorithmes de *k-best*. Murty (1968) a proposé un algorithme de k-best pour le problème d'affectation. Cet algorithme a ensuite été généralisé par Lawler (1972) pour les problèmes d'optimisation en variables binaires.

L'algorithme de Murty-Lawler est un Branch and Bound particulier puisque l'objectif visé n'est pas la recherche d'une solution optimale, mais de retourner les k meilleures solutions réalisables d'un problème d'optimisation combinatoire, rangées dans un ordre non décroissant.

Grâce à un mode de séparation original, les feuilles de l'arborescence de l'algorithme de Murty-Lawler contiennent des sous-ensembles de solutions réalisables qui forment une partition de l'ensemble $\hat{X} \setminus \hat{Y}$. On notera $\mathcal{L}$ l'ensemble des feuilles de l'arborescence.

Chaque nœud feuille $v \in \mathcal{L}$ contient le sous-ensemble de solutions réalisables $\hat{X}(v)$ définissant ainsi le sous-problème $\mathcal{P}(\hat{X}(v), \hat{f})$ qu'on notera aussi $\mathcal{P}(v)$. Un nœud v contient aussi la solution optimale du problème $\mathcal{P}(v)$ qui est notée indifféremment $x(\hat{X}(v), \hat{f})$ ou $x(v)$.

Initialement l'ensemble des feuilles $\mathcal{L}$ contient le nœud racine , qu'on notera v^r, tel que $\hat{X}(v^r)$ est l'ensemble de toutes les solutions réalisables du problème relâché, i.e. $\hat{X}(v^r) = \hat{X}$. La -première- meilleure solution est la solution optimale $x(v^r)$. De manière générale, lorsque les $k-1$ meilleures solutions sont connues et stockées dans l'ensemble $\hat{Y}$, la $k^{\text{ème}}$ meilleure solution x^k est obtenue parmi les solutions optimales des parties de l'ensemble $\hat{X} \setminus \hat{Y}$, i.e. $x^k \in \arg\min_{v \in \mathcal{L}} \hat{f}(x(v))$.

Notons v^{k-1} le nœud qui contient la $k-1^{\text{ème}}$ meilleure solution $x^{k-1} = (x_1^{k-1}, \ldots, x_n^{k-1})$. Une partition de $\hat{X}(v^{k-1})$ en $O(n)$ sous-ensembles est construite en ajoutant des contraintes à la description de $\hat{X}(v^{k-1})$. Sans perte de généralité, on peut supposer que

le sous-ensemble de solutions $\hat{X}(v^{k-1})$ est défini en fixant les s premières variables du problème $x_1, \ldots, x_s$ à 0 ou à 1, avec $s < n$. Notons $C^{te}(v^{k-1})$ l'ensemble des contraintes associées au nœud v^{k-1}. Les nœuds v_l^{k-1}, avec $l = 1, \ldots, n-s$, engendrés par la séparation du nœud v^{k-1} héritent de ses contraintes, i.e. les variables $x_1, \ldots, x_s$ sont fixées respectivement aux valeurs $x_1^{k-1}, \ldots, x_s^{k-1}$. Les $n-s$ nouveaux nœuds sont définis en fixant les variables $x_{s+1}, \ldots, x_n$ comme suit :

$$
\begin{array}{lcll}
C^{te}(v_1^{k-1}) & : & C^{te}(v^k) \cup \{x_{s+1} = 1 - x_{s+1}^k\} & \\
C^{te}(v_2^{k-1}) & : & C^{te}(v^k) \cup \{x_{s+2} = 1 - x_{s+2}^k & ; x_{s+1} = x_{s+1}^k\} \\
C^{te}(v_3^{k-1}) & : & C^{te}(v^k) \cup \{x_{s+3} = 1 - x_{s+3}^k & ; x_{s+1} = x_{s+1}^k, x_{s+2} = x_{s+2}^k\} \\
\vdots & & \vdots & \vdots \\
C^{te}(v_{n-s}^{k-1}) & : & C^{te}(v^k) \cup \{x_n = 1 - x_n^k & ; x_{s+1} = x_{s+1}^k, \ldots, x_{n-1} = x_{n-1}^k\}
\end{array}
$$

Les sous-ensembles $X(v_1^{k-1}), \ldots, X(v_{n-s}^{k-1})$ et le singleton $\{x^{k-1}\}$ forment une partition de $X(v^{k-1})$, et c'est là la particularité de la séparation de Murty-Lawler, à savoir, la $k-1^{\text{ème}}$ meilleure solution peut être isolée dans un singleton de la partition de $X(v^{k-1})$. Ainsi en remplaçant le nœud v^{k-1} par ses fils $v_1^{k-1}, \ldots, v_{n-s}^{k-1}$ dans $\mathcal{L}$ et en insérant la $k-1^{\text{ème}}$ meilleure solution x^{k-1} dans $\hat{Y}$, l'ensemble des nœuds $\mathcal{L}$ engendre toujours une partition de $\hat{X} \setminus \hat{Y}$. La difficulté des sous-problèmes $\mathcal{P}(\hat{X}(v_l^{k-1}), \hat{f})$ obtenus n'est pas accrue grâce à la nature des contraintes ajoutées lors de la séparation.

Dans les description des algorithmes qui suivent, nous désignons par **ML** le schéma de Murty-Lawler que nous venons de décrire.

Nous présentons dans l'algorithme 2.2 les opérations réalisées à l'étape 4 de l'algorithme 2.1 pour la recherche d'une $k^{\text{ème}}$ solution x^k.

Algorithme 2.2 : $k^{\text{ème}}$ solution selon Murty-Lawler

1 $\mathcal{L} \leftarrow \mathcal{L} \setminus \{v^{k-1}\}$
2 $\hat{X}(v^{k-1}) \leftarrow \hat{X}(v^{k-1}) \setminus \{x^{k-1}\}$
3 **tant que** $\hat{X}(v^{k-1}) \neq \emptyset$ **faire**
4 Créer le nœud v fils de v^{k-1} suivant le schéma **ML**
5 $\hat{X}(v^{k-1}) \leftarrow \hat{X}(v^{k-1}) \setminus \hat{X}(v)$
6 Déterminer $x(v)$ la solution optimale du problème $\mathcal{P}(\hat{X}(v), \hat{f})$
7 $\mathcal{L} \leftarrow \mathcal{L} \cup \{v\}$
8 $x^k \in \arg\min_{v \in \mathcal{L}} \hat{f}(x(v))$

Il existe d'autres algorithmes de k-best. À titre d'exemple citons les travaux de Hamacher dans (Hamacher et Queyranne, 1985), (Chegireddy et Hamacher, 1987) et (Hamacher, 1995). Il s'agit d'algorithmes de Branch and Bound avec séparation binaire. Afin d'utiliser ces algorithmes, il est nécessaire de savoir trouver la solution optimale et la seconde meilleure solution pour chacun des sous-problèmes définis par la séparation. Nous souhaitons maintenir un niveau de généralité que ces algorithmes ne permettent pas à cause de cette restriction. De plus et comme nous l'avons souligné dans la section 1.3.2.b,

l'algorithme de Murty (1968) pour le problème d'affectation est plus performant que les algorithmes de k-best proposés par Chegireddy et Hamacher (1987). Nous avons présenté plus de détails sur l'algorithme de Murty-Lawler et sur ces algorithmes alternatifs pour le problème d'affectation dans la section 1.3.2.

Remarques sur la complexité

Comme nous allons le voir, la complexité de l'algorithme 2.2 est en $O(n(C(n) + n))$, où $O(C(n))$ est la complexité des problèmes $\mathcal{P}(\hat{X}(v), \hat{f})$ et n la taille du problème $\mathcal{P}(\hat{X}, \hat{f})$. Nous commentons également dans ce qui suit la structure de données dans laquelle nous stockons l'ensemble $\mathcal{L}$, puis nous analysons de manière succincte la complexité de l'algorithme 2.2.

Il est pratique de stocker les nœuds de l'ensemble $\mathcal{L}$ dans un ordre non décroissant suivant la valeur de leurs solutions optimales selon la fonction $\hat{f}$. Une structure de données de type *tas* convient parfaitement à l'implémentation de l'ensemble $\mathcal{L}$. Sans rentrer dans les détails de la mise en œuvre informatique, l'insertion d'un nouvel élément dans un tas ou l'extraction de l'élément minimal et la remise en état du tas se fait en $O(\log t)$, où t est la taille de ce tas. Or, la taille de l'ensemble $\mathcal{L}$ ne peut pas dépasser dans notre cas le nombre de solutions réalisables dans $\hat{X}$, qui est lui-même une partie de $\{0,1\}^n$, ainsi $|\mathcal{L}|$ vaut au plus 2^n. Donc la complexité de l'insertion et de l'extraction d'un nœud de l'ensemble $\mathcal{L}$ est en $O(n)$.

Le nœud v^{k-1} qui contient la $k-1^{\text{ème}}$ meilleure solution est extrait de $\mathcal{L}$ et le tas est arrangé avec une complexité en $O(n)$. Puis le nœud v^{k-1} est séparé en $O(n)$ nœuds fils. Chaque nouveau nœud fils v contient le problème $\mathcal{P}(v)$ qui est résolu avec un effort au plus en $O(C(n))$, et le nœud v est inséré dans l'ensemble $\mathcal{L}$ avec une complexité en $O(n)$. La $k^{\text{ème}}$ meilleure solution est celle associée au premier nœud dans l'ensemble $\mathcal{L}$.

La complexité de l'algorithme 2.2 est donc en $O(n + n(C(n) + n))$, qui vaut après simplification $O(n(C(n) + n))$.

Il est possible d'améliorer la complexité de l'algorithme 2.2 en réduisant soit la complexité du problème $\mathcal{P}(v)$, ou bien le nombre de nœuds fils engendrés par la séparation de v^{k-1}. Prenons l'exemple de l'algorithme de k-best pour le problème de l'arbre couvrant de poids minimum (**MST**) dans un graphe valué non-orienté $G = (V, U)$. L'arbre couvrant est un problème de sélection d'arêtes, aussi à chaque séparation à la manière de Murty-Lawler, $O(|U|)$ fils sont créés. L'un des meilleurs algorithmes pour **MST** est une variation de l'algorithme de *Prim* proposée par Gabow et al. (1986). Sa complexité est de l'ordre de $O(|U| \log \beta(|V|, |U|))$, où $\beta(|V|, |U|)$ une fonction faiblement croissante. La complexité de la recherche du $k^{\text{ème}}$ meilleur arbre couvrant devrait être alors de l'ordre de $O(|U|(|U| \log \beta(|V|, |U|) + |U|))$. Or Katoh et al. (1981) calculent la $k^{\text{ème}}$ meilleure solution en $O(|U|)$ lorsque les $(k-1)$ meilleures solutions sont connues. L'objectif ici n'est pas l'étude du problème de l'arbre couvrant mais l'illustration des possibilités d'amélioration de la complexité de l'algorithme 2.2 compte tenu des propriétés du problème

étudié.

2.3 Énumération ordonnée et minoration variable

Nous avons supposé dans la section 2.2 que la fonction $\hat{f}$ est identique à chaque itération de l'algorithme 2.1. Autrement dit, la fonction $\hat{f}$ est définie préalablement et considérée comme un paramètre constant, choisi avant le début de l'algorithme 2.1. Cette hypothèse n'est pas nécessaire pour la validité de la proposition 1 et n'a été évoquée que pour donner l'intuition de la relation qui lie l'algorithme 2.1 et les algorithmes de k-best.

Parmi les commentaires de l'algorithme 2.1, nous avons mentionné l'utilité d'affiner l'approximation de la fonction objectif f par sa minoration $\hat{f}$. Intuitivement, il paraît naturel de penser que plus l'ensemble réalisable est restreint, plus une fonction minorante proche de f peut être définie, comme le montre l'exemple suivant.

Exemple. Il est facile de montrer que $e^y \geq y + 1$, donc la fonction $f(x) = e^{\sum_{i=1}^{n} x_i}$ peut être minorée pour tout $x \in \{0,1\}^n$ par la fonction linéaire $\hat{f}_{\{0,1\}^n}(x) = 1 + \sum_{i=1}^{n} x_i$. Supposons que nous définissions un sous-ensemble $X_1 \subset \{0,1\}^n$ tel que $X_1 = \{x \in \{0,1\}^n : x_1 = 1\}$. La fonction $\hat{f}_{\{0,1\}^n}$ reste une minoration valide pour l'ensemble X_1, néanmoins la fonction linéaire $\hat{f}_{X_1}(x) = e(1 + \sum_{i=2}^{n} x_i)$ est plus proche de la valeur de f pour les solutions appartenant à l'ensemble X_1.

Nous proposons une variante de l'algorithme 2.1 basée sur le principe de séparation de Murty-Lawler. La particularité de cette nouvelle version réside dans le fait que la valeur d'une solution x selon la fonction minorante $\hat{f}$ dépend du sous-ensemble auquel elle appartient, i.e. une fonction minorante est choisie pour chaque nœud v de l'ensemble $\mathcal{L}$.

Précisons que la fonction minorante associée à un nœud v, que nous noterons $\hat{f}_v$, est censée être meilleure que la minoration $\hat{f}_{v^{k-1}}$ du nœud père v^{k-1} qui a engendré v, i.e. $\hat{f}_v(x(v)) \geq \hat{f}_{v^{k-1}}(x(v^{k-1}))$. Ceci est nécessaire pour la validité de l'algorithme 2.1 qui repose essentiellement sur le principe de l'énumération ordonnée, dans le sens où la valeur de la $k^{\text{ème}}$ solution énumérée selon la fonction minorante doit être plus grande que la valeur des précédentes solutions énumérées. Si une meilleure fonction minorante ne peut pas être trouvée, alors il est préférable de garder la fonction minorante du père v^{k-1} comme minoration pour le nœud fils v, comme c'est le cas dans l'algorithme 2.2.

Les principales étapes de la recherche de la $k^{\text{ème}}$ meilleure solution pour cette variante sont décrites dans l'algorithme 2.3.

Algorithme 2.3 : $k^{\text{ème}}$ solution avec minoration choisie

$\mathcal{L} \leftarrow \mathcal{L} \setminus \{v^{k-1}\}$
$\hat{X}(v^{k-1}) \leftarrow \hat{X}(v^{k-1}) \setminus \{x^{k-1}\}$
tant que $\hat{X}(v^{k-1}) \neq \emptyset$ **faire**
    Créer le nœud v fils de v^{k-1} suivant le schéma **ML**
    $\hat{X}(v^{k-1}) \leftarrow \hat{X}(v^{k-1}) \setminus \hat{X}(v)$
    Choisir $\hat{f}_v$ `// Choisir` $\mathcal{P}(\hat{X}(v), \hat{f}_v)$ `relaxation de` $\mathcal{P}(\hat{X}(v) \cap X, f)$
    Déterminer $x(v)$ la solution optimale du problème $\mathcal{P}(\hat{X}(v), \hat{f}_v)$
    $\mathcal{L} \leftarrow \mathcal{L} \cup \{v\}$
$x^k \in \arg\min_{v \in \mathcal{L}} \hat{f}(x(v))$

L'algorithme 2.1 utilisant l'algorithme 2.3 pour la recherche d'une $k^{\text{ème}}$ meilleure solution peut être considéré comme un Branch and Bound quelconque. Cependant, il peut aussi être considéré comme une énumération ordonnée des solutions de la relaxation $\mathcal{P}(\hat{X}, \hat{f})$, avec :

$$\hat{f}(x) = \sum_{v \in \mathcal{L}} \mathbf{1}_{\hat{X}(v)}(x) \hat{f}_v(x)$$

telle que $\mathbf{1}_{\hat{X}(v)}$ est la fonction indicatrice définie par : $\mathbf{1}_{\hat{X}(v)}(x) = \begin{cases} 1 & \text{si } x \in \hat{X}(v), \\ 0 & \text{sinon.} \end{cases}$

L'algorithme 2.3 est une généralisation de l'algorithme 2.2. Nous ne ferons donc référence par la suite qu'à l'algorithme 2.3 et les remarques relatives à celui-ci sont aussi valables pour l'algorithme 2.2.

Remarques sur la complexité

Il est évident que l'algorithme 2.3 est plus laborieux que l'algorithme 2.2. En contrepartie, il est vraisemblable que l'algorithme 2.1, incluant l'algorithme 2.3 pour la recherche de la $k^{\text{ème}}$ meilleure solution, fasse moins d'itérations si les fonctions $\hat{f}_v$ sont convenablement choisies.

La complexité de l'algorithme 2.3 est de l'ordre de $O(n(F(n)+C(n)+n))$, où $F(n)$ est la complexité du choix de la fonction $\hat{f}_v$ et $C(n)$ la complexité des problèmes $\mathcal{P}(\hat{X}(v), \hat{f}_v)$. Nous montrons dans le chapitre 3 que dans certains cas, la sélection de la fonction $\hat{f}_v$ peut faciliter la résolution de $\mathcal{P}(\hat{X}(v), \hat{f}_v)$.

2.4 Amélioration des performances des algorithmes par énumération ordonnée

Il est possible d'introduire des perfectionnements à l'algorithme 2.1 qui améliorent ses performances lors de la mise en œuvre informatique, même si ces améliorations n'influent pas sur sa complexité théorique. Les conséquences de telles améliorations sont visibles par la diminution des temps de calcul ou par une meilleure gestion de l'espace mémoire.

Les algorithmes d'énumération ordonnée se heurtent à trois types de difficultés :

D1. L'algorithme 2.1 peut énumérer un nombre important de solutions avant d'atteindre la condition d'arrêt (étape 7 de l'algorithme 2.1) ;

D2. chacune des solutions énumérées s'obtient au prix de la résolution de $O(n)$ problèmes relâchés ;

D3. l'espace mémoire requis pour le stockage de l'ensemble des nœuds $\mathcal{L}$ est souvent important.

Afin de remédier à ces défauts, nous définissons des règles d'élagage que nous décrivons dans la proposition 2.

Proposition 2. Soit $\mathcal{P}(X, f)$ un problème d'optimisation combinatoire et soit x^* la meilleure solution courante de $\mathcal{P}(X, f)$ à l'itération k de l'algorithme 2.1. Soit v un nœud créé à l'étape 4 de l'algorithme 2.3 et $\hat{X}(v)$ le sous-ensemble de solutions qui lui est associé. Soit $\mathcal{P}(\tilde{X}(v), \tilde{f}_v)$ une relaxation du problème $\mathcal{P}(\hat{X}(v) \cap X, f)$. Si une des propositions suivantes est vérifiée,

$$\hat{X}(v) \cap X = \emptyset \tag{2.7}$$

$$\textbf{ou } \min_{y \in \tilde{X}(v)} \tilde{f}_v(y) \geq f(x^*) \tag{2.8}$$

$$\text{alors : } \forall x \in \hat{X}(v) \cap X : f(x) \geq f(x^*). \tag{2.9}$$

Preuve. L'assertion (2.9) est triviale lorsque (2.7) est vérifiée.

La preuve de l'implication sous l'hypothèse (2.8) est basée sur la définition de la relaxation : $\forall x \in \hat{X}(v) \cap X : f(x) \geq \tilde{f}_v(x) \geq \min_{y \in \tilde{X}(v)} \tilde{f}_v(y)$.
Or, si $\min_{y \in \tilde{X}(v)} \tilde{f}_v(y) \geq f(x^*)$, alors $\forall x \in \hat{X}(v) \cap X : f(x) \geq f(x^*)$. □

La proposition 2 permet de définir deux règles d'élagage simples. Nous les discutons plus en détail dans les sections 2.4.2 et 2.4.3, mais avant nous présentons dans la section 2.4.1 l'algorithme 2.4 qui est obtenu en introduisant ces règles d'élagage dans l'algorithme 2.3.

2.4.1 Algorithme de recherche de la $k^{\text{ème}}$ solution avec règles d'élagage

L'algorithme 2.4 contient les étapes de l'algorithme 2.3 auxquelles on a introduit des tests qui traduisent les règles d'élagage de la proposition 2. Les nouvelles instructions se distinguent par un astérisque placé avant le numéro de la ligne. Nous décrirons précisément ces tests dans les sections 2.4.2 et 2.4.3.

Algorithme 2.4 : $k^{\text{ème}}$ solution avec tests de bornes

```
 1  L ← L \ {v^{k-1}}
 2  X̂(v^{k-1}) ← X̂(v^{k-1}) \ {x^{k-1}}
 3  tant que X̂(v^{k-1}) ≠ ∅ faire
 4      Créer le nœud v fils de v^{k-1} suivant le schéma ML
 5      X̂(v^{k-1}) ← X̂(v^{k-1}) \ X̂(v)
 *6     si X̂(v) ∩ X ≠ ∅ alors
 *7         B̃(v) = min f̃_v(x) : x ∈ X̃(v), avec P(X̃(v), f̃_v) relaxation de
            P(X̂(v) ∩ X, f)
 *8         si B̃(v) < f(x*) alors
 9              Choisir P(X̂(v), f̂_v) relaxation de P(X̂(v) ∩ X, f)
 *10            B̂(v) = min f̆_v(x) : x ∈ X̆(v), avec P(X̆(v), f̆_v) relaxation de
                P(X̂(v), f̂_v)
 *11            si B̂(v) < f(x*) alors
 12                 Déterminer x(v) la solution optimale du problème P(X̂(v), f̂_v)
 *13                si f̂(v) < f(x*) alors                    // f̂(v) = f̂_v(x(v))
 14                     L ← L ∪ {X̂(v)}
 15 x^k ∈ arg min_{v∈L} f̂(x(v))
```

Les règles d'élagage issues de la proposition 2 sont de deux types. Celle issue de la condition (2.7) visant à supprimer les ensembles vides et celle issue de la condition (2.8) visant à supprimer les ensembles non améliorants en s'appuyant sur différents tests de bornes. Nous présentons et illustrons dans les sections 2.4.2 et 2.4.3 ces deux types de règles.

La règle d'élagage issue de la condition (2.7) est donnée dans l'algorithme 2.4 par le test de l'étape 6.

La règle d'élagage issue de la condition (2.8) est introduite dans l'algorithme 2.4 par trois tests de bornes. Le test de la borne $\hat{f}(v)$ (étape 13 de l'algorithme 2.4) est commun à tous les algorithmes de Branch and Bound et consiste à supprimer les nœuds, qui après résolution de la relaxation principale $\mathcal{P}(\hat{X}(v), \hat{f}_v)$, montrent qu'il ne peuvent contenir une solution optimale. D'autres tests de bornes visent à éviter la résolution du problème relâché principal $\mathcal{P}(\hat{X}(v), \hat{f}_v)$ associé à un nœud v. C'est le cas du test de la

borne $\hat{B}(v)$ qui est issue d'une relaxation de la relaxation principale notée ici $\mathcal{P}(\breve{X}(v), \breve{f}_v)$ (étape 11 de l'algorithme 2.4). Un troisième type de borne, dont $\tilde{B}(v)$ est un exemple, est obtenu par la résolution d'un problème relâché qui est construit sur la base d'un type relaxation différent de celui de la relaxation principale. Le test de la borne $\tilde{B}(v)$ (étape 8 de l'algorithme 2.4) vise à supprimer des nœuds qui contiennent des solutions candidates à l'énumération.

Notons que les positions des tests de bornes peuvent être modifiées sans altérer la validité de l'algorithme 2.4 ; elles ne sont données ici qu'à titre indicatif. Nous précisons les positions que les règles d'élagage peuvent occuper dans l'algorithme 2.4 lors de leur description dans les sections 2.4.2 et 2.4.3.

2.4.2 Suppression des sous-ensembles vides

Il peut arriver, suite aux séparations successives de l'ensemble réalisable du problème relâché, qu'un sous-ensemble $\hat{X}(v)$ soit vide ou ne contienne que des solutions de $\hat{X} \setminus X$ (i.e. $\hat{X}(v) \cap X = \emptyset$). Les solutions de ce type d'ensembles peuvent évidemment être supprimées du processus d'énumération sans conséquences sur la validité de l'algorithme 2.1.

Exemple. Prenons l'exemple du problème du voyageur de commerce asymétrique que l'on présentera dans le chapitre 4. Il est bien connu que le problème d'affectation est une relaxation de ce problème. Un nœud v est construit en imposant et en interdisant des arêtes pour le problème d'affectation. Ces arêtes correspondent à des arcs pour le problème du voyageur de commerce. Si l'ensemble de solutions $\hat{X}(v)$ est construit en imposant des arêtes qui forment un circuit dans le problème du voyageur de commerce asymétrique, et si ce circuit ne parcourt pas tous les sommets du graphe, alors $\hat{X}(v)$ ne contient aucune solution réalisable pour le problème du voyageur de commerce asymétrique.

Cette règle d'élagage est introduite sous la forme du test de l'étape 6 de l'algorithme 2.4, et son emploi impacte les trois défauts cités en début de la section 2.4. En effet, lorsque la condition du test n'est pas vérifiée pour un nœud v, la relaxation $\mathcal{P}(\hat{X}(v), \hat{f}_v)$ n'est pas résolue et le nœud v n'est donc pas stocké dans l'ensemble $\mathcal{L}$. Ceci permet d'améliorer le temps de calcul de la $k^{\text{ème}}$ solution et de réduire l'espace mémoire utilisé pour le stockage de l'ensemble $\mathcal{L}$. Par ailleurs, cette règle d'élagage supprime des solutions candidates à l'énumération. En effet, les solutions qui ne sont pas réalisables pour le problème principal $\mathcal{P}(X, f)$ et qui sont supprimées par cette règle peuvent avoir des valeurs selon $\hat{f}$ inférieures à la valeur de la solution optimale $f(x^*)$. Ce test diminue donc le nombre d'itérations du processus d'énumération ordonnée.

2.4.3 Suppression des sous-ensembles non-améliorants

Nous présentons dans les sections 2.4.3.c, 2.4.3.a et 2.4.3.b trois tests de bornes issus du principe de la règle (2.8).

La règle d'élagage issue de la condition (2.8) est mise en œuvre en calculant des bornes inférieures à la valeur de solution optimale du problème $\mathcal{P}(\hat{X}(v) \cap X, f)$ associée au nœud v. Ces bornes inférieures s'obtiennent par la résolution de différentes relaxations associées au problème $\mathcal{P}(\hat{X}(v) \cap X, f)$.

La combinaison de la position des tests dans l'algorithme 2.4 et du choix des relaxations qui produisent les bornes inférieures détermine les améliorations attendues de cette règle d'élagage.

2.4.3.a Amélioration de la gestion de la mémoire (borne $\hat{f}$)

Le problème $\mathcal{P}(\hat{X}(v), \hat{f}_v)$ est une relaxation de $\mathcal{P}(\hat{X}(v) \cap X, f)$ qui constitue un sous-problème de la relaxation principale $\mathcal{P}(\hat{X}, \hat{f}_v)$. La borne inférieure $\hat{f}(v)$ qui correspond à la valeur optimale du problème $\mathcal{P}(\hat{X}(v), \hat{f})$ (i.e. avec $\hat{f}(v) = \min \hat{f}_v(x) : x \in \hat{X}(v)$) pourrait aussi servir le test de l'étape 13 de l'algorithme 2.4. Ce type de test de borne est naturellement introduit dans tous les algorithmes de Branch and Bound.

Les solutions des nœuds coupés par ce test de borne ne seraient pas énumérées, même si ce test n'est pas introduit. En effet, la condition d'arrêt de l'algorithme d'énumération ordonnée (étape 7 de l'algorithme 2.1) est obtenue avant l'énumération de ces solutions, puisque pour tout $x \in \hat{X}(v) : \hat{f}(x) \geq f(x^*)$.

Ce test a de l'influence principalement sur la gestion de l'espace mémoire, de telle sorte que seuls les nœuds ayant des solutions candidates à l'énumération seront conservés dans l'ensemble $\mathcal{L}$. Par ailleurs, les insertions et les extractions des nœuds se font plus rapidement lorsque la taille de l'ensemble $\mathcal{L}$ est réduite, aussi l'amélioration du temps de calcul de la recherche de la $k^{\text{ème}}$ solution est une conséquence naturelle mais indirecte de cette règle d'élagage.

Ce test de borne est naturellement positionné après la résolution du problème $\mathcal{P}(\hat{X}(v), \hat{f}_v)$ (i.e. étape 12 de l'algorithme 2.4) et avant l'insertion du nœud v dans l'ensemble $\mathcal{L}$ (i.e. étape 14 de l'algorithme 2.4).

2.4.3.b Amélioration de la recherche de la $k^{\text{ème}}$ meilleure solution (borne $\hat{B}$)

La relaxation principale $\mathcal{P}(\hat{X}(v), \hat{f}_v)$ associée à certains nœuds de l'arborescence se trouvent être contrainte à force de séparations successives. Généralement dans ce type de cas, il suffit d'un minorant de la valeur optimale de ces nœuds pour prouver qu'ils ne peuvent pas contenir la solution optimale du problème principal. C'est pour cela qu'il est intéressant de considérer le cas spécial de la borne $\hat{B}(v)$ qui est fournie par une relaxation associée à la relaxation principal $\mathcal{P}(\hat{X}(v), \hat{f}_v)$. Placer un test de borne utilisant ce type de relaxation, comme on le montre à l'étape 11 de l'algorithme 2.4, permettrait d'éviter l'effort de la résolution du problème $\mathcal{P}(\hat{X}(v), \hat{f}_v)$ lorsque la condition n'est pas vérifiée.

Exemple. Un problème de plus court chemin entre deux sommets i^* et j^* contenant des

contraintes supplémentaires (contraintes de ressources par exemple) est un problème difficile. Un problème de plus court chemin non contraint constitue une relaxation pour ce problème difficile. Aussi des chemins peuvent être énumérés, dans un ordre non décroissant, afin résoudre le problème difficile de plus court chemin contraint. La borne $\hat{B}(v)$ associée au nœud v de l'arborescence, et que nous décrivons dans cette section, est une minoration de la valeur d'un plus court chemin non contraint associé au nœud v. En d'autres termes cette borne est obtenue par une relaxation de la relaxation principale. Ici par exemple on peut considérer que la borne $\hat{B}(v)$ vaut la somme des valuations minimales des arcs sortant du sommet i^* et des arcs entrant dans le sommet j^*.

Le test de la borne $\hat{B}(v)$ utilisant ce type de relaxation a des conséquences uniquement sur le temps de la recherche de la $k^{\text{ème}}$ meilleure solution. En effet, la valeur de la borne fournie par ce test est inférieure à celle de $\hat{f}(v)$ présentée dans la section 2.4.3.a (i.e. $\hat{B}(v) \leq \hat{f}(v)$). Par conséquent, tous les nœuds coupés par ce test seront aussi coupés par le test de l'étape 13 de l'algorithme 2.4 défini dans la section 2.4.3.a. Ce test de borne n'a de sens alors que si le problème $\mathcal{P}(\breve{X}(v), \breve{f}_v)$ est plus facile que le problème $\mathcal{P}(\hat{X}(v), \hat{f}_v)$. De plus, le test doit être placé avant la résolution de $\mathcal{P}(\hat{X}(v), \hat{f}_v)$ (étape 12 de l'algorithme 2.4).

2.4.3.c Amélioration de l'énumération (borne $\tilde{B}$)

Nous présentons dans cette section la borne $\tilde{B}(v)$ obtenue par la résolution du problème $\mathcal{P}(\tilde{X}(v), \tilde{f}_v)$ qui est une relaxation de $\mathcal{P}(\hat{X}(v) \cap X, f)$ différente de $\mathcal{P}(\hat{X}(v), \hat{f}_v)$ (i.e. $\tilde{B}(v) = \min \tilde{f}_v(x) \ : \ x \in \tilde{X}(v)$). Le test de la borne $\tilde{B}(v)$ est introduit à l'étape 8 de l'algorithme 2.4.

Le principe de l'énumération ordonnée impose que la relaxation principale $\mathcal{P}(\hat{X}(v), \hat{f}_v)$ (i.e. la relaxation qui mène le processus d'énumération de l'algorithme 2.1) soit un problème d'optimisation combinatoire. L'introduction du test de la borne $\tilde{B}(v)$ est motivée par la volonté d'utiliser d'autres types de relaxation pouvant fournir de meilleures bornes inférieures. En effet, la borne $\tilde{B}(v)$ est obtenue à l'aide d'une relaxation $\mathcal{P}(\tilde{X}(v), \tilde{f}_v)$ alternative à la relaxation principale $\mathcal{P}(\hat{X}(v), \hat{f}_v)$. Le fait que les solutions du problème $\mathcal{P}(\tilde{X}(v), \tilde{f}_v)$ ne soient pas destinées à l'énumération autorise plus de flexibilité dans la recherche de la relaxation qui réalise la condition (2.8).

Exemple. Considérons un algorithme par énumération ordonnée de chemins pour résoudre le problème difficile du plus court chemin contraint présenté dans l'exemple de la section 2.4.3.b. La borne $\tilde{B}(v)$ peut correspondre dans ce cas à l'optimum de la relaxation continue de la formulation en programme linéaire en nombre entiers du problème difficile.

Dans le cas général, un nœud v auquel il est possible d'associer une relaxation $\mathcal{P}(\tilde{X}(v), \tilde{f}_v)$ qui remplit la condition (2.8), peut contenir des solutions qui peuvent être énumérées, même si ces solutions ne conduisent pas à l'amélioration de la valeur de x^*. Par conséquent, la suppression de ces nœuds diminue le nombre de solutions énumérées et remédie

donc à la faiblesse **D1.** des méthodes par énumération ordonnée. Par ailleurs, lorsque la relaxation $\mathcal{P}(\tilde{X}(v), \tilde{f}_v)$ est relativement facile à résoudre, le test de la borne $\tilde{B}(v)$ allège aussi l'effort de la recherche de la $k^{\text{ème}}$ meilleure solution en rétrécissant la séparation de Murty-Lawler.

Le test de la borne $\tilde{B}(v)$ peut être déplacé dans l'algorithme 2.4, et cela dans toutes les positions entre la création du nœud v (i.e. étape 4 de l'algorithme 2.4) et l'insertion de celui-ci dans $\mathcal{L}$ (i.e. étape 14 de l'algorithme 2.4). Le choix de la position de ce test est heuristique et dépend de la difficulté du problème $\mathcal{P}(\tilde{X}(v), \tilde{f}_v)$ et de la qualité de la borne $\tilde{B}(v)$ qu'il fournit. Par exemple, lorsque le calcul de $\tilde{B}(v)$ et plus facile que la résolution de la relaxation principale $\mathcal{P}(\hat{X}(v), \hat{f}_v)$, le test de la borne $\tilde{B}(v)$ est placé avant la résolution de $\mathcal{P}(\hat{X}(v), \hat{f}_v)$.

2.4.3.d Récapitulatif sur les bornes

Les informations liées aux bornes $\hat{f}(v)$, $\hat{B}(v)$ et $\tilde{B}(v)$ sont récapitulées dans la table 2.1.

Notation	Relaxation	Relaxation de	Section	Algorithme 2.4	Conséquences directes
$\hat{f}(v)$	$\mathcal{P}(\hat{X}(v), \hat{f}_v)$	$\mathcal{P}(\hat{X}(v) \cap X, f)$	2.4.3.a	13	**D3.**
$\hat{B}(v)$	$\mathcal{P}(\breve{X}(v), \breve{f}_v)$	$\mathcal{P}(\hat{X}(v), \hat{f}_v)$	2.4.3.b.	11	**D2.**
$\tilde{B}(v)$	$\mathcal{P}(\tilde{X}(v), \tilde{f}_v)$	$\mathcal{P}(\hat{X}(v) \cap X, f)$	2.4.3.c	8	**D1.**

TABLE 2.1 – Récapitulatif des bornes

Dans l'ordre, chaque ligne de la table 2.1 nous donne la notation de la borne, la relaxation qui a permis de la calculer, le problème dont la valeur optimale est minorée par cette borne, la section dans laquelle elle a été détaillée, l'étape de l'algorithme 2.4 où elle est placée. Enfin, la dernière colonne indique le type de difficulté (voir début de section 2.4) sur lequel le test de borne agit directement.

2.5 Conclusion

Nous avons présenté dans ce chapitre une méthode générique qui procède par énumération ordonnée pour résoudre des problèmes d'optimisation combinatoire. Nous mettons en œuvre cette méthode comme un algorithme de Branch and Bound, pour lequel nous proposons deux variantes qui se distinguent par une minoration unique ou variable. Nous proposons d'introduire des améliorations dans le schéma général de notre algorithme. Ces améliorations sont données sous la forme de règles d'élagage permettant d'améliorer les performances de notre méthode.

Nous présenterons dans les chapitres 3 et 4 deux applications de cette méthode, une pour la recherche de solutions de bon compromis pour le problème d'affectation multiobjectif et la seconde pour la résolution du problème du voyageur de commerce asymétrique.

Chapitre 3
Solution de compromis pour le problème d'affectation multiobjectif [1]

Résumé Dans ce chapitre nous présentons les algorithmes que nous proposons pour la recherche d'une solution de bon compromis pour le problème d'affectation multiobjectif. Nous considérons qu'une telle solution minimise une fonction scalarisante non-linéaire de type point de référence.

Dans la section 3.1, nous formalisons le problème. Nous précisons dans les sections 3.2 et 3.3 les spécificités de la procédure générique d'énumération ordonnée, présenté dans le chapitre 2, lorsqu'elle est utilisée dans le contexte de la recherche de bon compromis et nous présentons ensuite différentes versions de cette procédure. Nous présentons dans la section 3.4 une synthèse des résultats expérimentaux obtenus par la mise en œuvre informatique de nos algorithmes. Nous montrons dans cette dernière section l'efficacité globale de notre méthode.

---

1. Ce chapitre est basé principalement sur le papier de Belhoul et al. (2014)

3.1 Définitions préliminaires

Le problème d'affectation multiobjectif peut être formulé comme un programme linéaire multiobjectif en nombres entiers :

$$\begin{array}{lll} \min z_k(x) = & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}^k x_{ij} & k = 1,\ldots,p \\ \text{s.c.} & \sum_{j=1}^{n} x_{ij} = 1 & i = 1,\ldots,n \\ & \sum_{i=1}^{n} x_{ij} = 1 & j = 1,\ldots,n \\ & x_{ij} \in \{0,1\} & i = 1,\ldots,n,\ j = 1,\ldots,n \end{array} \tag{3.1}$$

où $(c_{ij}^1,\ldots,c_{ij}^p) \in \mathbb{N}_{\geq}^p$ le coût vectoriel associé à l'affectation de l'élément i à l'élément j, avec $i,j = 1,\ldots,n$. Comme dans le cas mono-objectif, le problème d'affectation multiobjectif peut être considéré comme le problème du couplage parfait multiobjectif dans un graphe biparti complet $G = (U \cup V, E)$ muni des valuations vectorielles $(c_{ij}^1,\ldots,c_{ij}^p)$, avec $|U| = |V| = n$.

Sans renoncer à l'aspect multiobjectif du problème original, le problème de recherche de solution de bon compromis est bien un problème d'optimisation combinatoire mono-objectif non-linéaire, issu de l'agrégation des objectifs du problème (3.1) à l'aide d'une pseudo-distance de type point de référence. Autrement dit, notre problème peut se formaliser comme le programme mathématique non linéaire (3.2), avec des contraintes implicites sur l'efficacité de la solution optimale retournée.

$$\begin{array}{lll} \min & f(z(x)) = \max\limits_{k=1,\ldots,p}\{\lambda_k(\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}^k x_{ij} - \bar{z}_k)\} & \\ \text{s.c.} & \sum_{j=1}^{n} x_{ij} = 1 & i = 1,\ldots,n \\ & \sum_{i=1}^{n} x_{ij} = 1 & j = 1,\ldots,n \\ & x_{ij} \in \{0,1\} & i = 1,\ldots,n,\ j = 1,\ldots,n \end{array} \tag{3.2}$$

où $\lambda \in \mathbb{R}_{>}^p$ est la direction de recherche et $\bar{z} \in \mathbb{R}^p$ le point de référence. Le problème d'affectation avec le critère *min max Regret*, prouvé NP-difficile [2] par Kouvelis et Yu (1997), est un cas particulier du problème (3.2), aussi le problème (3.2) est NP-difficile.

Afin de résoudre le problème (3.2) tout en garantissant l'efficacité de la solution optimale, des approches en deux phases peuvent être envisagées. Ces approches consistent à générer toutes les solutions efficaces pour le problème (3.1), puis à choisir la solution

2. Réduction au problème de 2-partition

qui minimise la fonction f parmi les solutions efficaces. Cependant ce type d'approches a ses limites et n'est pas approprié pour résoudre notre problème. Nous donnons dans ce qui suit deux raisons qui nous ont conduits à écarter ces approches.

Le premier obstacle à l'utilisation des approches en deux phases est théorique ; le problème d'affectation multiobjectif est *intraitable*. Une famille d'instances du problème d'affectation avec deux critères qui le prouve peut être consultée dans le livre de Ehrgott (2005). Tous les points réalisables de chaque instance de cette famille sont non-dominés. Nous proposons une autre famille d'instances avec deux objectifs qui a les mêmes propriétés. Soient $c^1_{ij} = (j-1)n^{i-1}$ et $c^2_{ij} = n^n - c^1_{ij}$ les coûts de l'arête (i,j) selon le premier et le second objectif, pour tout $i,j = 1,\ldots,n$. Les points réalisables de cette instance sont tous non-dominés, i.e. pour toute paire de solutions différentes x_1, x_2, $z(x_1)$ ne domine pas $z(x_2)$ et $z(x_2)$ ne domine pas $z(x_1)$.
En effet, soit x_1 une solution réalisable avec $E(x_1) = \{(1,j_1),\ldots,(n,j_n)\}$. Sa valeur selon le premier critère est $z_1(x_1) = \sum_{k=1}^{n}(j_k-1)n^{k-1}$ et peut s'écrire aussi $z_1(x_1) = (j_n-1)\ldots(j_1-1)_n$ en base n, où (j_k-1) est le $k^{\text{ème}}$ chiffre de $z_1(x_1)$ écrit en base n. Soit x_2 une solution réalisable différente de x_1, i.e. il existe au moins une arête, notons la (k,j_k) avec $k \in \{1,\ldots,n\}$, telle que $(k,j_k) \in E(x_1)$ et $(k,j_k) \notin E(x_2)$. Les valeurs $z_1(x_1)$ et $z_1(x_2)$ diffèrent donc au moins par le $k^{\text{ème}}$ chiffre de leurs valeurs écrites en base n. Il en résulte alors que pour tout $x_1, x_2 \in X, x_1 \neq x_2, z_1(x_1) \neq z_1(x_2)$.
D'autre part pour toute solution réalisable x : $z_2(x) = n^{n+1} - z_1(x)$. Par conséquent, la comparaison de deux solutions quelconques x_1, x_2 de X donne lieu à deux situations : soit ($z_1(x_1) < z_1(x_2)$ et $z_2(x_1) > z_2(x_2)$ ou bien $z_1(x_1) > z_1(x_2)$ et $z_2(x_1) < z_2(x_2)$, donc $z(x_1)$ et $z(x_2)$ ne sont pas comparables.

D'un autre côté, les approches en deux phases sont inefficaces en pratique pour la recherche d'une solution de bon compromis. Plusieurs procédures ont été proposées pour déterminer l'ensemble de toutes les solutions efficaces pour le problème d'affectation multiobjectif. On doit les plus efficaces à Przybylski et al. (2008) pour le cas avec deux objectifs et Przybylski et al. (2010) pour le cas avec trois objectifs. Les auteurs parviennent à résoudre des instances de taille relativement grande pour la génération de toutes les solutions efficaces ($n = 100$ pour deux objectifs et $n = 50$ pour trois objectifs). Nous souhaitons néanmoins résoudre des instances de plus grande taille pour la recherche d'une solution de compromis. De plus, le passage de deux à trois critères n'est pas trivial.

Nous souhaitons proposer une procédure pour la recherche d'une solution de bon compromis qui ne dépend pas du nombre d'objectifs et qui permet de résoudre de grandes instances, aussi bien au sens de la taille du problème d'affectation n qu'en termes de nombre d'objectifs p. C'est pour cela que nous nous tournons vers des approches de recherche directe.

Les méthodes classiques de résolution des programmes linéaires mixtes en nombres entiers sont des approches directes envisageables pour résoudre le problème (3.2). En effet, le problème (3.3) est une formulation linéaire équivalente au problème (3.2). Cependant ces méthodes souffrent de deux faiblesses. Nous montrons dans la section 3.4, dévolue aux résultats expérimentaux, que les algorithmes par énumération ordonnée sont plus

efficaces que ces approches. De plus ce type de méthodes ne garantit pas l'efficacité de la solution optimale retournée.

$$\begin{array}{lll} \min & \mu & \\ \text{s.c.} & \mu \geq \lambda_k(\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{n} c_{ij}^k x_{ij} - \bar{z}_k) & k = 1, \ldots, p \\ & \sum\limits_{j=1}^{n} x_{ij} = 1 & i = 1 \ldots n \\ & \sum\limits_{i=1}^{n} x_{ij} = 1 & j = 1 \ldots n \\ & x_{ij} \in \{0, 1\} & i, j = 1, \ldots, n \end{array} \tag{3.3}$$

Nous proposons donc des algorithmes par énumération ordonnée pour la recherche d'une solution de bon compromis pour le problème d'affectation multiobjectif. Dans un premier temps nous présentons dans la section 3.2 une procédure d'énumération ordonnée avec une fonction minorante dont les paramètres ne varient pas pendant le processus d'énumération (voir la section 2.2). Nous précisons ensuite dans la section 3.3, les modifications qu'il faut apporter à la version précédente afin d'obtenir la variante de la procédure avec minoration variable (voir la section 2.3).

3.2 Énumération ordonnée avec minoration unique

L'algorithme 3.1 que nous présentons dans cette section est une version de l'algorithme 2.1, présentée dans le chapitre 2, instanciée pour résoudre le problème (3.2). On notera $\mathcal{P}(X, f)$ le problème (3.2), avec X l'ensemble réalisable du problème d'affectation et f la pseudo-distance de type point de référence. Puisque le problème (3.2) a pour ensemble de solutions réalisables l'ensemble des affectations possibles, il n'est pas nécessaire ici de relâcher l'ensemble X, mais seulement minorer la fonction f par une fonction linéaire $\hat{f}$. La relaxation principale $\mathcal{P}(X, \hat{f})$ est donc un problème d'affectation linéaire.

Algorithme 3.1 : Résolution du problème (3.2) par l'énumération ordonnée

Données : $\mathcal{P}(X, f), \mathcal{P}(X, \hat{f})$
$k \leftarrow 0$, $\hat{Y} \leftarrow \emptyset$
répéter
  $k \leftarrow k + 1$
  Déterminer $x^k \in \arg\min\limits_{x \in X \setminus \hat{Y}} \hat{f}(z(x))$ `// Résoudre` $\mathcal{P}(X \setminus \hat{Y}, \hat{f})$
  $\hat{Y} \leftarrow \hat{Y} \cup \{x^k\}$
  si $f(z(x^k)) < f(z(x^*))$ **alors** $x^* \leftarrow x^k$
jusqu'à *(*$\hat{f}(z(x^k)) \geq f(z(x^*))$ *ou* $X \setminus \hat{Y} = \emptyset$*)*
retourner x^*

À notre connaissance, Drezner et Nof (1984) ont été les premiers à proposer l'idée d'énumérer des solutions réalisables pour le problème d'affectation dans l'ordre afin de résoudre un problème d'industrie de *Bin picking*.

Sous certaines conditions, il est possible de tirer avantage de l'intégrité des valeurs des solutions dans l'espace des objectifs afin d'améliorer la condition d'arrêt (étape 7 de l'algorithme 3.1). Par exemple, pour optimiser une pseudo-distance de type point de référence, on peut observer que la zone grisée dans la figure 3.1 ne contient pas de point correspondant à une solution réalisable. Dans ce cas, la condition d'arrêt (étape 7) peut être remplacée par l'instruction 7' :

7' **jusqu'à** $(\hat{f}(z(x^k)) > f_{(\lambda,\bar{z})}(y(x^*))$ ou $X \setminus \hat{Y} = \emptyset)$

où $y(x^*) = (\left\lceil \frac{f(z(x^*))}{\lambda_1} + \bar{z}_1 \right\rceil - 1, \ldots, \left\lceil \frac{f(z(x^*))}{\lambda_p} + \bar{z}_p \right\rceil - 1)$.

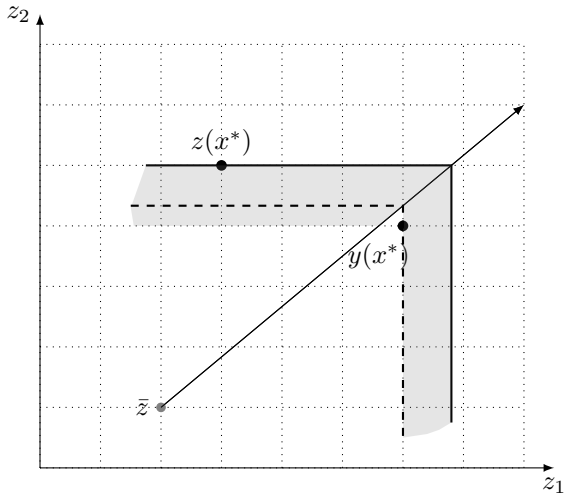


FIGURE 3.1 – Amélioration de la condition d'arrêtde l'énumération ordonnée dans le cas de coûts entiers

3.2.1 Fonction minorante

Rappelons que les solutions optimales de $\mathcal{P}(X, f)$ sont faiblement efficaces et qu'il existe au moins une parmi ces solutions qui est efficace. Nous montrons dans la proposition 3 que sous certaines conditions sur la fonction linéaire minorante $\hat{f}$, on peut garantir que la solution x^* retournée par l'algorithme 3.1 est efficace, et ce sans aucun test de dominance.

Proposition 3. Lorsque le problème (3.2) est résolu avec l'algorithme 3.1, en utilisant une relaxation $\mathcal{P}(X, \hat{f})$, telle que la fonction minorante $\hat{f}$ est fortement monotone, alors la solution optimale x^* retournée par l'algorithme 3.1 est efficace.

Preuve. Soient x et y deux solutions réalisables telles que $\hat{f}(z(x)) \leq \hat{f}(z(y))$. La solution y ne peut pas dominer x. En effet, si la solution y domine la solution x alors $\hat{f}(z(y)) < \hat{f}(z(x))$, pour toute fonction $\hat{f}$ fortement monotone. Ceci implique qu'une solution x ne peut pas être dominée par une autre solution y énumérée après elle. Par conséquent, stocker la première meilleure solution courante dans x^* (étape 6 de l'algorithme 3.1) garantit que x^* est efficace, sans aucun test de dominance. □

Dans cette version de l'énumération ordonnée, nous utilisons la fonction linéaire (3.4). La fonction $\hat{f}_\omega$ est définie pour toute solution x de X dont l'image dans l'espace des objectifs est $z(x) \in Z$, tout vecteur $\lambda \in \mathbb{R}^p_>$ et pour tout point de référence $\bar{z} \in \mathbb{R}^p$.

$$\hat{f}_\omega(z(x)) = \sum_{k=1}^{p} \omega_k \lambda_k (z_k(x) - \bar{z}_k) \tag{3.4}$$

où, le vecteur de poids $\omega \in \mathbb{R}^p_\geq$ vérifie la condition $\sum_{k=1}^{p} \omega_k = 1$. La fonction $\hat{f}_\omega$ peut s'écrire $\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} - C^{te}$, avec $c_{ij} = \sum_{i=1}^{p} \omega_k \lambda_k c_{ij}^k$ pour $i, j = 1, \ldots, n$ et $C^{te} = \sum_{i=1}^{p} \omega_k \lambda_k \bar{z}_k$.

L'ensemble de ces jeux de poids Ω définit une famille F_Ω de fonctions linéaires minorantes. Notons que toute fonction linéaire minorante $\hat{f}_\omega$ de la famille F_Ω peut être utilisée dans l'algorithme 3.1 pour résoudre le problème (3.2).

Soit $\Omega_>$ un sous-ensemble de Ω dans lequel les composantes des jeux de poids sont strictement positives (i.e. $\Omega_> = \Omega \cap \mathbb{R}^p_>$), et soit $F_{\Omega_>}$ la famille des fonctions engendrées par l'ensemble de poids $\Omega_>$. Les fonctions $\hat{f}_\omega$ appartenant aux familles F_Ω et $F_{\Omega_>}$ sont respectivement strictement et fortement monotones.

Le nombre de solutions énumérées dépend du jeu de poids ω qui définit la fonction minorante $\hat{f}_\omega$. Afin d'améliorer la qualité de la minoration, nous proposons de choisir le jeu de poids ω qui optimise le problème (3.5).

$$\max_{\omega \in \Omega_>} \min_{x \in X} \quad \sum_{k=1}^{p} \omega_k \lambda_k \Big(\sum_{i=1}^{n}\sum_{j=1}^{n} \quad c_{ij}^k x_{ij} - \bar{z}_k\Big) \tag{3.5}$$

Nous supposons ici que la fonction minorante de la famille $F_{\Omega_>}$ dont la valeur de la -première- meilleure solution $f^w(z(x^1))$ est la plus proche de la valeur $f(z(x^1))$, est aussi une bonne minoration pour les solutions qui seront énumérées par la suite $(x^2, x^3, \ldots)$.

Galand et al. (2010) utilisent l'algorithme du *sous-gradient* (voir Shor (1985)) pour choisir une fonction minorante afin d'optimiser l'intégrale de Choquet par une procédure d'énumération ordonnée. Le problème (3.5) peut être résolu de manière plus efficace. En effet, relâcher les contraintes d'intégrité du problème (3.5) donne la formulation équivalente (3.6).

$$\max_{\omega\in\Omega_>} \min_{x} \quad \sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{k=1}^{p} \omega_k\lambda_k c_{ij}^k x_{ij} \quad -\sum_{k=1}^{p}\lambda_k\bar{z}_k\omega_k$$

$$\text{s.c.} \qquad \sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1,\ldots,n \tag{3.6.a}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1,\ldots,n \tag{3.6.b}$$

$$x_{ij} \geq 0 \qquad i,\ j = 1,\ldots,n$$

Pour un jeu de poids ω fixe, la minimisation du problème à l'intérieur du problème de maximisation est équivalente à la maximisation de sa version duale, et ce en vertu du principe fort de la dualité.

$$\max \quad \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j - \sum_{k=1}^{p}\lambda_k\bar{z}_k\omega_k$$

$$\text{s.c.} \qquad u_i + v_j \leq \sum_{k=1}^{p}\lambda_k c_{ij}^k\omega_k \qquad i,\ j = 1,\ldots,n \tag{3.7.a}$$

$$\sum_{k=1}^{p}\omega_k = 1 \tag{3.7.b}$$

$$\omega_k > 0 \qquad k = 1,\ldots,p \tag{3.7.c}$$

où u_i et v_j sontles variables duales associées aux contraintes (3.6.a) et (3.6.b).

Les inégalités (3.7.b) et (3.7.c) sont les contraintes qui définissent l'ensemble des jeux de poids $\Omega_>$. En remplaçant les contraintes (3.7.c) par $\omega_k \geq 0$ pour $k = 1,\ldots,p$, on obtient un programme linéaire. Le jeu de poids ω^* qui optimise le programme linéaire (3.7) n'est pas forcément dans $\Omega_>$, puisqu'une de ses composantes peut être nulle. Afin de garantir l'efficacité de la solution optimale retournée par l'algorithme 3.1, nous proposons de modifier légèrement les composantes du jeu de poids optimal ω^* si toutefois une de ses composantes est nulle. Les nouvelles valeurs de ω^* sont données par $\omega_k^* \longleftarrow \frac{\omega_k^*+\varepsilon}{1+p\varepsilon}$ pour $k = 1,\ldots,p$, avec ε une petite quantité positive.

Il est intéressant de remarquer que la version duale (3.8) du programme linéaire (3.7)

avec $\omega_k \geq 0$ pour tout $k = 1, \ldots, p$ est la relaxation continue du problème (3.3).

$$
\begin{aligned}
\min \quad & \mu & & \\
\text{s.c.} \quad & \mu \geq \lambda_k \Big(\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}^k x_{ij} - \bar{z}_k\Big) & k = 1, \ldots, p & \quad (3.8.a)\\
& \sum_{j=1}^{n} x_{ij} = 1 & i = 1 \ldots n & \quad (3.8.b)\\
& \sum_{i=1}^{n} x_{ij} = 1 & j = 1 \ldots n & \quad (3.8.c)\\
& x_{ij} \geq 0 & i,\ j = 1, \ldots, n &
\end{aligned}
$$

où x_{ij}, pour $i, j = 1, \ldots, n$, sont les variables duales associées aux contraintes (3.7.a) et μ la variable duale associée à la contrainte (3.7.b).

Le jeu de poids optimal ω^* s'obtient donc par la résolution de la relaxation continue de la formulation linéaire du problème (3.2). Ses composantes correspondent aux p variables duales des contraintes (3.8.a) de linéarisation de la pseudo-distance. Le vecteur ω^* doit être modifié lorsqu'il n'est pas dans $\Omega_>$, comme on l'a expliqué précédemment.

Remarques sur la définition de fonctions minorantes pour d'autres fonctions scalarisantes

La démarche que nous venons de décrire pour la recherche du meilleur jeu de poids ω^* peut être appliquée à des problèmes dont la version mono-objectif linéaire est avec *saut de dualité nul*, c'est-à-dire les problèmes qui peuvent être formulés par un programme linéaire continu comme c'est le cas pour le problème d'affectation. Par exemple, les problèmes qui peuvent être formulés comme un flot, tel que le problème du plus court chemin, possèdent cette propriété.

Nous montrons ici les étapes qui prouvent l'équivalence entre la recherche d'une bonne minoration pour la fonction scalarisante additive non-linéaire du problème (3.9) proposée par (Ruiz et al., 2008) et la relaxation continue du problème (3.9).

$$
\begin{aligned}
\min \quad & \sum_{k=1}^{p} \max\{\lambda_k(z_k(x) - \bar{z}_k), 0\} \\
\text{s.c.} \quad & x \in X
\end{aligned}
\qquad (3.9)
$$

Pour plus de clarté, nous montrons les passages successifs du primal au dual pour le problème d'affectation, mais nous insistons sur le fait que cette démarche est valable pour tous les problèmes avec un saut de dualité nul. Nous proposons la fonction (3.4) comme minoration linéaire de la pseudo distance additive, avec ω dans l'ensemble des jeux de poids admissibles $\Omega' = \{\omega \in \mathbb{R}^p_{\geq} : \omega_k \leq 1\}$ et $\Omega'_> = \Omega' \cap \mathbb{R}^p_>$. La recherche de la meilleure fonction minorante se formule par problème (3.10). Le jeu de poids optimal

sera modifié légèrement, après résolution du problème (3.10), dans le cas certaines de ses composantes sont nulles.

$$\max_{\omega\in\Omega'}\min_{x\in X} \quad \sum_{k=1}^{p}\omega_k\lambda_k(\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}^k x_{ij} - \bar{z}_k) \tag{3.10}$$

Le passage à la version duale pour le problème d'affectation avec un jeu de poids connu, et l'introduction des contraintes de Ω' donnent le problème (3.11).

$$\begin{array}{lll} \max & \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j - \sum_{k=1}^{p} \lambda_k \bar{z}_k \omega_k & \\ \text{s.c.} & u_i + v_j \leq \sum_{k=1}^{p} \lambda_k c_{ij}^k \omega_k & i,\ j = 1,\ldots,n \\ & 0 \leq \omega_k \leq 1 & k = 1,\ldots,p \end{array} \tag{3.11}$$

Un nouveau passage au dual du problème (3.11) donne le problème (3.12).

$$\begin{array}{lll} \min & \sum_{p=1}^{n} u_k & \\ s.c. & \sum_{j=1}^{n} x_{ij} = 1 & i = 1\ldots n \\ & \sum_{i=1}^{n} x_{ij} = 1 & j = 1\ldots n \\ & \mu_k \geq \lambda_k(z_k(x) - \bar{z}_k) & k = 1,\ldots,p \\ & \mu_k \geq 0 & k = 1,\ldots,p \\ & x_{ij} \geq 0 & i,\ j = 1,\ldots,n \end{array} \tag{3.12}$$

Le problème (3.12) correspond à la relaxation continue de la formulation linéaire du problème (3.9) de recherche d'une solution de compromis avec une fonction scalarisante additive.

Il est aussi intéressant d'inverser cette démarche afin de définir l'ensemble des jeux de poids admissibles pour une fonction minorante, lorsque la minoration d'une fonction scalarisante n'est pas facile à définir. Prenons l'exemple de *la norme infinie* (3.13).

$$\min_{x\in X}\max_{k=1,\ldots,p}\{\lambda_k|z_k(x) - \bar{z}_k|\} \tag{3.13}$$

Le programme linéaire en nombres entiers (3.14) est équivalent au problème (3.13).

$$\begin{array}{lll} \min & \mu & \\ \text{s.c.} & \lambda_k(z_k(x) - \bar{z}_k) - \mu_k^+ + \mu_k^- = 0 & k = 1,\ldots,p \quad (3.14.a) \\ & \mu_k^+ + \mu_k^- \leq \mu & k = 1,\ldots,p \quad (3.14.b) \\ & \mu_k^+ \geq 0,\ \mu_k^- \geq 0 & k = 1,\ldots,p \\ & x \in X & \end{array}$$

Le problème dual associé à la relaxation continue du problème (3.14) s'écrit :

$$\begin{aligned}
\max \quad & \sum_{i=1}^{n} u_i + \sum_{j=1}^{n} v_j - \sum_{k=1}^{p} \lambda_k \bar{z}_k \omega_k & & \\
\text{s.c.} \quad & u_i + v_j \leq \sum_{k=1}^{p} \lambda_k c_{ij}^k \omega_k & i,\ j = 1, \ldots, n & \quad (3.15.a) \\
& -\sigma_k \leq \omega_k \leq \sigma_k & k = 1, \ldots, p & \quad (3.15.b) \\
& \sum_{k=1}^{p} \sigma_k = 1 & & \quad (3.15.c) \\
& \sigma_k \geq 0 & k = 1, \ldots, p & \quad (3.15.d)
\end{aligned}$$

où, u_i et v_j pour $i,\ j = 1, \ldots, n$ sont les variables duales associées au problème d'affectation, et ω_k et σ_k pour $k = 1, \ldots, p$ sont les variables duales correspondant respectivement aux contraintes (3.14.a) et (3.14.b).

Les contraintes (3.15.b), (3.15.c) et (3.15.d) permettent de définir l'ensemble des jeux de poids Ω'' qui engendre la famille $F_{\Omega''}$ des fonctions linéaires minorantes admissibles pour la norme infinie (3.13), où $\Omega'' = \{\omega \in \mathbb{R}^p : \sum_{k=1}^{p} |\omega_k| = 1\}$.

3.2.2 Recherche de la $k^{\text{ème}}$ meilleure solution avec minoration unique

Nous apportons dans l'algorithme 3.2 des précisions sur la méthode que nous proposons pour la recherche de la $k^{\text{ème}}$ meilleure solution, ce qui correspond à l'étape 4 de l'algorithme 3.1. Dans cette version de l'algorithme, les paramètres $\omega \in \Omega_>$ de la fonction linéaire minorante fortement monotone $\hat{f}_\omega$ (voir la fonction (3.4)) sont constants durant tout le processus d'énumération, aussi pour alléger la notation nous ferons référence à cette fonction par $\hat{f}$ dans cette section.

Nous avons présenté dans le détail la séparation de Murty-Lawler pour le problème d'affectation dans la section 1.3.2.a. Le nœud v' qui contient la $k-1^{\text{ème}}$ meilleure solution est extrait de l'ensemble des feuilles de l'arborescence $\mathcal{L}$, puis séparé en $O(n)$ nœuds fils. Un nœud v de l'arborescence comporte plusieurs informations, à savoir, les ensembles $I(v)$ et $F(v)$ des arêtes imposées et interdites, respectivement, qui définissent le sous-ensemble de solutions réalisables $X(v)$. Un nœud v contient aussi la solution optimale $x(v)$ du problème $\mathcal{P}(X(v), \hat{f})$ qu'on note aussi par $\mathcal{P}(v)$. On note $E(x)$ l'ensemble des arêtes qui composent le couplage parfait équivalent à une solution réalisable x. Les bornes décrites dans la section 2.4.3 sont appliquées à chaque nœud v de l'arborescence, et seuls les nœuds susceptibles de contenir une solution optimale sont stockés progressivement dans l'ensemble $\mathcal{L}$. Enfin, la $k^{\text{ème}}$ meilleure solution est la meilleure parmi les solutions optimales associées aux nœuds de l'ensemble $\mathcal{L}$.

L'algorithme 3.2 est une version de l'algorithme 2.4, adaptée à la recherche d'une solution de compromis pour le problème d'affectation multiobjectif. Nous discutons ici les modifications qui ont été introduites dans l'algorithme 2.4.

Algorithme 3.2 : $k^{\text{ème}}$ meilleure solution pour résoudre (3.2) par l'énumération ordonnée avec minoration unique

1 $\mathcal{L} \leftarrow \mathcal{L} \setminus \{v'\}$; $I \leftarrow I(v')$
2 **tant que** $|I| < n-1$ **faire**
3 Choisir l'arête (i^*, j^*) dans $E(x(v'))\backslash I$ à interdire
4 Définir le nœud v tel que : $I(v) \leftarrow I$ et $F(v) \leftarrow F(v') \cup \{(i^*, j^*)\}$
5 **si** $\max\{\tilde{B}(v), \hat{B}(v)\} \leq f(z(x^{**}))$ *et* $\max\{\tilde{B}(v), \hat{B}(v)\} < f(z(x^*))$ **alors**
6 Déterminer $x(v)$, une solution optimale du problème $\mathcal{P}(X(v), \hat{f})$
7 **si** $\hat{f}(v) < f(z(x^{**}))$ **alors**
8 Insérer v dans $\mathcal{L}$
9 **si** $f(z(x(v))) < f(z(x^{**}))$ **alors** $x^{**} \leftarrow x(v)$
10 $I \leftarrow I \cup \{(i^*, j^*)\}$
11 $x^k \in \arg\min_{v \in \mathcal{L}} \hat{f}(z(x(v)))$

Le test des ensembles vides de l'étape 6 de l'algorithme 2.4, est inutile puisque la relaxation principale $\mathcal{P}(X, \hat{f})$ est définie sur l'ensemble admissible du problème principal $\mathcal{P}(X, f)$.

La valeur $f(z(x^*))$ de la meilleure solution courante parmi toutes les solutions *énumérées* par l'algorithme 3.1, est une borne supérieure pour la solution optimale du problème principal $\mathcal{P}(X, f)$. Une meilleure borne supérieure est $f(z(x^{**}))$, avec x^{**} la meilleure solution courante parmi toutes les solutions réalisables *explicitées*, ce qui comprend les solutions énumérées et toutes les solutions optimales des sous-problèmes $\mathcal{P}(X(v), \hat{f})$ résolus à l'étape 6 de l'algorithme 3.2. Il est évident que $f(z(x^{**})) \leq f(z(x^*))$, mais la solution x^{**} ne doit être considérée pour la mise à jour de la meilleure solution courante x^* qu'au moment où elle est énumérée dans l'algorithme 3.1. Cela est nécessaire afin de garantir l'efficacité de x^*, puisque le point $z(x^{**})$ peut s'avérer dominé par un autre point $z(x')$ qui a la même valeur selon f, c'est-à-dire $f(z(x^{**})) = f(z(x'))$, et qui n'a pas encore été énuméré, la fonction f étant strictement -et non pas fortement- monotone.

Les tests de bornes des étapes 5 et 7 de l'algorithme 3.2 sont expliqués dans la section 2.4.3, et sont basés sur l'estimation de la valeur de la solution optimale du problème $\mathcal{P}(X(v), f)$ (voir la proposition 2), qu'on note $x^*(v)$. Rappelons que la solution $x^*(v)$ est différente de $x(v)$, la solution optimale du problème $\mathcal{P}(X(v), \hat{f})$.

étape 5 Les deux bornes $\hat{B}(v)$ et $\tilde{B}(v)$ du test de l'étape 5 sont expliquées dans la section 3.2.3.

étape 7 Comme on l'explique dans la section 2.4.3.a, la valeur $\hat{f}(z(x(v)))$ est par définition une borne inférieure de $f(z(x^*(v)))$. Aussi la valeur de la borne du test de l'étape 7, qu'on note $\hat{f}(v)$, est calculée lors de la résolution de la relaxation

$\mathcal{P}(X(v), \hat{f})$ à l'étape 6 avec une complexité en $O(n^2)$; résoudre $\mathcal{P}(X(v), \hat{f})$ revient à chercher un plus court chemin (voir la section 1.3.2.a).

Nous avons choisi des bornes simples qui s'obtiennent avec une complexité en $O(n)$ et $O(1)$ respectivement.

Nous expliquons ici les conditions des tests des étapes 5 et 7. La condition la plus simple est $\max\{\tilde{B}(v), \hat{B}(v)\} \leq f(z(x^{**}))$ et $\hat{B}(v) \leq f(z(x^{**}))$. Les inégalités strictes doivent être utilisées avec précaution puisque, comme nous l'avons déjà mentionné, il est possible que $f(z(x^*(v))) = f(z(x^{**}))$ et $x^*(v)$ domine x^{**}.
Ces tests peuvent être améliorés.

- Pour l'étape 5, lorsque $\max\{\tilde{B}(v), \hat{B}(v)\} = f(z(x^{**})) = f(z(x^*))$, le nœud v peut être coupé, ce qui justifie le second terme dans la condition de l'étape 5.

- Pour l'étape 7, on peut utiliser une inégalité stricte. En effet, lorsque $\hat{f}(z(x(v))) = f(z(x^{**}))$, on a $f(z(x^*(v))) \geq \hat{f}(z(x^*(v))) \geq \hat{f}(z(x(v))) = f(z(x^{**})) \geq \hat{f}(z(x^{**}))$. La fonction minorante $\hat{f}$ étant fortement monotone et $\hat{f}(z(x^*(v))) \geq \hat{f}(z(x^{**}))$, la solution $x^*(v)$ ne peut pas dominer la solution x^{**}, même si $f(z(x^*(v))) = f(z(x^{**}))$.

Ici aussi il est possible de tirer profit de l'intégrité des composantes des points dans l'espace des objectifs. Les tests de bornes peuvent être améliorés en remplaçant $z(x^*)$ par $y(x^*)$ comme on le montre dans la figure 3.1. Il convient alors de remplacer les instructions 5 et 7 par les lignes 5' et 7' respectivement :

5' **si** $\max\{\tilde{B}(v), \hat{B}(v)\} \leq \min\{f(z(x^{**})), f(y(x^*))\}$ **alors**

7' **si** $\hat{f}(z(x(v))) < f(z(x^{**}))$ *et* $\hat{f}(z(x(v))) \leq f(y(x^*))$ **alors**

Une dernière remarque sur la complexité des extractions et des insertions des nœuds qui ont une complexité en $O(n)$. Comme nous l'expliquons dans la section 2.2, il est plus efficace au sens de la complexité de stocker l'ensemble $\mathcal{L}$ dans un tas. La taille de l'ensemble $\mathcal{L}$ ne peut pas excéder le nombre de solutions réalisables $n!$, d'où la complexité en $O(n)$ de l'insertion et de l'extraction d'un nœud de l'ensemble $\mathcal{L}$.

Finalement il est facile de voir que la complexité de l'algorithme 3.2 est en $O(n^3)$. En effet, au plus $n-1$ fils sont créés et l'opération la plus coûteuse que subit chacun de ces nœuds est la résolution du problème $\mathcal{P}(v)$ (étape 6) qui a une complexité en $O(n^2)$.

3.2.3 Bornes $\hat{B}$ et $\tilde{B}$

Nous apportons dans cette section des précisions sur le calcul des bornes $\hat{B}(v)$ et $\tilde{B}(v)$ que nous associons au nœud v issu de la séparation du nœud v' dans l'algorithme 3.2.

3.2.3.a Borne $\hat{B}$

Soit (i^*, j^*) la nouvelle arête choisie à l'étape 3 de l'algorithme 3.2. Cette arête est posée comme interdite pour définir l'ensemble des solutions réalisables de $X(v)$.

La quantité $\hat{B}(v)$ est une borne inférieure de la valeur $\hat{f}(z(x(v)))$ de la solution optimale $x(v)$ selon la fonction minorante $\hat{f}$ (voir la section 2.4.3.b). La borne $\hat{B}(v)$ est donnée par la valeur de la détérioration de la valeur de $\hat{f}(z(x(v')))$. Rappelons que $\hat{f}(z(x(v))) = \hat{f}(z(x(v'))) + \pi$, avec π le coût du plus court chemin entre les extrémités de la nouvelle arête interdite i^* et j^* dans le graphe d'écart associé au nœud v.

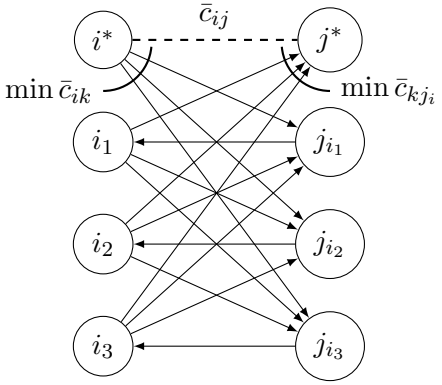


FIGURE 3.2 – Principe de la borne $\hat{B}$

La figure 3.2 montre le principe du calcul de $\hat{B}$. La valeur du plus court chemin π entre i^* et j^* est forcément minorée par la somme du poids minimum sur les arcs sortants du sommet i^* et du poids minimum sur les arcs entrant dans le sommet j^*.

$$\hat{B}(v) = \hat{f}(z(x(v'))) + \min_{j \neq j^* : (i^*, j) \notin F(v) \cup I(v)} \bar{c}_{i^* j} + \min_{i \neq i^* : (i, j^*) \notin F(v) \cup I(v)} \bar{c}_{i j^*} \tag{3.16}$$

où $\bar{c}_{ij}$ désigne le coût réduit associé à la variable x_{ij}, pour $i, j = 1, \ldots, n$. Rappelons que $\bar{c}_{ij} = c_{ij} - u_i - v_j$, avec u_i et v_j les coûts duaux optimaux associés à la solution optimale $x(v')$ et $c_{ij} = \sum_{i=1}^{p} \omega_k \lambda_k c_{ij}^k$ pour $i, j = 1, \ldots, n$.

3.2.3.b Borne $\tilde{B}$

Soit $z^*(v) = (\sum_{(i,j) \in I(v)} c_{ij}^1, \ldots, \sum_{(i,j) \in I(v)} c_{ij}^p)$ le vecteur des coûts des arêtes imposées du nœud v. La borne inférieure $\tilde{B}$ est définie pour tous les nœuds $v \in \mathcal{L}$ par : $\tilde{B}(v) = f(z^*(v))$. La borne $\tilde{B}(v)$ peut être raffinée en ajoutant à $z^*(v)$ le vecteur $(\sum_{i=1}^{n} \min_{j:(i,j) \notin I(v') \cup F(v')} c_{ij}^1, \ldots, \sum_{i=1}^{n} \min_{j:(i,j) \notin I(v') \cup F(v')} c_{ij}^p)$, ou en ajoutant les composantes du point idéal local.

Les résultats expérimentaux ont montré qu'il vaut mieux utiliser un mode de calcul facile pour $\tilde{B}$, bien que la valeur de la borne obtenue soit moins précise. De plus, le calcul de la borne $\tilde{B}$ sur la base de $z^*(v)$ permet de couper plusieurs nœuds fils à la fois. En effet, supposons que la séparation d'un nœud v consiste en la création de m nœuds fils $v_1, \ldots, v_m$. S'il existe un nœud v_j avec $j \leq m$, tel que $\tilde{B}(v_j) \geq f(z(x^*))$, alors le nœud v_j et tous les nœuds $v_{j+1}, \ldots, v_m$ peuvent être coupés. Ceci résulte du fait que la valeur de $\tilde{B}$ croît durant le branchement, puisque $I(v^j) \subset I(v^{j+1}) \subset \ldots \subset I(v^k)$ et les coûts $c_{ij}^k \in \mathbb{N}$ pour $i, j = 1, \ldots, n$ et $k = 1, \ldots, p$.

Complémentarité entre les bornes $\hat{B}$ et $\tilde{B}$

La différence principale entre les bornes $\hat{B}$ et $\tilde{B}$ réside dans le fait que $\hat{B}$ permet de couper des nœuds v dont la solution optimale $x(v)$ ne serait pas énumérée même si le test de borne $\hat{B}$ n'était pas introduit. Ceci n'est pas le cas de la borne $\tilde{B}$, qui coupe des nœuds v même si la solution $x(v)$ est candidate à l'énumération.

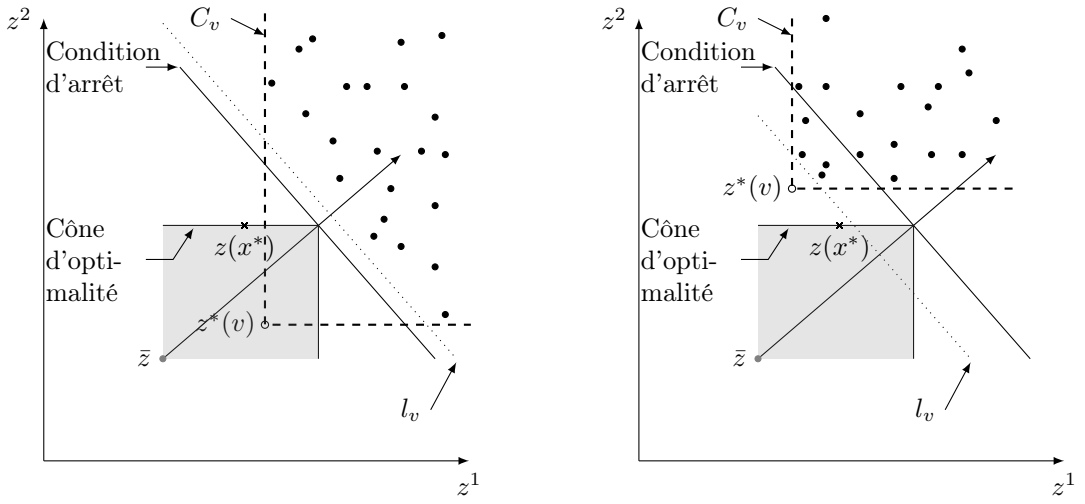


FIGURE 3.3 – Complémentarité entre les bornes $\hat{B}$ et $\tilde{B}$

La figure 3.3 montre deux situations différentes qui illustrent la complémentarité des deux bornes $\hat{B}$ et $\tilde{B}$ dans l'espace des objectifs. Dans cette figure, on représente l'image de l'ensemble des solutions réalisables $X(v)$ d'un nœud v dans l'espace des objectifs. Le point en croix $z(x^*)$ est l'image de la meilleure solution courante x^*. Le point de référence $\bar{z}$, qui n'est pas forcément réalisable, est représenté par le point gris en bas à gauche de chacune des deux sous-figures 3.3a et 3.3b et la direction de recherche est donnée par le vecteur partant de $\bar{z}$ vers le coin nord-est. Tout point dans la zone grisée, délimitée par la courbe d'iso-préférence de l'image de x^*, est meilleur que la solution x^*

selon f. Ce cône est dit *cône d'optimalité*. Il est donc trivial de déduire que si on obtient la preuve que le cône d'optimalité est vide alors la solution x^* est optimale.

Dans les deux figures, la droite en pointillés l_v représente la meilleure valeur de la borne $\hat{B}$ qui minore la valeur de la solution optimale $\hat{f}(z(x(v)))$ du nœud v. Ceci implique qu'il n'existe aucune solution de $X(v)$ dont l'image est en dessous de la droite l_v. Si l'intersection entre le demi-plan au dessus de la droite l_v et le cône d'optimalité est vide, on peut conclure que le nœud v ne contient aucune solution meilleure que x^*, i.e. $\forall x \in X(v) : f(z(x)) \geq \hat{f}(z(x)) \geq \hat{B}(v) \geq f(z(x^*))$ et le nœud v peut être coupé. La figure 3.3a décrit cette situation, ce qui n'est pas le cas de la figure 3.3b.

Le cône C_v, délimité par les lignes discontinues, représente la borne $\tilde{B}$. Les images des solutions de $X(v)$ sont à l'intérieur du cône C_v. Si l'intersection entre le cône C_v et le cône d'optimalité est vide, alors la meilleure solution courante x^* est meilleure que toutes les solutions de l'ensemble $X(v)$ selon f, i.e. pour tout $x \in X(v)$, on a $f(z(x)) \geq \tilde{B}(v) \geq f(z(x^*))$ et le nœud v peut être coupé. On peut voir cette situation dans la figure 3.3b, mais pas dans la figure 3.3a. Dans la figure 3.3b on peut voir que quelques points sont en dessous de la condition d'arrêt, représentée par la droite qui passe par le sommet du cône d'optimalité. Si la borne $\tilde{B}$ n'est pas utilisée, toutes ces solutions seraient alors énumérées.

Vu la complémentarité de ces bornes $\hat{B}$ et $\tilde{B}$, il semble intéressant de les utiliser simultanément dans l'algorithme 3.2. Ceci donne des résultats expérimentaux intéressants.

3.2.4 Ordre de séparation

La valeur de la solution optimale $\hat{f}(z(x(v)))$ dépend de l'arête (i^*, j^*) de $E(x(v')) \setminus I(v')$ qui a été interdite lors de la création du nœud v fils du nœud v'. Murty (1968) ne spécifie aucun ordre, aussi l'ordre arbitraire basé sur les indices est utilisé dans l'examen des arêtes de $E(x(v')) \setminus I(v')$.

Nous avons abordé cet aspect de la séparation lorsqu'on a présenté les améliorations apportées dans la littérature pour l'algorithme de Murty (voir la section 1.3.2.a). Nous nous inspirons des remarques de Miller et al. (1997) pour pousser les meilleures solutions à appartenir aux problèmes les plus contraints.

Dans notre procédure de Branch and Bound, nous proposons d'utiliser la borne $\hat{B}$ afin de définir l'ordre dans lequel les arêtes de $E(x(v'))\setminus I(v')$ sont considérées pour la séparation. Soit $v(i,j)$ un nœud, fils du nœud v', créé par l'interdiction de l'arête (i,j). Nous proposons donc de remplacer l'étape 3 de l'algorithme 3.2 par l'instruction 3' :

3' Choisir l'arête (i^*, j^*) à interdire, où $\hat{B}(v(i^*, j^*)) = \max_{(i,j) \in E(x(v'))\setminus I(v')} \hat{B}(v(i,j))$

Ce mode de sélection vise à choisir l'arête dont l'interdiction induit la plus grande détérioration de la valeur de la solution optimale $\hat{f}(z(x(v')))$. En effet, les solutions du premier nœud fils créé en interdisant l'arête (i^*, j^*), comme on le montre dans l'ins-

truction 3', font partie des plus mauvaises solutions de $X(v')$. Cette arête est ensuite imposée pour créer le deuxième nœud fils. Les solutions du deuxième nœud sont censées être meilleures que celles du premier nœud, et les solutions du troisième nœud meilleures que celles du deuxième nœud par le même principe. Par la répétition de ce procédé, le dernier nœud fils devrait contenir les meilleures solutions réalisables de $X(v')$, puisque les plus mauvaises ont été affectées à ses frères *aînés*.

Par ailleurs, lors de la recherche de l'arête interdite (i^*, j^*), il est possible d'imposer toutes les arêtes (i, j) pour lesquelles $\hat{B}(v(i, j)) \geq f(z(x^*))$. Cela permet d'introduire le test de la borne $\hat{B}$ prématurément et de réduire rapidement le nombre des nœuds engendrés par la séparation du nœud v'. De plus, le choix de l'arête qui maximise la valeur de la borne $\hat{B}$ devrait la rendre plus active, i.e. un nœud défini par l'interdiction de l'arête qui maximise la borne $\hat{B}$ a plus de chances d'être coupé par le test de la même borne qu'un nœud dont la nouvelle arête interdite est choisie arbitrairement.

Nous énonçons ici des suppositions, puisque le choix de l'arête comme nous le décrivons dans cette section est heuristique. Cependant, les résultats expérimentaux montrent que c'est l'une des améliorations les plus remarquables que nous ayons introduites pour le perfectionnement de l'énumération ordonnée. La complexité de l'instruction 3' est en $O(n^2)$, la complexité de l'algorithme 3.2 ($O(n^3)$) n'est donc pas modifiée.

3.3 Énumération ordonnée avec minoration variable

Nous avons proposé dans la section 3.2 une version de l'algorithme d'énumération ordonnée où une fonction minorante $\hat{f}_{\omega^*}$ est choisie avant le début de l'énumération. Le jeu de poids ω^* est obtenu en résolvant le problème (3.5). Dans cette section, nous présentons une version de cet algorithme où une fonction minorante est choisie à chaque fois qu'un nouveau nœud est créé.

Ici le jeu de poids $\omega(v) \in \Omega_>$ qui définit la fonction minorante $\hat{f}_{\omega(v)}$ pour chaque nœud v est choisi tel que la valeur $\hat{f}_{\omega(v)}(z(x(v)))$ est la plus proche possible de $f(z(x^*(v)))$, avec $x(v)$ et $x^*(v)$ les solutions optimales des problèmes $\mathcal{P}(X(v), \hat{f}_{\omega(v)})$ et $\mathcal{P}(X(v), f)$ respectivement. Un tel jeu de poids est optimal pour le problème (3.17).

$$\max_{\omega \in \Omega_>} \min_{x \in X(v)} \sum_{k=1}^{p} \omega_k \lambda_k (z_k(x) - \bar{z}_k) \tag{3.17}$$

Un raisonnement similaire à celui présenté dans la section 3.2.1 nous conduit à la conclusion que le problème (3.17), défini pour un nœud v, est équivalent à la relaxation continue de la formulation linéaire du problème $\mathcal{P}(X(v), f)$.

Les solutions sont énumérées dans un ordre croissant selon la fonction $\hat{f}(z(x)) = \sum_{v \in \mathcal{L}} 1_{X(v)}(x) \hat{f}_{\omega(v)}(z(x))$, avec $1_{X(v)}(x)$ la fonction indicatrice qui vaut 1 si x est dans $X(v)$ et 0 sinon. La fonction $\hat{f}$ n'est pas fortement monotone, donc un test de

dominance est nécessaire pour garantir l'efficacité de la solution optimale retournée. Nous montrons en nous appuyant sur la proposition 4 qu'il est suffisant de tester l'efficacité d'une solution nouvellement énumérée x^k uniquement lorsque $f(z(x^k)) = f(z(x^*))$.

Proposition 4. Soit x^* une solution optimale du problème (3.2) et soit y une solution réalisable. Si $z(y)$ domine $z(x^*)$ alors y est aussi une solution optimale du problème (3.2).

Preuve. On a $z(y)$ domine $z(x^*)$ implique que $\forall k \in \{1,\ldots,p\} : z_k(y) \leq z_k(x^*)$ et $z(y) \neq z(x^*)$. Par conséquent, $\max_{k\in\{1,\ldots,p\}}\{\lambda_k(z_k(y)-\bar{z}_k)\} \leq \max_{k\in\{1,\ldots,p\}}\{\lambda_k(z_k(x^*)-\bar{z}_k)\}$. Alors, $f(z(y)) \leq f(z(x^*)) \leq f(z(x))$ pour tout $x \in X$, puisque x^* est optimale. □

Énoncée autrement, la proposition 4 prouve qu'une solution optimale du problème (3.2) ne peut être dominée que par une autre solution optimale.

Dans le cas où $\hat{f}$ n'est pas fortement monotone, les lignes 6 et 7 de l'algorithme 3.1 doivent être remplacées par les lignes 6' et 7' respectivement.

6' **si** $(f(z(x^k)) < f(z(x^*))$ *ou* $(f(z(x^k)) = f(z(x^*))$ *et* $z(x^k) \leq z(x^*))$ **alors** $x^* \leftarrow x^k$

7' **jusqu'à**$(\hat{f}(z(x^k)) > f(z(x^*))$ ou $\hat{X} \setminus \hat{Y} = \emptyset)$

L'étape 6' corrige la mise à jour de la meilleure solution courante et l'étape 7' retarde la condition d'arrêt en forçant l'algorithme 3.1 à énumérer toutes les solutions optimales.

3.3.1 Recherche de la $k^{\text{ème}}$ meilleure solution avec minoration variable

Dans cette section, nous présentons l'algorithme 1 qui résume les étapes principales du calcul de la $k^{\text{ème}}$ meilleure solution pour la version de l'énumération ordonnée lorsque la fonction minorante est variable. Les étapes 1 à 7 et celles de 9 et 14 de l'algorithme 3.3 sont les mêmes instructions de l'algorithme 3.2.

La borne $\hat{B}(v)$ et la borne $\hat{f}_{\omega(v')}(v)$ s'obtiennent avec les paramètres du nœud père v'. En effet, la borne $\hat{B}(v)$ est donnée par les coûts réduits associés à la solution optimale du nœud père avec le jeu de poids $\omega(v')$. La valeur de la borne $\hat{f}_{\omega(v')}(v)$ est la valeur optimale du problème $\mathcal{P}(X(v), \hat{f}_{\omega(v')})$.

Si les conditions des tests des étapes 5 et 7 sont vérifiées, alors un nouveau jeu de poids $\omega(v)$ est déterminé par la résolution du problème (3.17).

La solution optimale $x(v)$ et sa valeur $\hat{f}_{\omega(v)}(v)$ sont censés être obtenues par la résolution du problème $\mathcal{P}(X(v), \hat{f}_{\omega(v)})$. La valeur $\hat{f}_{\omega(v)}(v)$ sera utilisée pour un dernier test de borne (étape 9). Nous montrons dans la section 3.3.2 que la valeur de la borne $\hat{f}_{\omega(v)}(v)$ peut être obtenue grâce à l'optimum du problème (3.17), aussi la détermination de la solution optimale du problème $\mathcal{P}(X(v), \hat{f}_{\omega(v)})$ peut être différée jusqu'à l'étape 10.

Dans la version de l'énumération ordonnée que nous présentons dans cette section, toutes les solutions optimales du problème principal $\mathcal{P}(X, f)$ doivent être énumérées, et cela parce que la fonction $\hat{f}$ n'est pas fortement monotone. C'est pour cette raison que

Algorithme 3.3 : $k^{\text{ème}}$ meilleure solution pour résoudre (3.2) par l'énumération ordonnée avec minoration variable

1 $\mathcal{L} \leftarrow \mathcal{L} \setminus \{v'\}$ $I \leftarrow I(v)$
2 **tant que** $|I| < n-1$ **faire**
3 Choisir l'arête (i^*, j^*) dans $E(x(v'))\backslash I$ à interdire
4 Définir le nœud v tel que : $I(v) \leftarrow I$ et $F(v) \leftarrow F(v') \cup \{(i^*, j^*)\}$
5 **si** $\max\{\tilde{B}(v), \hat{B}(v)\} \leq f(z(x^{**}))$ **alors**
6 Résoudre le problème $\mathcal{P}(X(v), \hat{f}_{\omega(v')})$
   // $\hat{f}_{\omega(v')}(v) = \min \hat{f}_{\omega(v')}(x) : x \in X(v)$
7 **si** $\hat{f}_{\omega(v')}(v) \leq f(z(x^{**}))$ **alors**
8 Déterminer $\omega(v)$, un jeu de poids optimal du problème (3.17)
9 **si** $\hat{f}_{\omega(v)}(v) \leq f(z(x^{**}))$ **alors**
10 Déterminer $x(v)$, une solution optimale du problème $\mathcal{P}(X(v), \hat{f}_{\omega(v)})$
11 Insérer v dans $\mathcal{L}$
12 **si** $f(z(x(v))) < f(z(x^{**}))$ **alors** $x^{**} \leftarrow x(v)$
13 $I \leftarrow I \cup \{(i^*, j^*)\}$
14 $x^k \in \arg\min_{v \in \mathcal{L}} \hat{f}(z(x(v)))$

les inégalités des tests de l'algorithme 3.3 sont larges, contrairement aux tests de bornes de l'algorithme 3.2 que nous avons pu affiner.

La complexité de l'algorithme 3.3 dépend de l'étape 8 où un programme linéaire est résolu pour trouver le jeu de poids $\omega(v)$. Nous faisons appel à l'algorithme *dual du simplexe* qui est exponentiel, notamment lorsque le programme linéaire est dégénéré comme c'est le cas du problème d'affectation. Bien que du point de vue de la pratique, les méthodes du simplexe soient très efficaces, les résultats expérimentaux confirment la lenteur du temps de réponse de l'algorithme 3.3. Cependant, nous montrerons des instances particulières dans la section 3.4 qui ne peuvent être résolues qu'à l'aide de l'algorithme 3.3.

3.3.2 Résolution du problème d'affectation associé à un nœud de l'arborescence dans la version avec minoration variable

Lorsqu'un jeu de poids $\omega(v)$ est déterminé pour un nœud v, le nouveau problème d'affectation $\mathcal{P}(X(v), \hat{f}_{\omega(v)})$ qui lui est associé possède une fonction objectif différente de celle de nœud père. Ici le recours à la ré-optimisation par le calcul d'un plus court chemin, comme c'est le cas dans la version avec minoration unique, n'est donc plus envisageable.

Afin d'éviter la résolution du problème $\mathcal{P}(X(v), \hat{f}_{\omega(v)})$ comme un problème d'affectation classique avec une complexité en $O(n^3)$, nous montrons dans cette section une ma-

nière d'utiliser les coûts réduits, obtenus lors de la recherche du jeu de poids optimal $\omega(v)$ (étape 8), afin de déterminer une solution optimale $x(v)$ du problème $\mathcal{P}(X(v), \hat{f}_{\omega(v)})$ (étape 10).

Sans perte de généralité, nous ignorons dans les explications à suivre les contraintes relatives aux arêtes imposées $I(v)$ et interdites $F(v)$, et ce afin de simplifier les notations. Nous avons montré dans la section 3.2.1 que le problème (3.17) est équivalent au problème (3.18) et à sa version duale (3.19).

$$\begin{array}{lll} \min & \mu & \\ \text{s.c.} & \mu \geq \lambda_k(\sum\limits_{i=1}^{n}\sum\limits_{j=1}^{n} c_{ij}^k x_{ij} - \bar{z}_k) & k = 1, \ldots, p \\ & \sum\limits_{j=1}^{n} x_{ij} = 1 & i = 1 \ldots n \\ & \sum\limits_{i=1}^{n} x_{ij} = 1 & j = 1 \ldots n \\ & x_{ij} \geq 0 & i,\ j = 1, \ldots, n \end{array} \tag{3.18}$$

$$\begin{array}{lll} \max & \sum\limits_{i=1}^{n} u_i + \sum\limits_{j=1}^{n} v_j - \sum\limits_{k=1}^{p} \lambda_k \bar{z}_k \omega_k(v) & \\ \text{s.c.} & u_i + v_j \leq \sum\limits_{k=1}^{p} \lambda_k c_{ij}^k \omega_k(v) & i,\ j = 1, \ldots, n \\ & \sum\limits_{k=1}^{p} \omega_k(v) = 1 & \\ & \omega_k(v) \geq 0 & k = 1, \ldots, p \end{array} \tag{3.19}$$

Il est facile de montrer qu'à l'optimum, les valeurs optimales des variables duales u_i et v_j sont aussi optimales pour la formulation duale du problème d'affectation $\mathcal{P}(X(v), \hat{f}_{\omega(v)})$. En effet, le problème (3.19) avec un vecteur de poids connu $\omega(v)$ est équivalent à la formulation en programme linéaire duale d'un problème d'affectation avec les coûts $\sum\limits_{k=1}^{p} \lambda_k c_{ij}^k \omega_k(v)$.

Par le principe fort de la dualité on peut déduire que la valeur $\hat{f}_{\omega(v)}(v)$ d'une solution optimale du problème $\mathcal{P}(X(v), \hat{f}_{\omega(v)})$ est égale à la valeur optimale du problème (3.19). Ceci donne la possibilité de placer la recherche d'une solution optimale $x(v)$ à différentes positions. En effet, la solution $x(v)$ peut être déterminée soit juste après le test de la borne $\hat{f}_{\omega(v)}$ (étape 11), ou bien au moment où la solution est énumérée (si elle est énumérée). La première possibilité permet d'avoir une bonne borne supérieure $f(z(x^{**}))$ mais ralentit l'algorithme 3.3. La seconde possibilité améliore les performances de la recherche de la $k^{\text{ème}}$ meilleure solution mais produit une borne supérieure moins précise. Les expérimentations montrent qu'il est préférable de mettre à jour fréquemment la *meilleure solution explicitée*, notée x^{**}. Aussi la solution $x(v)$ est calculée à l'étape 11 de l'algorithme 3.3.

La solution optimale du programme linéaire primal (3.18) n'est pas forcément entière. Afin de déterminer une solution entière pour $\mathcal{P}(X(v), f_{\omega(v)})$ il est possible de procéder à l'aide de l'un des algorithmes du problème d'affectation avec une complexité en $O(n^3)$. Nous proposons de déterminer la solution $x(v)$ avec moins d'efforts.

L'intuition de l'approche que nous proposons émane du principe des écarts complémentaires ($\bar{c}_{ij}x_{ij} = 0$). Les n arêtes qui forment l'affectation optimale peuvent être sélectionnées parmi celles qui correspondent à des coûts réduits nuls. Le problème de la recherche d'une solution entière pour le problème $\mathcal{P}(X(v), f_{\omega(v)})$ dans ce cas se réduit à la recherche d'un couplage parfait dans le graphe biparti $G(v)(U, V; E(v))$, où U et V sont les deux ensembles de n sommets du problème d'affectation original et $E(v)$ l'ensemble des arêtes (i, j) telles que le coût réduit $\bar{c}_{ij} = 0$ pour $i, j = 1, \ldots, n$. Remarquons que l'ensemble des arêtes imposées $I(v)$ est forcément partie de $E(v)$. Une manière d'interdire les arêtes de l'ensemble $F(v)$ est de poser $c_{ij} = +\infty$, par conséquent pour tout $(i, j) \in F(v) : \bar{c}_{ij} > 0$. Les arêtes de $F(v)$ ne font donc pas partie de $E(v)$.

Nous utilisons l'algorithme proposé par Hopcroft et Karp (1973) qui permet de trouver un couplage maximal (i.e. qui contient le maximum d'arêtes) dans un graphe biparti avec une complexité en $O(\sqrt{n}m)$, où n est le nombre de sommets et m le nombre d'arêtes. Un principe similaire est utilisé dans les méthodes dites de *cost-scaling* et plus généralement dans les méthodes duales pour le problème d'affectation. Les versions les plus simples de ces méthodes peuvent être consultés dans Gabow (1983); Gabow et Tarjan (1991) ou dans le livre de Burkard et al. (2009).

3.4 Résultats expérimentaux

Nous présentons dans cette section les résultats expérimentaux afin de montrer l'efficacité des procédures que nous proposons.

Dans un premier temps, nous montrons l'impact de nos améliorations sur l'algorithme par énumération ordonnée avec minoration unique, à savoir :

- le choix de ω dans $\Omega_>$ afin de déterminer la fonction linéaire minorante $\hat{f}$;
- l'introduction des bornes $\tilde{B}$ et $\hat{B}$ présentées dans la section 3.2.3 ;
- le choix de l'ordre de séparation discuté dans la section 3.2.4.

Nous montrons à l'issue de l'interprétation de ces résultats l'efficacité de la version avec minoration unique qui inclut toutes les améliorations.

Dans un second temps, nous discutons de l'intérêt d'utiliser la variante avec une fonction minorante variable, tant les résultats expérimentaux sont nettement en faveur de la version avec minoration unique.

Nous spécifions dans la section 3.4.3 les caractéristiques de deux familles d'instances particulièrement difficiles.

Remarques générales

Les instances aléatoires que nous utilisons dans nos expérimentations sont composées de coûts entiers et positifs c_{ij}^k générés selon une loi uniformément distribuée sur l'ensemble $\{1, \ldots, 100\}$. Le point de référence $\bar{z}$ est choisi aléatoirement entre le point idéal z^* et une approximation du point nadir z^N. Le point z^N est obtenu grâce à un tableau de gains contenant l'image des p solutions qui ont permis de déterminer le point idéal. La direction de recherche λ est *centrée* dans l'espace des points non-dominés, i.e. $\lambda_k = \frac{1}{z_k^N - z_k^*}$, $k = 1, \ldots, p$. Les résultats que nous présentons dans les sections 3.4.1 et 3.4.2 sont des moyennes sur 20 instances de même taille en utilisant la même version de l'algorithme.

Nous utilisons aussi des instances différentes de celles que nous venons de décrire. Dans ce cas, un avertissement et une caractérisation précise de ces instances seront donnés.

Nous disposons pour ces tests d'une machine dotée d'un processeur Intel(R) Xeon(R) E5507 2.27 GHz et d'une RAM de 4 GB, et qui tourne sous le système d'exploitation Debian GNU/Linux 6.0. Les algorithmes sont codés en langage C et les fichiers binaires sont générés en utilisant le compilateur gcc avec l'option d'optimisation -O3.

3.4.1 Conséquences des améliorations sur la procédure d'énumération ordonnée avec minoration unique

Les expérimentations concernent dans cette section l'algorithme 3.1 utilisant l'algorithme 3.2 pour la recherche de la $k^{\text{ème}}$ meilleure solution. Les performances de cette procédure seront testées à différents niveaux de l'algorithme (initialisation, Branch and Bound) et à l'aide de différentes mesures (temps CPU, nombre de solutions énumérées et taille de l'arborescence de recherche).

3.4.1.a Phase d'initialisation

Dans cette section, nous étudions l'impact du mode de choix d'une fonction linéaire minorante dans la famille $F_{\Omega_>}$. Pour ce faire, nous comparons dans la table 3.1 les moyennes du temps de calcul de l'algorithme 3.1. Ici la fonction minorante $\hat{f}$ de la relaxation principale est définie par un jeu de poids ω :

- avec des composantes équivalentes : $\omega_k = \frac{1}{p}$ pour $k = 1, \ldots, p$ (colonne ω_{eq}) ;
- choisi aléatoirement dans $\Omega_>$ (colonne ω_{al}) ;
- solution optimale du problème (3.5) (colonne ω^*).

Dans nos tests expérimentaux, le vecteur ω^* est obtenu en résolvant le programme linéaire (3.8) à l'aide du solver CPLEX (IBM ILOG CPLEX v12.4). Les comparaisons entre les différents modes de choix de la fonction minorante conduisent aux mêmes conclusions quelle que soit la variante de l'algorithme 3.2 utilisée (par exemple avec ou sans les tests de bornes). Nous montrons donc dans la table 3.1 le temps moyen de calcul d'une version qui contient tous les tests de bornes $\hat{B}$, $\tilde{B}$ et $\hat{f}$ et où la séparation est réorganisée. Précisons qu'à cause des temps de calcul très élevés de certaines instances (voir les colonnes ω_{eq} et ω_{al}), ces résultats sont les moyenne du temps CPU de 5 instances aléatoires en secondes.

Par ailleurs, le symbole '_' dans la table 3.1 indique que le temps de calcul ne peut être récupéré pour au moins une instance à cause de la saturation de la mémoire.

n	p	ω_{eq}	ω_{al}	ω^*
	2	0.158	0.128	0.000
10	3	0.146	0.001	0.000
	4	0.490	0.242	0.000
	2	141.352	13.160	0.004
15	3	1 409.064	209.900	0.004
	4	1 368.496	994.660	0.006
	2	745.578	949.781	0.004
20	3	7 905.421	_	0.007
	4	_	_	0.008

TABLE 3.1 – Comparaison de différents modes d'initialisation (temps CPU en secondes)

Les écarts entre les temps CPU ne laissent aucun doute sur l'intérêt du calcul de ω^* dans la phase d'initialisation. De plus les variantes utilisant ω_{eq} et ω_{al} sont instables. De manière étonnante, la comparaison du temps de calcul moyen pour deux instances de tailles différentes pour les variantes qui utilisent ω_{eq} ou ω_{al} peut être en faveur de l'instance la plus grande. Par exemple, pour ω_{eq} et $n = 15$, le temps moyen de calcul est de 1409s avec trois objectifs alors qu'il est de 1368s avec quatre objectifs. Il est aussi surprenant de voir que la variante utilisant un jeu de poids ω_{al}, choisi au hasard, puisse être meilleure que la variante utilisant ω_{eq} (voir par exemple $n = 15$ et $p = 3$).

Nous avons observé, dans des résultats expérimentaux que nous ne montrons pas ici, que lorsque le point de référence est le point idéal z^*, la variante avec ω_{eq} est plus compétitive mais reste dominée par la variante ω^*. Cela est dû vraisemblablement au fait que la direction de recherche λ suffit pour définir la pente de l'hyperplan linéaire qui représente la fonction linéaire minorante $\hat{f}$ pour ce type d'instances.

Ces premiers résultats nous permettent de proposer la meilleure variante pour la phase d'initialisation qui consiste à faire le calcul de ω^*. Dans les résultats à suivre la fonction linéaire minorante est toujours définie par le jeu de poids ω^*.

3.4.1.b Introduction de bornes et changement de l'ordre de la séparation

Dans cette section nous mesurons l'impact de l'introduction des bornes et de la réorganisation du branchement sur l'efficacité de l'énumération ordonnée avec minoration unique. Les variantes de l'algorithme 3.2 sont désignées en utilisant trois caractères. Le premier caractère vaut $\hat{B}$ si la variante utilise le test de la borne $\hat{B}$, et il vaut $-$ sinon. De la même manière le deuxième caractère vaut $\tilde{B}$ ou $-$ et indique si la variante utilise ou non le test de la borne $\tilde{B}$. Le troisième caractère est R si la nouvelle arête interdite à l'étape 3 de l'algorithme 3.2 est choisie telle qu'elle maximise la borne $\hat{B}$ comme on le montre dans la section 3.2.4. Ce troisième caractère vaut $-$ si le choix de l'arête interdite est arbitraire. Par exemple, la variant '$\hat{B} - R$' utilise le test de la borne $\hat{B}$ et réordonne la séparation mais n'utilise pas le test de la borne $\tilde{B}$.

Le test de la borne $\hat{f}$ est utilisé dans toutes les variantes que nous présentons, car la valeur de la borne $\hat{f}$ s'obtient sans aucun effort de calcul. En effet, la borne $\hat{f}(v)$ pour un nœud v est la valeur de la solution optimale du problème $\mathcal{P}(X(v), \hat{f})$, aussi aucune opération propre au calcul de la borne $\hat{f}(v)$ n'est réalisée.

Dans toutes les versions, nous tirons avantage de l'intégrité des coûts par l'amélioration de la condition d'arrêt (voir la section 3.2) et les tests de bornes (voir la fin de section 3.2.2).

Nous étudions l'impact de l'introduction de ces perfectionnements sur l'algorithme 3.1 en termes :

- de nombre de solutions énumérées avant d'atteindre la condition d'arrêt ;
- de taille de l'arborescence de recherche à la fin de l'algorithme ;
- et de temps de calcul.

Nous présentons les résultats obtenus sur 20 instances aléatoires pour tout n dans $\{50, 100, 200\}$ et p dans $\{2, 3, 4\}$.

Nombre de solutions énumérées

La table 3.2 montre le nombre moyen des solutions énumérées durant l'exécution de l'algorithme 3.1. Une partie de ces solutions est énumérée après que la solution optimale ait été retrouvée. Ces solutions sont déterminées uniquement pour prouver l'optimalité de la solution retournée x^*. Les valeurs entre parenthèses dans la table 3.2 sont les proportions de ces solutions par rapport à toutes les solutions énumérées.

Rappelons que la borne $\hat{B}$ n'a pas d'impact sur le nombre de solutions énumérées (voir les sections 2.4.3.b et 3.2.3). Les versions qui diffèrent uniquement par l'utilisation de la borne $\hat{B}$ partagent donc les mêmes colonnes dans la table 3.2.

n	p	$---$ $\hat{B}--$	$-\tilde{B}-$ $\hat{B}\tilde{B}-$	$--R$ $\hat{B}-R$	$-\tilde{B}R$ $\hat{B}\tilde{B}R$
	2	41 (71%)	40 (73%)	40 (73%)	39 (72%)
50	3	662 (53%)	653 (53%)	733 (58%)	570 (62%)
	4	4 684 (89%)	4 655 (89%)	4 632 (89%)	3 505 (89%)
	2	36 (86%)	36 (86%)	57 (91%)	56 (91%)
100	3	2 380 (78%)	2 375 (78%)	2 084 (75%)	1 756 (75%)
	4	33 236 (92%)	33 199 (92%)	31 176 (91%)	21 833 (90%)
	2	99 (92%)	99 (92%)	118 (92%)	110 (94%)
200	3	4 384 (86%)	4 384 (86%)	3 904 (84%)	3 205 (85%)
	4	166 318 (90%)	166 199 (90%)	151 816 (89%)	122 827 (89%)

TABLE 3.2 – Nombre de solutions énumérées (pourcentage des solutions énumérées après l'obtention de la solution optimale)

On peut observer que les améliorations $\tilde{B}$ et R sont complémentaires. Introduites séparément ces améliorations réduisent légèrement le nombre de solutions énumérées, alors que la combinaison des deux améliorations engendre un gain plus significatif dans le nombre de solutions énumérées. La ligne du tableau correspondant à $n = 100$ et $p = 4$ par exemple, montre cette complémentarité.

On peut aussi observer que la séparation réordonnée peut détériorer les performances de notre procédure pour les instances en deux objectifs qui, de toute manière, sont faciles à résoudre. Lorsque le nombre des objectifs p augmente, l'utilisation de la borne $\tilde{B}$ et la réorganisation de la séparation nous permettent de réduire sensiblement le nombre de solutions énumérées par l'algorithme 3.1.

Il est aussi remarquable qu'une grande majorité des solutions soit énumérée après l'obtention de la solution optimale. Trouver la solution optimale est généralement plus facile que de prouver son optimalité. Par conséquent, lorsque l'optimalité n'est pas requise, il peut être intéressant de considérer une version où le processus d'énumération est stoppé avant l'obtention de la condition d'arrêt et la meilleure solution courante est retournée comme solution approchée. La valeur de la dernière solution énumérée donne l'écart maximum entre la valeur de la solution retournée et la valeur optimale. Cette démarche est d'autant plus intéressante si on prend en considération le fait que le principe de l'énumération ordonnée avec une fonction minorante fortement monotone garantit l'efficacité de la meilleure solution courante (voir la proposition 4), ainsi on a la garantie que la solution approchée retournée est efficace.

Taille de l'arborescence de recherche

Une mise en œuvre idéale d'un algorithme d'énumération ordonnée sous la forme d'un Branch and Bound conduit à la création d'autant de nœuds que de solutions énumérées. La table 3.3 montre le nombre de nœuds de l'arborescence qui ont été créés par l'algorithme 3.2 sans avoir été coupés par les bornes $\hat{B}$ ou $\tilde{B}$. Pour chacun de ces nœuds, une solution optimale a été déterminée. Par conséquent le nombre de nœuds créés correspond au nombre de fois où notre procédure a fait appel à un algorithme de recherche de plus court chemin avec une complexité en $O(n^2)$. Cette indication est très importante puisque l'opération la plus coûteuse en temps à chaque itération de l'algorithme 3.2 est bien la recherche d'un plus court chemin. Notons que ce nombre est plus grand que le nombre de solutions énumérées, qui correspond au nombre de nœuds de l'arborescence qui ont été séparés durant l'énumération ordonnée.

n	p	$---$	$-\tilde{B}-$	$\hat{B}--$	$\hat{B}\tilde{B}-$	$--R$	$-\tilde{B}R$	$\hat{B}-R$	$\hat{B}\tilde{B}R$
	2	893	857	196	196	297	272	46	45
50	3	16 194	14 255	3 276	3 243	5 338	3 985	783	618
	4	118 115	103 316	23 283	23 058	32 697	22 590	4 885	3 753
	2	1 422	1 391	295	295	378	358	68	67
100	3	118 996	111 283	23 124	23 084	24 672	19 555	2 285	1 956
	4	1 758 476	1 606 686	303 129	302 470	332 779	234 149	33 620	24 231
	2	9 407	9 119	1 754	1 753	1 458	1 282	145	137
200	3	456 064	433 016	81 086	80 980	62 870	48 416	4 308	3 604
	4	17 022 015	16 178 901	2 739 754	2 738 772	2 359 308	1 784 570	166 553	137 378

TABLE 3.3 – Taille de l'arborescence

Par contraste avec les précédentes observations sur le nombre de solutions énumérées, la borne $\tilde{B}$ a moins d'impact sur la taille de l'arborescence.

La borne $\hat{B}$ est plus appropriée pour réduire la taille de l'arborescence et ainsi accélérer la recherche de la $k^{\text{ème}}$ meilleure solution (voir la section 2.4.3.b), particulièrement lorsqu'elle est combinée avec le changement d'ordre de séparation R. Cela est dû au fait que le changement de l'ordre de séparation utilise cette même borne $\hat{B}$ afin de choisir l'arête interdite (voir la section 3.2.4), ce qui augmente la possibilité d'élagage d'un nœud avec le test de la borne $\hat{B}$.

La variante qui utilise toutes les améliorations ($\hat{B}\tilde{B}R$) conduit à la création d'une arborescence de recherche dont la taille est très proche du nombre de solutions énumérées. Par conséquent très peu de problèmes d'affectation linéaire sont résolus inutilement. Dans ce sens cette variante est quasiment optimale. Pour $p \geq 3$, le nombre de nœuds résolus pour la variante $\hat{B}\tilde{B}R$ est inférieur au nombre de solutions énumérées avec la variante dans laquelle nous n'avons introduit aucune amélioration ($---$ dans la table 3.2). Cela montre que même une mise en œuvre idéale de l'énumération ordonnée sans les améliorations que nous proposons est moins intéressante que la variante $\hat{B}\tilde{B}R$.

Enfin nous commentons le choix de la l'arité du tas dans lequel sont stockés les nœuds de l'ensemble $\mathcal{L}$. Ce paramètre, que nous notons d, a des implications sur les performances des algorithmes. En effet, la complexité précise de l'insertion d'un élément dans un tas de taille t est donnée en $O(\log_d t)$, où $\log_d$ est le logarithme en base d. L'extraction du nœud qui contient la solution de valeur minimum du même tas s'effectue avec une complexité en $O(d \log_d t)$. Il semble qu'il est préférable de choisir des tas d'arité élevée (i.e. d grand) lorsque le nombre des insertions est plus important que le nombre des extractions, ce qui est le cas pour les algorithmes basés sur le schéma de Murty-Lawler. En effet, augmenter l'arité d permet de diminuer la complexité d'insertion qui est l'opération la plus fréquente. Nous proposons de choisir l'arité d, tel qu'il engendre un équilibre entre l'effort des extractions et l'effort des insertions.

Néanmoins, nous avons varié l'arité des tas que nous utilisons dans les expérimentations et il semble que les tas d'arité faible ($2 \leq d \leq 5$) conviennent plus aux variantes performantes qui contiennent nos améliorations. Cela s'explique par les résultats des tables 3.2 et 3.3, qui montrent que le nombre de solutions énumérées, donnant le nombre des nœuds extraits, est presque équivalent à la taille des arborescences qui correspond au nombre de nœuds insérés. Des questions similaires sur la largeur des tas ont été soulevées par Tarjan (1983) dans ses commentaires sur les implémentations par les tas des algorithmes de plus courts chemins.

Temps de calcul

Les deux tables précédentes montrent que l'utilisation des améliorations permet de diminuer de manière significative le nombre de solutions énumérées et la taille des arborescences de recherche. Cependant ces améliorations nécessitent des opérations supplémentaires qui peuvent pénaliser le temps CPU de l'algorithme 3.1. Nous présentons ici les temps CPU moyens des différentes versions de notre procédure avec une fonction minorante unique.

À notre connaissance il n'existe pas d'algorithme proposé dans la littérature pour résoudre le problème de la recherche d'une meilleure solution de compromis pour le problème d'affectation multiobjectif. Nous comparons donc les différentes versions de notre procédure à la résolution de programme linéaire mixte en nombres entiers (3.3). Rappelons que la résolution du problème (3.3) ne garantit pas l'efficacité de la solution optimale, et c'est pour cela que nous présentons aussi les résultats de sa version augmentée avec $\rho = 10^{-4}$ (voir la fonction (1.5) de la section 1.1.4). Dans nos expérimentations, le problème (3.3) et sa version augmentée sont résolus avec le solver CPLEX (IBM ILOG CPLEX v12.4) et sont notés dans la table 3.4 CPLEX et CPLEX$_\rho$. On présente dans la table 3.4 le temps CPU moyen pour la résolution de 20 instances aléatoires.

Il apparaît clairement que la version $\hat{B}\tilde{B}R$ est meilleure que toutes les autres variantes, et ce en incluant les deux versions CPLEX et CPLEX$_\rho$. En scrutant les temps de calculs instance par instance (dans les résultats bruts que nous ne présentons pas ici), la version $\hat{B}\tilde{B}R$ domine toutes les autres versions pour chaque instance, excepté les

n	p	$---$	$-\tilde{B}-$	$\hat{B}--$	$\hat{B}\tilde{B}-$	$--R$	$-\tilde{B}R$	$\hat{B}-R$	$\hat{B}\tilde{B}R$	CPLEX	CPLEX$_\rho$
	2	0.06	0.05	0.04	0.05	0.05	0.05	0.05	0.04	0.20	0.23
50	3	0.23	0.19	0.10	0.11	0.10	0.09	0.07	0.06	0.43	0.53
	4	1.15	1.22	0.47	0.56	0.23	0.24	0.14	0.13	0.83	0.96
	2	0.28	0.28	0.20	0.23	0.28	0.28	0.24	0.23	0.89	1.03
100	3	3.35	3.64	1.39	1.32	0.51	0.61	0.37	0.37	2.05	2.84
	4	41.75	46.81	15.85	14.62	3.78	3.55	1.55	1.33	5.82	7.31
	2	2.02	2.21	1.44	1.24	1.58	1.56	1.07	1.02	3.86	5.97
200	3	40.98	47.08	15.58	12.73	3.44	3.39	1.53	1.45	12.03	14.82
	4	1 293.78	1 248.22	464.89	371.41	48.05	43.62	14.70	13.24	101.52	117.55

TABLE 3.4 – Temps CPU moyens pour différentes variantes (secondes)

petites instances qui, de toute manière, sont résolues en quelques millisecondes. De plus, chaque perfectionnement introduit tend à améliorer les performances de la procédure d'énumération ordonnée.

Nous obtenons le gain le plus important en temps de calcul grâce au changement de l'ordre de séparation R, comme on peut l'observer en comparant l'écart entre la colonne $---$ et $\hat{B}\tilde{B}-$ avec l'écart entre les colonnes $---$ et $--R$. La borne $\hat{B}$ est très importante pour le temps de calcul, puisque c'est grâce à elle qu'il est possible d'éviter la résolution des problèmes associés aux nouveaux nœuds. L'impact de la borne $\tilde{B}$ semble être moins significatif sur le temps de calcul, mais son apport est plus remarquable lorsque le nombre d'objectifs est plus important.

Comme on l'a déjà mentionné dans la section 3.1, les meilleurs algorithmes de génération de l'ensemble de solutions efficaces pour le problème d'affectation multiobjectif parviennent à résoudre des instances avec $n = 100$ pour $p = 2$ et $n = 50$ pour $p = 3$. Nos résultats montrent que la recherche directe d'une solution de bon compromis est plus efficace que les approches en deux phases.

En marge de ces résultats sur les performances des algorithmes, nous mentionnons quelques observations liées au choix du paramètre ρ de la partie linéaire de la pseudo-distance augmentée. Avant de décider de la valeur de $\rho = 10^{-4}$, nous avons procédé à plusieurs essais pour des valeurs entre 10^{-6} et 10^{-2}. Nous avons remarqué qu'avec le paramètre ρ tel que $10^{-4} < \rho \leq 10^{-2}$, les solutions optimales de la version augmentée du problème (3.3) ne sont pas optimales pour la version simple (non-augmentée) du même problème, et cela pour plusieurs instances. Autrement dit, le terme linéaire qui est ajouté à la pseudo-distance uniquement pour garantir l'efficacité de la solution optimale perturbe le problème et change la valeur de l'optimum. Ce défaut de la pseudo-distance n'est pas le plus grave, puisqu'il suffit de résoudre le problème simple et de comparer les deux optima. D'un autre coté, avec $\rho = 10^{-6}$, nous nous sommes aperçus que la solution optimale d'une instance est dominée, et ce grâce au fait que la version avec minoration variable énumère toutes les solutions optimales. Dans ce cas l'augmentation n'accomplit pas la seule fonction pour laquelle elle a été introduite, à savoir, assurer l'efficacité de la

solution optimale. De plus, vérifier l'efficacité d'une solution est plus difficile que résoudre le problème (3.3), puisque cela revient à résoudre un nouveau problème très proche du problème (3.3) avec p contraintes supplémentaires dites de *budget*. Il est connu -et nous l'avons abordé brièvement à la fin de section 1.1.3- que l'ajout de contraintes dans un problème augmente sa difficulté.

C'est pour cela que nous critiquons la pseudo-distance augmentée et que nous pensons que bien qu'elle soit théoriquement intéressante, la réalité des problèmes pratiques du calcul montre ses limites.

3.4.1.c Performances de la meilleure variante sur de grandes instances

Dans cette section nous discutons l'efficacité de la meilleure variante $\hat{B}\tilde{B}R$ lorsque la taille de l'instance n ou le nombre d'objectifs p augmente. D'un côté nous résolvons une petite instance ($n = 50$) en augmentant le nombre des objectifs $p = 2, \ldots, 7$, et d'un autre côté nous relevons les temps de calcul d'instances dont la taille varie entre 50 et 500 avec deux objectifs. Les résultats que nous présentons dans la table 3.5 sont respectivement, la moyenne, le minimum et le maximum du temps de calcul en secondes. Précisons que les versions CPLEX et CPLEX$_\rho$ ne sont pas indiquées ici puisqu'elles ne peuvent pas résoudre toutes les instances de cette taille.

n	p	moyenne	min	max
50	2	0.04	0.04	0.06
	3	0.06	0.04	0.23
	4	0.13	0.06	0.56
	5	5.52	0.08	64.34
	6	82.70	0.06	1 213.32
	7	187.09	0.07	1 021.23

p	n	moyenne	min	max
2	50	0.04	0.04	0.06
	100	0.23	0.21	0.38
	200	1.02	0.92	1.97
	300	2.83	1.86	4.03
	400	7.83	5.46	11.55
	500	16.08	11.54	23.04

TABLE 3.5 – Temps CPU moyens de la variante $\hat{B}\tilde{B}R$ pour les grandes instances (secondes)

Ces résultats montrent l'efficacité de notre procédure avec minoration stable lorsqu'elle utilise toutes les améliorations ($\hat{B}\tilde{B}R$). Elle nous permet en effet de résoudre des instances qui, à notre connaissance, n'étaient pas résolues jusqu'à présent.

Dans d'autres résultats que nous ne montrons pas ici, nous n'arrivons pas à résoudre des instances avec $p \geq 6$ si la procédure n'utilise pas le test de la borne $\tilde{B}$. Ceci montre l'apport de la borne $\tilde{B}$ pour la résolution d'instances possédant un nombre important d'objectifs.

Il est aussi remarquable que notre procédure s'adapte mal à l'augmentation du nombre des objectifs p. C'est là l'une des principales faiblesses de l'énumération ordonnée avec minoration unique, à laquelle la version avec minoration variable apporte une solution. Nous discutons de ce cas dans la section 3.4.2 qui est dévolue aux résultats expérimentaux de la procédure avec minoration variable.

3.4.2 Résultats expérimentaux pour la procédure d'énumération ordonnée avec minoration variable

Nous présentons dans la table 3.6 une comparaison des deux versions de l'algorithme par énumération ordonnée avec minoration variable et unique. La version unique correspond à $\hat{B}\tilde{B}R$ et variable est une version qui utilise l'algorithme 3.3 pour la recherche de la $k^{\text{ème}}$ meilleure solution et qui intègre tous les tests de bornes et le changement de l'ordre de la séparation. Dans la version variable, les conditions d'arrêt et de mise à jour de la meilleure solution courante de l'algorithme 3.1 doivent être modifiées comme nous le montrons dans les commentaires de la proposition 4 dans la section 3.3.

Les deux premières colonnes de la table 3.6 donnent le nombre de solutions énumérées et la proportion des solutions énumérées après l'obtention de l'optimum pour les deux variantes (voir la table 3.2 pour des résultats similaires). La troisième et la quatrième colonnes montrent la taille des arborescences obtenues et les deux dernières colonnes les temps CPU en secondes. Nous avons recueilli les résultats de 20 instances aléatoires dont les moyennes sont présentées dans la table 3.6.

Mesures		Solutions énumérées		Arborescence		Temps CPU (s.)	
n	p	unique	variable	unique	variable	unique	variable
	2	39 (72 %)	15 (49%)	45	57	0.04	0.25
50	3	570 (62%)	66 (25%)	618	289	0.06	1.20
	4	3 505 (89%)	189 (47%)	3 753	938	0.13	4.63
	2	56 (91%)	23 (65%)	67	100	0.23	1.75
100	3	1 756 (75%)	196 (48%)	1 956	921	0.37	20.37
	4	21 833 (90%)	910 (69%)	24 231	4 845	1.33	141.34
	2	110 (94%)	65 (89%)	137	270	1.02	36.55
200	3	3 205 (85%)	405 (63%)	3 604	2 164	1.45	428.91
	4	122 827 (89%)	_	137 378	_	13.24	_

TABLE 3.6 – Comparaison des variantes avec minoration unique et variable

Bien que la version variable énumère moins de solutions et engendre des arborescences plus petites que la version unique, elle est moins intéressante en termes de temps de calcul. On en déduit immédiatement que pour améliorer le temps CPU de la version variable, les efforts doivent être concentrés sur la réduction du temps de la recherche de la $k^{\text{ème}}$ meilleure solution, et non sur la diminution du nombre de solutions énumérées. Par conséquent, les étapes qu'il faut améliorer sont celles qui consomment le plus d'efforts de calcul lors de la recherche d'une $k^{\text{ème}}$ meilleure solution, à savoir, la recherche de la meilleure minoration et de la solution optimale d'un nœud (i.e. étapes 8 et 10 de l'algorithme 3.3). Ceci peut être confirmé aussi grâce au rapport entre le temps CPU et le nombre de solutions énumérées et le rapport entre le temps CPU et la taille de l'arborescence. Ces rapports donnent le temps moyen de l'énumération d'une solution ou

le temps moyen de traitement d'un nœud. Ces deux rapports sont en faveur de la version unique.

Le rapport de la taille de l'arborescence sur le nombre de solutions énumérées donne une information sur la forme des arborescences développées par chacune des deux variantes. Avec une moyenne proche de 5 nœuds traités par solution énumérée (pour tous n et p de la table 3.6), les arborescences engendrées par la version variable se développent en largeur, alors que les arborescences de la version unique sont profondes avec une moyenne très proche d'un nœud résolu pour toute solution énumérée. Un compromis est donc à trouver entre le nombre de solutions énumérées et l'effort nécessaire pour la recherche de chacune de ces solutions.

Nous avons procédé à diverses expérimentations afin d'évaluer certains choix algorithmiques de la version variable. Nous commentons ici brièvement quelques-uns des résultats remarquables que nous obtenons.

Nous avons comparé trois manières d'introduire les nouvelles contraintes des arêtes imposées et interdites $I(v)$ et $F(v)$ dans le programme linéaire (3.18), à savoir :

- forcer les variables concernées à valoir 0 ou 1 avec de nouvelles contraintes ;
- procéder en pénalisant ou en favorisant des variables selon qu'elles correspondent à des arêtes interdites ou imposées ;
- supprimer les variables contraintes du problème (3.18).

Nous avons retenu la première manière, ce qui permet de résoudre le nouveau problème avec la méthode duale du simplexe en partant de la base duale optimale du nœud père.

Le choix de la position de l'instruction qui calcule la solution optimale $x(v)$ d'un nœud v (étape 11 de l'algorithme 3.3.) a fait aussi l'objet de tests expérimentaux. Comme nous l'expliquons dans la section 3.3.2, la solution $x(v)$ peut être calculée au besoin, c'est-à-dire au moment où cette solution serait énumérée. La solution $x(v)$ peut aussi être déterminée de manière systématique, comme c'est le cas dans l'algorithme 3.3, i.e. après que les divers tests de bornes aient échoué à couper le nœud v et avant l'insertion du nœud v dans l'ensemble $\mathcal{L}$. La comparaison des résultats expérimentaux relatifs à ces deux alternatives nous a conduit à retenir la seconde version et donc à positionner la recherche de la solution $x(v)$ à l'étape 11 de l'algorithme 3.3.

Les instances avec un nombre important d'objectifs p, même si elles sont de petite taille (n petit), ralentissent la version unique, comme on peut l'observer dans les résultats de la table 3.5. Nous présentons dans la table 3.7 les temps de calcul moyens pour 20 instances aléatoires résolues avec les mêmes versions de l'énumération ordonnée unique et variable utilisées dans les tests de la table 3.6. Nous avons choisi des instances de petite taille ($n = 20$) par rapport à ce que nous pouvons résoudre avec nos procédures, et celà afin de pouvoir comparer les deux versions en augmentant le nombre des objectifs.

La version variable est meilleure que la version unique pour ce type d'instances.

n	p	unique	variable
20	7	2.57	2.34
	8	0.97	2.87
	9	119.54	8.09
	10	110.88	11.44
	11	572.02	10.28
	12	33.58	10.98
	13	–	43.36
	14	–	80.93

TABLE 3.7 – Temps CPU des versions unique et variable pour $7 \leq\ p \leq\ 14$ (secondes)

Les résultats de la table 3.7 montrent en effet que la version variable supporte mieux l'augmentation du nombre d'objectifs p par rapport à la version unique.

Cela s'explique par le fait que pour la version unique, les contraintes qui s'ajoutent au problème original, afin de créer les nouveaux nœuds, détériorent la qualité de l'approximation de la fonction f par $\hat{f}$. La raison est que le jeu de poids optimal ω^*, qui définit $\hat{f}$, est calculé une fois au début de l'énumération pour la version unique. La version avec minoration variable ne souffre pas de ce problème, puisque la fonction minorante est redéfinie pour chaque nouveau nœud de l'arborescence, ce qui corrige l'erreur d'approximation de f par $\hat{f}_v$ causée par les contraintes du nœud v.

Ce défaut de la version unique est plus prononcé pour les instances avec beaucoup d'objectifs, puisque pour ce cas particulier l'approximation de f par $\hat{f}$ est de mauvaise qualité. En effet, la valeur d'une solution selon la pseudo-distance f est donnée par une seule composante du vecteur critère qui lui est associé. La valeur de la même solution x selon la fonction $\hat{f}$ est une combinaison des p composantes de son image $z(x)$, donc les $p-1$ composantes qui ne réalisent pas le maximum faussent l'approximation de la fonction f par $\hat{f}$.

La comparaison des résultats des tables 3.6 et 3.7 prouve que l'impact négatif de l'augmentation de la taille n est beaucoup plus important que celui causé par l'augmentation du nombre des objectifs p sur les performances de la version variable. Ceci peut être expliqué par le fait qu'une instance de grande taille engendre une séparation large avec le schéma de Murty-Lawler, ce qui ralentit la recherche de la $k^{\text{ème}}$ meilleure solution. Or comme nous l'avons évoqué dans les commentaires de la table 3.6, c'est la principale faiblesse de la version variable. D'un autre côté, l'augmentation du nombre des objectifs impacte uniquement la recherche du meilleur jeu de poids, où une contrainte est ajoutée au programme linéaire (3.18) pour tout objectif supplémentaire. Ces contraintes ont peu d'effet sur le temps de calcul du programme linéaire continu (3.18).

3.4.3 Instances difficiles

Au vu des résultats des algorithmes par énumération ordonnée présentés dans les tables 3.5 et 3.7 pour les instances aléatoires, nous testons de nouveaux types d'instances. Knowles et Corne (2001) font une comparaison de plusieurs familles d'instances pour le problème de l'*arbre couvrant multiobjectif avec des contraintes sur les degrés des sommets*. Le principe général de ces instances est d'introduire du conflit entre les composantes des points réalisables dans l'espace des objectifs. Les instances obtenues par les auteurs sont difficiles, au sens où lorsqu'ils résolvent ce type d'instances, les temps de calculs de leurs procédures pour la génération de l'ensemble des points non-dominés sont plus élevés que les temps qu'ils obtiennent pour des instances aléatoires.

Les solutions réalisables du problème de l'arbre couvrant sont constituées d'un nombre fixe d'arêtes ($(n-1)$ arêtes, où n est le nombre de sommets du graphe). Le problème d'affectation partage cette cette propriété avec le problème de l'arbre couvrant, puisque ses solutions réalisables sont données par n arêtes. Par conséquent, nous nous attendons à des conclusions similaires à celles de Knowles et Corne sur la difficulté de ces nouvelles instances. Cependant, les auteurs cherchent tous les points non-dominés pour leurs problèmes, ce qui n'est pas notre cas. De plus, nos instances ne sont pas composées uniquement des coûts associés aux arêtes, mais elles nécessitent aussi de définir un point de référence et une direction de recherche, et la définition de ces paramètres a un impact crucial sur les performances des algorithmes. Nous nous inspirons donc des instances de Knowles et Corne pour générer deux nouvelles familles d'instances pour le problème d'affectation multiobjectif, à savoir, les instances avec des coûts conflictuels et les instances concaves.

3.4.3.a Instances avec des coûts conflictuels

Nous décrivons dans cette section une famille d'instances dont les p coûts $(c_{ij}^1, \ldots, c_{ij}^p)$ associés à une arête (i,j) pour $i,j = 1,\ldots,n$ sont *conflictuels*.

Soit $I = \{I_{\min}, \ldots, I_{\max}\}$ un ensemble d'entiers naturels. Les coûts c_{ij}^k, $i,j = 1,\ldots,n$ et $k = 1,\ldots,p$, sont tirés aléatoirement dans l'ensemble I. Afin que les ensembles I_k, $k \in \{1,\ldots,p\}$, ne soient pas vides nous supposons que $I_{\max} - I_{\min} \geq p$. Soit $\{I_1,\ldots,I_p\}$ une partition de l'ensemble I telle que $I_k = \{I_{\min} + \lfloor \frac{k-1}{p}(I_{\max} - I_{\min})\rfloor, \ldots, I_{\min} + \lceil \frac{k}{p}(I_{\max} - I_{\min})\rceil\}$ pour $k = 1,\ldots,p$. Chaque vecteur de coûts $(c_{ij}^1,\ldots,c_{ij}^p)$ pour cette famille d'instances est tiré dans $I_1 \times \ldots \times I_p$. Les composantes de ce vecteur, initialement trié, sont ensuite mélangées aléatoirement.

Pour chacune de ces instances avec $I = \{0,\ldots,100\}$, nous choisissons aléatoirement différents points de référence dans la zone délimitée par le point idéal et la même approximation du point nadir que celle que nous utilisons pour les instances aléatoires. Ensuite, pour chaque instance avec un point de référence fixé, nous générons plusieurs directions de recherche λ. Enfin, nous recueillons les moyennes du temps de calcul pour 20 de ces

instances pour les variantes unique et variable que nous comparons aux résultats des instances aléatoires présentés dans la table 3.6.

L'objectif ici est d'interpréter le comportement global de nos procédures avec cette nouvelle famille d'instances. En moyenne les temps CPU de ces nouvelles instances sont plus élevés que ceux des instances aléatoires. Cependant, le fait de faire varier le point de référence et le jeu de poids nous permet d'obtenir quelques instances très difficiles dont le temps CPU est à l'origine de l'augmentation du temps CPU moyen. Les temps moyens n'apportent donc pas de nouvelles conclusions par rapport aux temps présentés dans la table 3.6. Afin de montrer la différence entre les instances aléatoires et les instances conflictuelles il faudrait présenter tous les résultats obtenus, ce qui ne nous semble pas raisonnable.

3.4.3.b Instances concaves

Dans cette section nous présentons une famille d'instances qui pénalisent fortement nos approches par l'introduction d'une région concave dans l'espace des objectifs. Pour ce faire nous utilisons un mécanisme que les chercheurs qui ont participé à l'ANR GUEPARD [3] s'accordent à appeler un *"gadget"*.

Dans le cas biobjectif, une instance concave de taille $n+2$ est obtenue en assemblant deux instances, l'une quelconque (par exemple aléatoire) de taille n et l'autre de taille 2 spécialement construite pour introduire un cône vide dans l'espace des objectifs. Le graphe biparti associé à cette instance est donc un graphe contenant $2n+4$ sommets, les $2n$ sommets de l'instance aléatoire et les sommets $1', 2', 1''$ et $2''$ de la figure 3.4. Les coûts des arêtes adjacentes à ces nouveaux sommets sont donnés par : $c_{1'1''} = c_{2'2''} = (q_1, 0)$, $c_{1'2''} = c_{2'1''} = (0, q_2)$ et $c_{1'j} = c_{2'j} = c_{i1''} = c_{j2''} = (\infty, \infty)$ pour $i, j = 1, \ldots, n$, où q_1 et q_2 dans $\mathbb{N}$. Afin d'alléger la figure 3.4 nous ne représentons pas les arêtes avec coût infini.

La figure 3.5a représente l'image des solutions réalisables dans l'espace des objectifs de l'instance aléatoire et la figure 3.5b montre l'instance concave obtenue par l'introduction du sous-graphe engendré par les sommets $1', 2', 1''$ et $2''$.

Les points gris et noirs de la figure 3.5b sont les images des points de la figure 3.5a par la translation de vecteur $\binom{2q_1}{0}$ et $\binom{0}{2q_2}$ respectivement. Cela crée un cône vide dont le sommet est donné par $(2q_1 + z_1^*, 2q_2 + z_2^*)$. Dans la figure 3.5b nous avons choisi $q_k = (\lceil z_k^N - z_k^* \rceil / 2\rceil$ pour $k = 1, 2$.

Pour obtenir des instances difficiles, nous proposons le point idéal comme point de référence et la direction de recherche allant du point idéal z^* vers le sommet du cône vide. Les instances de cette famille sont extrêmement difficiles même lorsqu'elles sont de

3. "GUaranteed Efficiency for PAReto optimal solutions Determination in multiobjective combinatorial optimization problems", est un projet scientifique français qui a bénéficié du support de l'agence nationale de la recherche (ANR) entre 2010 et 2013. Ce projet fut un partenariat entre trois laboratoires : LAMSADE, LINA et LIP6.

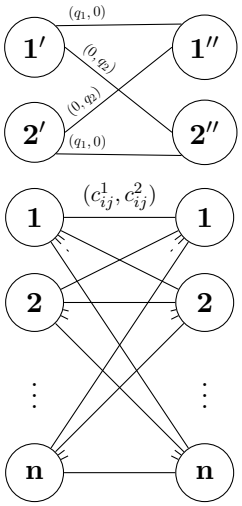
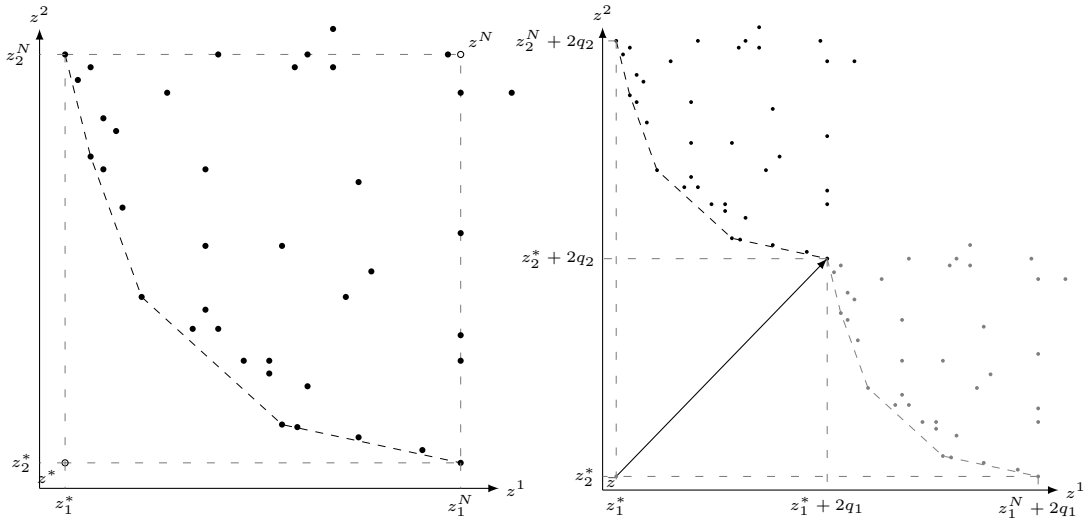


FIGURE 3.4 – Principe des instances concaves



(a) (b)

FIGURE 3.5 – Instance concaves dans l'espace des objectifs

petite taille. Nous parvenons à résoudre des instances avec 2 objectifs et de taille $n = 11$ pour la version unique et $n = 18$ pour la version variable, et cela après des temps de calcul très élevés (plusieurs heures).

Il est intéressant de remarquer que la taille maximale des instances de cette famille que nous pouvons résoudre est bien précise, contrairement aux instances aléatoires où nos procédures atteignent leurs limites progressivement. Par exemple, nous pouvons ré-

soudre avec la version unique toutes les instances concaves de taille $n = 11$, mais on ne peut résoudre aucune instance de taille $n = 12$, alors que de manière générale pour les instances aléatoires, en augmentant la taille nous remarquons d'abord que certaines instances conduisent à la saturation de la RAM, puis le nombre de ces instances problématiques augmente progressivement jusqu'à ce que les temps deviennent trop élevés pour être relevés. Cela peut s'interpréter par le fait que notre procédure subit complètement les méfaits de la combinatoire, et ce parce qu'elle énumère la quasi-totalité des solutions réalisables. Par conséquent, le passage d'une taille d'instance n à $n + 1$ implique une explosion exponentielle du temps CPU.

3.5 Conclusion

Nous proposons dans ce chapitre des algorithmes d'énumération ordonnée pour la recherche d'une solution de bon compromis pour le problème d'affectation multiobjectif. Nous introduisons des améliorations à différents niveaux de la méthode générique présentée dans le chapitre précédent, ce qui engendre plusieurs variantes.

Nous avons procédé à l'analyse des résultats expérimentaux observés lors de la résolution de différentes instances. Aussi nous nous sommes basés sur des observations empiriques, telles que le temps CPU ou le nombre de solutions énumérées, pour apprécier l'efficacité des variantes de notre algorithme et les comparer aux performances du solver CPLEX lorsque la formulation en programme linéaire mixte en nombres entiers de notre problème est résolue. Les résultats obtenus ont prouvé l'utilité de nos améliorations et l'efficacité globale de notre méthode. En plus de leurs bonnes performances calculatoires, les différentes variantes de notre algorithme permettent de prendre en considération la nature multiobjectif du problème original en garantissant l'efficacité des solutions optimales qu'elles retournent.

Chapitre 4
Problème du voyageur de commerce asymétrique

Résumé Nous proposons dans ce chapitre une méthode par énumération ordonnée de solutions réalisables du problème d'affectation afin de résoudre le problème du voyageur de commerce asymétrique. Dans la section 4.1 nous rappelons la définition du problème du voyageur de commerce asymétrique et nous mettons l'accent sur son lien avec le problème d'affectation. Nous discutons dans la section 4.2 dans un premier temps des principales méthodes présentes dans la littérature qui se basent sur la relaxation en problème d'affectation (section 4.2.1), puis nous décrivons dans ses grandes lignes l'algorithme que nous proposons pour résoudre ce problème (section 4.2.2). Nous apportons dans la section 4.3 des précisions sur les améliorations que nous proposons d'introduire par rapport aux algorithmes de la littérature. Nous procédons dans la section 4.4 à l'analyse des résultats expérimentaux que nous obtenons. Cette analyse nous conduit à conclure sur l'efficacité globale de notre algorithme et à établir que le choix d'une version efficace de l'algorithme dépend des caractéristiques de l'instance résolue.

4.1 Définitions préliminaires

Nous nous intéressons dans ce chapitre au problème du voyageur de commerce asymétrique, et plus précisément à une méthode par énumération ordonnée pour la résolution de ce problème. De manière formelle, le problème du voyageur de commerce asymétrique, noté aussi **ATSP**, est défini par la donnée d'un graphe simple orienté complet $G = (W, A)$, avec $W = \{1, \ldots, n\}$ l'ensemble des sommets et $A = \{(i, j) : i, j \in W\}$ l'ensemble des arcs. À chaque arc (i, j) est associé un coût $c_{ij} \in \mathbb{R}_{\geq}$. Le problème du voyageur

de commerce asymétrique consiste à trouver un circuit $\mathcal{C}$ passant une et une seule fois par chaque sommet du graphe G (*circuit hamiltonien*) dont le coût total $\sum_{(i,j)\in\mathcal{C}} c_{ij}$ est minimal.

Notons que le problème du voyageur de commerce symétrique qui est souvent formalisé comme la recherche d'un cycle de coût minimum dans un graphe non-orienté, est un cas particulier de **ATSP**, où les coûts sont symétriques $c_{ij} = c_{ji}$ pour i, j dans W.

Les problèmes du voyageur de commerce, symétrique et asymétrique, sont des problèmes NP-difficiles (voir Garey et Johnson (1979)).

Problème du voyageur de commerce asymétrique et problème d'affectation

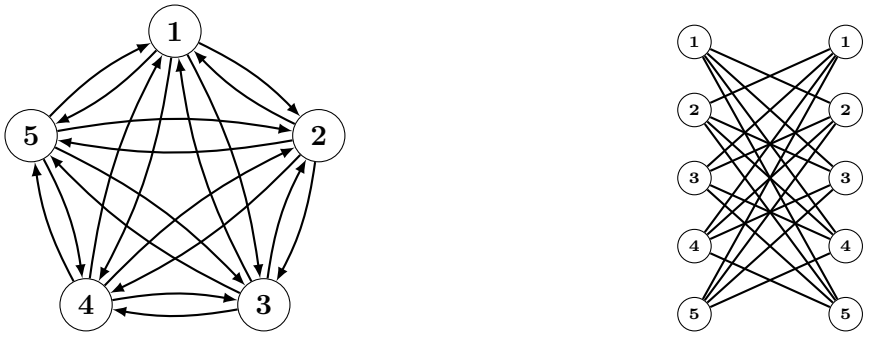
Le programme linéaire en nombres entiers (4.1) est une des formulations du problème du voyageur de commerce asymétrique.

$$\begin{aligned}
\min \quad & \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} & & \\
\text{s.c.} \quad & \sum_{j=1}^{n} x_{ij} = 1 & i = 1,\dots,n & \text{(4.1.a)}\\
& \sum_{i=1}^{n} x_{ij} = 1 & j = 1,\dots,n & \text{(4.1.b)}\\
& \sum_{i\in S, j\notin S} x_{ij} \geq 1 & S \subset W,\ S \neq \emptyset & \text{(4.1.c)}\\
& x_{ij} \in \{0,1\} & i, j = 1,\dots,n & \text{(4.1.d)}
\end{aligned}$$

Les contraintes (4.1.a),(4.1.b) et (4.1.d) avec la fonction objectif définissent le problème d'affectation (voir le problème (1.7)). Le problème d'affectation est par conséquent une relaxation du problème du voyageur de commerce asymétrique.

Il existe diverses formulations de **ATSP** qui diffèrent par les contraintes (4.1.c). Une *sous-tournée* est un circuit élémentaire qui, contrairement à une *tournée*, ne parcourt pas tous les sommets du graphe G. Les contraintes (4.1.c) éliminent les solutions du problème d'affectation qui contiennent des sous-tournées et elles sont à l'origine de la difficulté de **ATSP**. La figure 4.1 montre la correspondance entre **ATSP** et le problème d'affectation, où l'on passe du premier au second en dédoublant les sommets du graphe $G = (W, A)$ et en associant une arête dans le graphe biparti $G' = (U, V; A')$ pour chaque arc (i, j) dans A. Une arête (i, j) d'une solution du problème d'affectation correspond à l'arc (i, j) pour **ATSP**, aussi il peut arriver dans ce chapitre que nous fassions référence indifféremment aux arcs ou aux arêtes.

Le graphe $G = (W, A)$ du problème du voyageur de commerce asymétrique étant sans

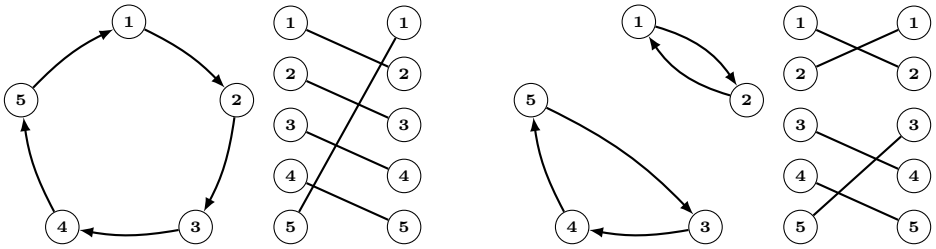


(a) (b)

FIGURE 4.1 – Relaxation de **ATSP** en problème d'affectation

boucles, les coûts c_{ii} sont fixés à $+\infty$. C'est pour cette raison que nous ne représentons pas les arêtes qui correspondent à des boucles dans la figure 4.1b.

Les correspondances entre les solutions réalisables du problème d'affectation et problème du voyageur de commerce asymétrique sont illustrées par les deux exemples de la figure 4.2.



(a) (b)

FIGURE 4.2 – Correspondances entre les solutions de **ATSP** et les solutions du problème d'affectation

La figure 4.2a montre une solution réalisable pour les deux problèmes alors que la figure 4.2b donne une affectation réalisable qui comporte deux sous-tournées, et qui n'est donc pas une solution réalisable pour le problème du voyageur de commerce asymétrique. Une solution réalisable du problème d'affectation correspond soit à une tournée, soit à un ensemble de sous-tournées sommets-disjointes, dont l'union visite tous les sommets du graphe G.

4.2 Énumération ordonnée pour ATSP

Nous apportons dans cette section des précisions sur l'application du schéma général de l'énumération ordonnée présenté dans les algorithmes 4.1 et 4.3 du chapitre 2 au

problème du voyageur de commerce asymétrique.

Nous présentons dans la section 4.2.1 quelques algorithmes de la littérature qui procèdent par l'énumération de solutions réalisables pour le problème d'affectation. Nous détaillons ensuite dans la section 4.2.2, notamment par les algorithmes 4.2 et 4.3, les particularités de l'énumération ordonnée lorsqu'elle est appliquée à la résolution de **ATSP**.

4.2.1 Algorithmes fondés sur la relaxation en problème d'affectation dans la littérature

Nous décrivons dans cette section les principales méthodes présentes dans la littérature qui se fondent sur la relaxation du problème du voyageur de commerce asymétrique en problème d'affectation.

À notre connaissance, Murty et al. (1962) sont les premiers à avoir proposer d'énumérer des solutions réalisables du problème d'affectation dans l'ordre croissant de leur coût pour résoudre le problème du voyageur de commerce asymétrique. L'énumération est stoppée lorsque la $k^{\text{ème}}$ meilleure solution du problème d'affectation constitue une tournée. Les étapes principales de cette procédure sont résumées dans l'algorithme 4.1. Le problème principal est noté $\mathcal{P}(X, f)$ et la relaxation principale en problème d'affectation est notée $\mathcal{P}(\hat{X}, f)$. Notons que le problème principal et sa relaxation en problème d'affectation ont la même fonction objectif f. L'ensemble des k meilleures solutions pour le problème d'affectation $\mathcal{P}(\hat{X}, f)$ est noté $\hat{Y}$.

Algorithme 4.1 : Énumération ordonnée par Murty et al. (1962) pour le problème du voyageur de commerce asymétrique

Données : $\mathcal{P}(X, f), \mathcal{P}(\hat{X}, f)$
$k \leftarrow 0$, $\hat{Y} \leftarrow \emptyset$
répéter
  $k \leftarrow k + 1$
  Déterminer $x^k \in \arg\min_{x \in \hat{X} \setminus \hat{Y}} f(x)$ `// Résoudre` $\mathcal{P}(\hat{X} \setminus \hat{Y}, f)$
  $\hat{Y} \leftarrow \hat{Y} \cup \{x^k\}$
jusqu'à $(x^k \in X)$
retourner x^k

Le papier original de Murty et al. (1962) a été révisé pour donner lieu à un nouvel article (Little et al., 1963) où les auteurs proposent ce qui est connu comme *la méthode de Little* pour la version symétrique du problème du voyageur de commerce. La nouvelle version de leur algorithme, où la tournée optimale se construit progressivement, s'est donc éloignée de l'énumération ordonnée.

L'idée de recourir à une énumération ordonnée des affectations a ensuite été reprise par Carpaneto et Toth (1980) qui modifièrent légèrement le schéma de séparation de Murty. Les arêtes interdites ou imposées pour créer un nouveau nœud sont toutes choi-

sies dans une même sous-tournée. Ceci permet de casser une sous-tournée à chaque séparation. Considérons par exemple lors de la recherche de la $k^{\text{ème}}$ meilleure solution, que la solution de la figure 4.2b représente la $k-1^{\text{ème}}$ meilleure solution pour le problème d'affectation. Notons cette solution $x(v')$ et v' le nœud auquel elle appartient. Supposons, pour faciliter les notations et sans perte de généralité, que les ensembles des arêtes imposées et interdites sont vides, i.e. $I(v') = F(v') = \emptyset$. L'application directe du schéma de séparation de Murty du nœud v' est donnée dans la figure 4.3.

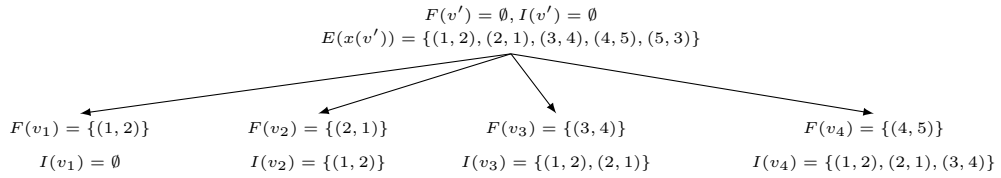


FIGURE 4.3 – Séparation de Murty naïve pour **ATSP**

Bellmore et Malone (1971) font remarquer que dans une telle situation les nœuds v_3 et v_4 ne contiennent aucune tournée puisque les arêtes $(1,2)$ et $(2,1)$ imposées pour ces nœuds forment une sous-tournée. C'est pour cela que Carpaneto et Toth (1980) proposent de séparer les nœuds sur la base des arcs qui correspondent à une sous-tournée. Ceci permet de ne pas créer certains nœuds parmi ceux qui auraient pu être créés par une séparation naïve comme celle de la figure 4.3. Ils précisent aussi qu'il est préférable de choisir la tournée qui a le moins d'arcs non imposés, ou *arcs libres*, ce qui conduit à créer à chaque séparation un nombre minimal de nœuds. Afin de simplifier la gestion des ensembles des arêtes imposées ils proposent de sélectionner les arêtes à interdire dans l'ordre dans lequel les arcs qui leur correspondent apparaissent dans la sous-tournée.

Pekny et Miller (1992) proposent une parallélisation de l'algorithme de Carpaneto et Toth (1980). À la même période Fischetti et Toth (1992) proposent d'introduire de nouvelles relaxations pour le problème du voyageur de commerce asymétrique (fondées sur les problèmes d'*arborescence* et d'*anti-arborescence couvrantes de poids minimum*) en plus du problème d'affectation qui reste la relaxation principale.

L'algorithme de Carpaneto et Toth (1980) a été amélioré par les mêmes auteurs et Dell'Amico dans Carpaneto et al. (1995).

1. La première amélioration est un prétraitement de l'instance du problème du voyageur de commerce asymétrique. Les coûts réduits du problème d'affectation résolu à la racine v^r de l'arborescence sont utilisés afin de supprimer quelques arcs du problème du voyageur de commerce asymétrique suivant la règle suivante : tout arc (i,j) avec un coût réduit $\bar{c}_{ij}$ tel que $f(x(v^r)) + \bar{c}_{ij} \geq f(x^0)$ ne peut appartenir à la solution optimale, où x^0 est une tournée initiale quelconque, de préférence obtenue par une méthode approchée. Dans leur méthode, Carpaneto et al. calculent la tournée initiale x^0 avec l'heuristique du *Patching Algorithm* de Karp (1979).

2. Une seconde amélioration consiste en la résolution des sous-problèmes d'affectation

dans les nouveaux nœuds à l'aide d'un plus court chemin tel que nous l'expliquons dans les commentaires de la figure 1.6.

3. Les auteurs proposent aussi de *fusionner* les sous-tournées de la solution optimale d'un nœud. Les auteurs rappellent que la solution optimale du problème d'affectation associé à un nœud de l'arborescence n'est pas forcément unique. Aussi, ils proposent de choisir parmi les solutions optimales équivalentes d'un nœud celle qui contient le moins de sous-tournées, et ce en avançant des raisons pratiques. Pour ce faire ils proposent une heuristique qui est basée sur des échanges d'arêtes de coût réduit nul.

Nous avons cité dans cette section des références utilisant uniquement des méthodes de Branch and Bound basées sur la relaxation du problème du voyageur de commerce asymétrique en problème d'affectation, sans pour autant être exhaustifs. Néanmoins, nous avons cité dans cette section les méthodes qui ont introduit de nouvelles idées et qui sont par ailleurs les plus performantes.

4.2.2 Algorithmes par énumération ordonnée pour ATSP

Nous proposons une procédure par énumération ordonnée fondée sur l'algorithme de Carpaneto et Toth (1980), enrichie des idées des améliorations introduites par Carpaneto et al. (1995) ainsi que de nouvelles améliorations que nous proposons.

L'énumération dans nos procédures s'effectue suivant l'algorithme 4.2 qui est essentiellement basé sur l'algorithme 4.1. Quelques différences entre les deux algorithmes sont néanmoins à noter. En effet, la meilleure tournée courante, que nous notons x^*, est spécifique à l'algorithme 4.2. Cette tournée est initialisée à une tournée quelconque x^0 et elle est mise à jour lors de la recherche de la $k^{\text{ème}}$ meilleure solution à l'étape 4. Nous discuterons ce point dans les commentaires de l'algorithme 4.3. Précisons qu'il est préférable d'initialiser la tournée x^* avec une tournée de bonne qualité obtenue par exemple avec une heuristique. Le second changement remarquable se situe à l'étape 6 qui représente la condition d'arrêt. Cette nouvelle condition signifie que dans l'algorithme 4.2, l'énumération des affectations et la mise à jour de la meilleure tournée courante x^* s'arrêtent lorsque la valeur de la tournée x^* est meilleure que toutes les affectations qui n'ont pas encore été énumérées.

Algorithme 4.2 : Résolution de **ATSP** par énumération ordonnée des affectations

Données : $\mathcal{P}(X, f), \mathcal{P}(\hat{X}, f), x^0$
1 $x^* \leftarrow x^0$, $k \leftarrow 0$, $\hat{Y} \leftarrow \emptyset$
2 **répéter**
3 $\quad k \leftarrow k + 1$
4 $\quad$ Déterminer $x^k \in \arg\min_{x \in \hat{X} \setminus \hat{Y}} f(x)$ et mettre à jour x^* `// Algorithme 4.3`
5 $\quad \hat{Y} \leftarrow \hat{Y} \cup \{x^k\}$
6 **jusqu'à** $(f(x^k) \geq f(x^*))$
7 **retourner** x^*

Le calcul de la $k^{\text{ème}}$ meilleure affectation (étape 4) est détaillé dans l'algorithme 4.3. Notons que la mise à jour de la meilleure tournée courante est accomplie par l'algorithme 4.3. La fonction objectif du problème principal $\mathcal{P}(X, f)$ est linéaire. C'est pour cela que nous ne proposons pas ici une version de notre procédure où la fonction objectif de la relaxation principale est adaptée pour chaque nouveau nœud (minoration variable).

Dans l'algorithme 4.3, $\mathcal{L}$ est l'ensemble des feuilles de l'arborescence de recherche, x^{k-1} la $k-1^{\text{ème}}$ solution réalisable pour le problème d'affectation $\mathcal{P}(\hat{X}(v'), f)$ énumérée dans l'algorithme 4.2 et v' le nœud contenant la solution x^{k-1}. Les valeurs $\hat{B}(v)$,$f(v)$ et $\tilde{B}(v)$ sont des bornes inférieures associées au nouveau nœud v qui est créé à l'étape 7 de l'algorithme 4.3.

Algorithme 4.3 : $k^{\text{ème}}$ meilleure affectation pour résoudre **ATSP**

1 $\mathcal{L} \leftarrow \mathcal{L} \setminus \{v'\}$; $I \leftarrow I(v')$
2 Chercher une bonne tournée pour le nœud v' et mettre à jour x^*
3 Chercher une solution optimale équivalente à x^{k-1} dans $\hat{X}(v')$
4 Choisir la sous-tournée $\mathcal{C}$ qui sera cassée lors de la séparation
5 **tant que** $|\mathcal{C} \setminus I| > 0$ **faire**
6 $\quad$ Choisir l'arête (i^*, j^*) dans $\mathcal{C} \setminus I$ à interdire
7 $\quad$ Définir le nœud v tel que : $I(v) \leftarrow I$ et $F(v) \leftarrow F(v') \cup \{(i^*, j^*)\}$
8 $\quad$ **si** $\hat{B}(v) < f(x^*)$ **alors**
9 $\quad\quad$ Résoudre le problème $\mathcal{P}(\hat{X}(v), f)$
10 $\quad\quad$ **si** $f(v) < f(x^*)$ **alors**
11 $\quad\quad\quad$ **si** $x(v) \in X$ **alors**
12 $\quad\quad\quad\quad$ $x^* \leftarrow x(v)$
13 $\quad\quad\quad$ **sinon**
14 $\quad\quad\quad\quad$ Calculer la borne $\tilde{B}(v)$ `// Algorithme 4.4`
15 $\quad\quad\quad\quad$ **si** $\tilde{B}(v) < f(x^*)$ **alors** Insérer v dans $\mathcal{L}$
16 $\quad I \leftarrow I \cup \{(i^*, j^*)\}$
17 $x^k \in \arg\min_{v \in \mathcal{L}} f(x(v))$

Avant de procéder à la séparation du nœud v', nous proposons à l'étape 2 de calculer une bonne solution pour **ATSP** $P(X(v'), f)$ associé au nœud v'. Nous utilisons une version modifiée de l'heuristique de Karp (1979). Nous présentons brièvement dans la section 4.3.1 l'idée générale de cet algorithme et les légères modifications que nous y avons introduites.

Dans certaines des variantes de l'algorithme 4.3, nous cherchons à l'étape 3 une solution équivalente à x^{k-1} dans l'ensemble $\hat{X}(v')$. Comme nous l'avions évoqué précédemment, Carpaneto et al. (1995) proposent une heuristique par fusion de sous-tournées qui tend à privilégier les solutions équivalentes qui comptent le moins de sous-tournées. Nous présentons dans la section 4.3.3 l'heuristique que nous proposons pour la recherche d'une solution équivalente. L'alternative que nous proposons est fondée sur l'énumération de couplages parfaits et présente l'avantage d'être compatible avec divers critères de choix de solution équivalente.

L'étape 4 consiste à choisir la sous-tournée $\mathcal{C}$ sur laquelle se basera la séparation. Nous détaillons dans la section 4.3.2 les critères que nous proposons pour la sélection de la sous-tournée $\mathcal{C}$.

À l'étape 6 un arc est choisi dans $\mathcal{C} \setminus I$ et interdit pour créer un nouveau fils v. Cet arc sera ensuite imposé à l'étape 16 pour tous les nœuds frères du nœud v. Ici aussi nous proposons une alternative à l'algorithme de Carpaneto et al. (1995) qui interdit les arcs dans l'ordre de leur apparition dans la sous-tournée $\mathcal{C}$, ce qui facilite la gestion de l'ensemble des arêtes imposées $I(v)$. Nous proposons de changer l'ordre de la séparation, en adoptant le même ordre que celui que nous présentons pour les procédures de recherche d'une solution de compromis pour le problème d'affectation multiobjectif (voir la section 3.2.4). Nous discutons dans la section 4.3.5 de l'ensemble des arêtes imposées $I(v)$ et de l'ensemble des arêtes interdites $F'(v)$ induit par l'ensemble $I(v)$.

Les étapes 8, 10 et 15 sont les tests de bornes auxquels sont soumis les nœuds.

étape 8 La borne $\hat{B}$ est la même que celle que nous proposons dans la section 3.2.3. Pour rappel, la valeur de la borne $\hat{B}(v)$ correspond à la somme des valuations minimales des arcs sortant du sommet i^* et entrant dans le sommet j^* dans le graphe d'écart associé au nœud v, telle que (i^*, j^*) est l'arête interdite lors de la définition du nœud v.

étape 10 La valeur $f(v)$ représente la valeur optimale du problème d'affectation $\mathcal{P}(\hat{X}(v), f)$ associé au nœud v.

étape 15 La borne $\tilde{B}$ provient d'une relaxation alternative à la relaxation principale en problème d'affectation. Plus de précisions sur la borne $\tilde{B}$ seront apportées dans la section 4.3.4, notamment l'algorithme 4.4 qui donne la valeur de $\tilde{B}(v)$.

Si le nouveau nœud v satisfait la condition du test de l'étape 8 alors le problème d'affectation $\mathcal{P}(\hat{X}(v), f)$ est résolu par la recherche d'un plus court chemin dans le graphe d'écart associé au nœud v (voir la section 1.3.2.a). La valeur de la solution optimale du

nœud est comparée à la valeur de la meilleure tournée courante x^*, et seuls les nœuds qui remplissent la condition de l'étape 10 sont candidats pour contenir la solution optimale.

Après l'étape 10, deux situations peuvent survenir :

étape 11 La solution optimale $x(v)$ du problème d'affectation $\mathcal{P}(\hat{X}(v), f)$ associé au nœud v est une tournée. Dans ce cas, nul besoin de développer le nœud v, puisque $x(v)$ est une tournée optimale pour le nœud v. La meilleure tournée courante x^* est donc mise à jour.
En effet, lors de la résolution du problème du voyageur de commerce asymétrique il n'y a aucune contrainte implicite sur la solution optimale, telle que l'efficacité dans le cas multiobjectif. Ici la meilleure tournée x^* peut être mise à jour dès que la solution optimale $x(v)$ du nœud v correspond à une tournée, et pas forcément au moment où cette solution est énumérée.

étape 13 La solution optimale $x(v)$ du nœud v n'est pas une tournée. Nous proposons de calculer dans ce cas une nouvelle borne inférieure $\tilde{B}(v)$, présentée dans la section 4.3.4, sur la base d'une nouvelle relaxation. Les nœuds qui remplissent le test de cette borne à l'étape 15 seront stockés dans l'ensemble $\mathcal{L}$.

L'algorithme 4.3 est une version qui contient toutes les améliorations que nous proposons. Lors des tests expérimentaux de la section 4.4, nous considérerons des variantes qui ne contiennent pas toutes ces améliorations, et ce afin de juger de l'intérêt de leur introduction dans l'algorithme 4.3.

La complexité de l'algorithme 4.3 avec les bornes que nous proposons est en $O(n^3)$ si la complexité de la recherche d'une solution équivalente (étape 3) ne dépasse pas $O(n^3)$. En effet, $\lfloor n/2 \rfloor$ fils sont créés au pire des cas lors de la séparation du nœud v'. Les opérations les plus coûteuses sont l'étape 9 qui équivaut à la recherche d'un plus court chemin en $O(n^2)$ et l'étape 14 qui s'exécute également en $O(n^2)$ comme nous le montrons dans la section 4.3.4. Nous discutons de la complexité de l'étape 3 dans la section 4.3.3.

4.3 Amélioration de l'énumération ordonnée pour ATSP

Nous apportons dans cette section des précisions sur les améliorations que nous proposons dans l'algorithme 4.3.

La section 4.3.1 est dévolue à la discussion de la recherche d'une bonne tournée pour les nœuds de l'arborescence. Nous développons ensuite dans la section 4.3.2 les critères de choix de la sous-tournée sur laquelle se base la séparation d'un nœud. Nous discutons dans la section 4.3.3 de la recherche d'une solution équivalente pour la solution optimale d'un nœud, puis nous présentons dans la section 4.3.4 le principe et le mode de calcul de la borne $\tilde{B}(v)$. Enfin nous consacrons la section 4.3.5 à la discussion de la gestion des ensembles des arêtes imposées et interdites.

4.3.1 Recherche d'une bonne tournée

Le principe du *patching algorithm*, proposé par Karp (1979), consiste à calculer la solution optimale d'un problème d'affectation, soit dans notre cas la solution optimale $x(v)$ d'un nœud v, et de fusionner de manière itérative ses sous-tournées deux à deux grâce à des échanges d'arcs jusqu'à l'obtention d'une tournée. À chaque itération la sous-tournée de coût minimum, notons-la $\mathcal{C}$, est choisie. Il convient ensuite de trouver la paire d'arcs de coût d'échange minimum, c'est-à-dire trouver deux arcs $((i,j),(i',j'))$ appartenant à la solution courante, où $(i,j) \in \mathcal{C}$ et $(i',j') \notin \mathcal{C}$ et tels que la quantité $(-c_{ij} - c_{i'j'} + c_{i'j} + c_{ij'})$ est minimale. La solution courante est donc modifiée en supprimant les arcs (i,j) et (i',j') et en y introduisant les arcs (i',j) et (i,j'). La complexité du *patching algorithm* est en $O(n^3)$.

Dans notre cas les problèmes **ATSP** associés aux nœuds de l'arborescence possèdent des contraintes supplémentaires. Aussi les arcs (i,j) et (i',j') ne doivent pas être imposés et il faut s'assurer que les arcs (i',j) et (i,j') ne soient pas interdits.

Nous utilisons cette heuristique pour tous les nœuds, ce qui nous fournit à la racine de l'arborescence de recherche une bonne solution initiale (x^0 dans l'algorithme 4.2).

4.3.2 Critères de choix de la sous-tournée de séparation

Nous développons ici l'étape 4 de l'algorithme 4.3. Nous avons abordé dans la section 4.2.1 l'intérêt de la séparation suivant les arcs appartenant à une même sous-tournée. Carpaneto et al. (1995) proposent de choisir la sous-tournée qui compte le moins d'arcs libres, ce qui favorise une séparation *étroite*. Nous proposons de considérer deux autres critères, à savoir, choisir la sous-tournée de taille minimum ou de coût minimum.

Ces deux nouveaux critères sont dans la continuité de l'intuition de Carpaneto et al. (1995) en engendrant les séparations les moins larges, car les sous-tournées les plus courtes et les moins coûteuses possèdent généralement peu d'arcs libres. De plus ces deux critères de choix possèdent d'autres avantages. En effet, la séparation selon les petites sous-tournées a tendance à ne pas modifier les plus grandes, ce qui permet de converger plus vite vers une tournée. Aussi, la séparation selon la sous-tournée de coût minimum favorise la division de ces sous-tournées fortement liées. Nous avons constaté que souvent les sous-tournées de faible coût, lorsqu'elles sont observées dans la solution optimale d'un nœud, persistent dans les solutions optimales des nœuds fils, et ce tant que des contraintes qui se rapportent directement à ce type de sous-tournées ne sont pas introduites.

Nous considérons dans la partie expérimentale, trois familles de variantes de l'algorithme 4.3 qui se différencient par le critère de choix de la sous-tournée de séparation, à savoir,

- la sous-tournée ayant le moins d'arcs libres ;
- la plus petite sous-tournée ;

- la sous-tournée de moindre coût.

4.3.3 Recherche de solutions équivalentes

Nous discutons dans cette section de la recherche d'une solution équivalente (étape 3 de l'algorithme 4.3). Carpaneto et al. (1995) proposent de choisir la solution qui compte le moins de sous-tournées. Ils procèdent à l'aide d'une heuristique qui rappelle le *patching algorithm*, tant les sous-tournées sont fusionnées deux à deux. La modification que les auteurs ont apporté au *patching algorithm* consiste à se limiter à des échanges d'arcs qui sont associés à des coûts réduits nuls. Nous proposons ici des alternatives pour le critère de choix et pour le mode de recherche d'une solution équivalente.

En effet, nous reprenons, pour la recherche d'une solution équivalente, les critères de choix de la sous-tournée de séparation que nous avons cités dans la section 4.3.2. Aussi nous proposons une méthode qui consiste à énumérer tout ou partie des affectations optimales pour un nœud, et de retenir celle qui convient le mieux au critère de choix considéré.

Six classes de versions de l'algorithme 4.3 sont envisageables, suivant le mode de recherche d'une solution équivalente :

- ne pas chercher de solution équivalente ;
- fusionner les sous-tournées avec la méthode de Carpaneto et al. (1995) ;
- énumération de solutions équivalentes et choix de la solution contenant :
  - le moins de sous-tournées ;
  - la sous-tournée ayant le moins d'arcs libres ;
  - la plus petite sous-tournée ;
  - la sous-tournée de coût min.

La recherche de toutes les solutions optimales pour le problème d'affectation $\mathcal{P}(\hat{X}(v), f)$ peut se faire par l'énumération de tous les couplages parfaits dans un graphe biparti qu'on notera $G = (U', V'; E_{\bar{c}})$. Ce graphe se construit sur la base de la solution optimale $x(v)$ du nœud v et des coûts réduits $\bar{c}_{ij}$ qui lui sont associés, avec $U' = \{i \in U : \forall j \in V, (i,j) \notin I(v)\}$, $V' = \{j \in V : \forall i \in U, (i,j) \notin I(v)\}$ et $E_{\bar{c}} = \{(i,j) \in U' \times V' : \bar{c}_{ij} = 0\}$ (voir Fukuda et Matsui (1992)).

Fukuda et Matsui (1994) proposent un algorithme arborescent pour l'énumération de tous les couplages parfaits dans un graphe biparti. Cet algorithme est basé sur la séparation de Chegireddy-Hamacher (voir la section 1.3.2.b). Un nœud w de l'arborescence de cet algorithme contient deux couplages parfaits notés $M(w)$ et $M'(w)$. Le premier couplage parfait $M(w^r)$ de la racine w^r de l'arborescence correspond à la solution optimale $x(v)$. Un second couplage parfait $M'(w^r)$ existe si le graphe d'écart qui est associé

au graphe $G = (U', V'; E_{\bar{c}})$ et au couplage parfait $M(w^r)$ contient un circuit C. Si un tel circuit n'existe pas alors le développement du nœud w^r est stoppé, sinon le circuit C correspond à un cycle alterné dans le graphe original $G = (U', V'; E_{\bar{c}})$. La différence symétrique entre $M(w^r)$ et les arêtes du cycle correspondant au circuit C donne le nouveau couplage parfait $M'(w^r)$. Soit w le nœud courant, et soit e une arête choisie dans $M(w) \setminus M'(w)$. L'arête e est imposée pour créer le nœud w_e et interdite pour créer le nœud $w_{\not e}$, par conséquent $M(w_e) = M(w)$ et $M(w_{\not e}) = M'(w)$ sont des couplages parfaits associés respectivement aux nœuds w_e et $w_{\not e}$. Les seconds couplages parfaits dans les deux nœuds fils w_e et $w_{\not e}$ sont obtenus comme décrit précédemment pour le nœud racine w^r.

Nous prenons en considération les améliorations proposées par Uno (2001) pour cet algorithme. Ces améliorations consistent à supprimer les arêtes qui n'appartiennent à aucun couplage parfait ou bien à imposer celles qui appartiennent à tous les couplages parfaits. Les améliorations de Uno permettent de diminuer la complexité de la recherche d'un nouveau couplage parfait, qui initialement est équivalente à une recherche en profondeur en $O(n + |E_{\bar{c}}|)$ (voir *DFS* (Tarjan, 1972)), pour finalement se faire en $O(n)$ (voir Uno (1997, 2001)).

En procédant de la sorte, toutes les solutions optimales du nœud v seront énumérées et la meilleure solution optimale pour le critère choisi sera retenue pour la séparation. Il est évident que si l'une de ces solutions correspond à une tournée alors elle est choisie et l'énumération des couplages parfaits est stoppée, et ce quel que soit le critère de choix de la solution équivalente.

L'énumération de toutes les solutions équivalentes est une opération coûteuse en temps, aussi il est préférable de procéder à cette recherche exhaustive au moment où le nœud correspondant à cette solution est éventuellement séparé (c'est-à-dire à l'étape 3 de l'algorithme 4.3). Il est néanmoins possible de chercher une solution équivalente juste après la résolution du problème associé à un nouveau nœud (c'est-à-dire juste après l'étape 9), et ce dans la perspective de trouver une tournée et ainsi mettre à jour la meilleure tournée courante.

La complexité de la recherche de tout nouveau couplage parfait est en $O(Nn)$, avec N le nombre de couplages parfaits dans le graphe $G = (U', V'; E_{\bar{c}})$. Aussi pour que la complexité de l'algorithme 4.3 en $O(n^3)$ ne soit pas modifiée, nous proposons de borner le nombre de couplages parfaits -donc de solutions équivalentes- énumérés. Cette borne peut dépendre de la taille du problème et doit être fixée dans ce cas à un seuil en $O(n^2)$ au plus.

4.3.4 Principe et mode de calcul de la borne $\tilde{B}$

Si un nœud v n'est pas coupé avec le test de la borne $f(v)$ (étape 10 de l'algorithme 4.3) alors nous proposons de calculer une nouvelle borne $\tilde{B}(v)$. Nous expliquons dans un premier temps le principe de cette borne puis nous proposons un algorithme pour la calculer.

4.3.4.a Principe de la borne $\tilde{B}$

La valeur de la borne $f(v)$ est la valeur de la solution optimale $x(v)$ et s'obtient par la recherche d'un plus court chemin améliorant dans le graphe d'écart associé au nœud v. La quantité $f(v)$ est donnée alors par la somme de la valeur optimale du nœud séparé et de la valeur du plus court chemin améliorant.

Il arrive souvent que le plus court chemin passe uniquement par les sommets de la sous-tournée $\mathcal{C}$ sur laquelle se base la séparation. La figure 4.4 montre un exemple de cette situation.

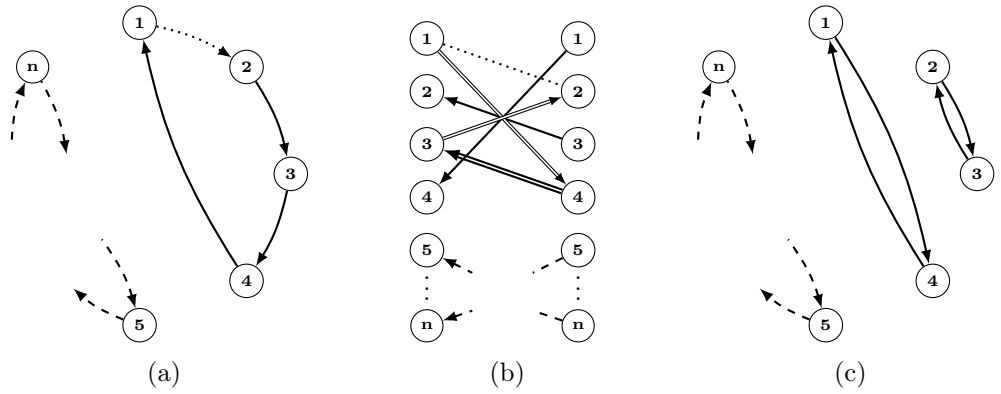


FIGURE 4.4 – Exemple d'un plus court chemin dans le graphe d'écart

La figure 4.4a correspond à la solution optimale d'un nœud séparé. Dans cet exemple la sous-tournée sur laquelle se base la séparation est donnée par $\mathcal{C} = \{(1,2),(2,3),(3,4),(4,1)\}$ et les arcs incidents aux sommets $\{5,\ldots,n\}$ sont dispersés sur une ou plusieurs autres sous-tournées. Nous expliquons ici le calcul de la solution optimale $x(v)$ du premier nœud fils v (étape 9 de l'algorithme 4.3). Nous supposons que l'ensemble des arêtes imposées $I(v)$ associé au nœud v est vide et que l'arc $(1,2)$ en pointillés correspond à la seule arête interdite, (i.e. $F(v) = \{(1,2)\}$). La figure 4.4b montre le graphe d'écart associé au nœud v dans lequel nous représentons uniquement les arcs arrières et en ligne double le plus court chemin entre les extrémités de l'arc interdit. La figure 4.4c donne la solution optimale $x(v)$ du problème $\mathcal{P}(\hat{X}(v), f)$ associée au nœud v. La nouvelle solution est obtenue par la suppression de l'arc interdit $(1,2)$ et de l'arc $(3,4)$ correspondant à l'arc arrière $(4,3)$ dans le plus court chemin améliorant, et par l'introduction des arcs directs $(1,4)$ et $(3,2)$ dans la solution optimale $x(v)$.

Dans ce cas, la sous-tournée $\mathcal{C}$ se divise en plusieurs nouvelles sous-tournées au lieu de fusionner avec d'autres sous-tournées pour en former une plus grande pour ainsi se rapprocher d'une tournée réalisable pour **ATSP**. De plus dans cette situation les sous-tournées autres que $\mathcal{C}$ ne sont pas cassées.

Le chemin améliorant qui engendrerait une solution optimale pour **ATSP** $\mathcal{P}(\hat{X}(v) \cap X, f)$ associé au nœud v passerait par chacune des sous-tournées de la solution optimale

du nœud séparé. Calculer une telle borne reviendrait à résoudre le problème du voyageur de commerce asymétrique. Cette borne est donc difficile à calculer.

Une autre borne consiste à imposer que le plus court chemin améliorant dans le graphe d'écart passe par au moins un sommet qui n'appartient pas à la sous-tournée $\mathcal{C}$. Calculer une telle borne est équivalent à chercher la meilleure solution parmi les solutions du problème d'affectation de l'ensemble $\hat{X}(v)$ qui satisfont une contrainte parmi les inégalités (4.1.c) de la formulation du problème du voyageur de commerce asymétrique que nous présentons au début de la section 4.1. Ce problème est une relaxation du problème du voyageur de commerce asymétrique $\mathcal{P}(\hat{X}(v) \cap X, f)$ et peut être formulé par :

$$\min f(x) : x \in \hat{X}(v), \text{ où } \sum_{i \in \mathcal{C}, j \notin \mathcal{C}} x_{ij} \geq 1 \tag{4.2}$$

Le problème (4.2) est équivalent à imposer qu'une arête au moins parmi les arêtes du graphe biparti dont une et une seule extrémité appartient à la sous-tournée de la séparation, à faire partie de la solution optimale du problème d'affectation.

Le problème (4.2) est équivalent, dans le pire cas, à la résolution de n^2 problèmes d'affectations. En effet, cette borne peut être obtenue en résolvant autant de problèmes d'affectation que d'arêtes qui possèdent une seule extrémité dans la sous-tournée $\mathcal{C}$. Chacune de ces arêtes sera contrainte à faire partie de l'affectation optimale. Forcer une arête à appartenir à la solution optimale désaffecte deux sommets. Aussi, les solutions optimales de ces problèmes d'affectation contraints s'obtiennent avec un effort de calcul équivalent à la recherche d'un plus court chemin améliorant entre les deux sommets libres, d'où la complexité du calcul de cette borne en $O(n^4)$ (au pire, n^2 plus courts chemins). La valeur d'une telle borne est donnée par la meilleure affectation parmi les affectations optimales. Le calcul de cette borne nécessite une préparation et peut s'avérer laborieux en pratique. Nous souhaitons obtenir une borne facile à calculer même si sa valeur est de moins bonne qualité. C'est pour cette raison que nous nous contentons d'une valeur approchée de l'optimum du problème 4.2.

Nous proposons la borne inférieure $\tilde{B}(v)$ qui s'obtient en sommant la valeur optimale du nœud séparé et la valeur d'un plus court chemin entre les extrémités de l'arc interdit dans le graphe d'écart associé au nœud v, avec la condition supplémentaire que ce chemin passe au moins par un sommet qui n'appartient pas à la sous-tournée $\mathcal{C}$. Nous acceptons des chemins non élémentaires, par conséquent le calcul de la borne $\tilde{B}(v)$ s'effectue avec une complexité en $O(n^2)$. Soulignons que la borne $\tilde{B}(v)$ est de meilleure qualité que la borne $f(v)$ (i.e. $\tilde{B}(v) \geq f(v)$).

4.3.4.b Mode de calcul de la borne $\tilde{B}$

Dans un graphe orienté, notons P_{ij} un plus court chemin entre les sommets i et j de coût π_{ij} et notons P_{ij}^k un plus court chemin entre les sommets i et j passant par le sommet k de coût π_{ij}^k. Il est trivial qu'un plus court chemin P_{ij}^k peut être obtenu par la

juxtaposition de deux plus courts chemins P_{ik} et P_{kj}. Le chemin P_{ij}^k ainsi obtenu n'est pas forcément élémentaire du fait que certains des sommets visités par le chemin P_{ik}, peuvent aussi être parcourus par le chemin P_{kj}.

Pour déterminer la borne $\tilde{B}(v)$, nous souhaitons trouver un plus court chemin entre les sommets i^* et j^*, passant par au moins un sommet qui n'appartient pas à la sous-tournée $\mathcal{C}$. Notons ce chemin $P_{i^*j^*}^{\mathcal{C}}$ de coût $\pi_{i^*j^*}^{\mathcal{C}} = \min_{k \notin \mathcal{C}} \pi_{i^*j^*}^k$, où $\pi_{i^*j^*}^k = \pi_{i^*k} + \pi_{kj^*}$ pour tout $k \notin \mathcal{C}$. La valeur de la borne $\tilde{B}(v)$ pour un nœud v est donnée par $\tilde{B}(v) = f(x^{k-1}) + \pi_{i^*j^*}^{\mathcal{C}}$, où x^{k-1} est la $k-1^{\text{ème}}$ meilleure affectation. Calculée de la sorte, la borne $\tilde{B}(v)$ a une complexité en $O(n^2)$. En effet, le calcul de la borne $\tilde{B}(v)$ est équivalent, dans le pire cas, à la recherche des plus courts chemins du sommet i^* vers tous les sommets qui n'appartiennent pas à la sous-tournée $\mathcal{C}$ à laquelle s'ajoute la recherche des plus courts chemins partant de ces sommets vers le sommet j^*. Les deux appels à l'algorithme de Dijkstra donnent la complexité du calcul de la borne $\tilde{B}(v)$ en $O(n^2)$.

Nous proposons de déterminer la solution optimale $x(v)$ et la valeur de la borne $\tilde{B}(v)$ avec l'algorithme 4.4. Cela permet d'économiser quelques efforts de calcul lors de la recherche de la valeur de la borne $\tilde{B}(v)$ mais ne permet pas de réduire sa complexité.

Soit le graphe d'écart $G = (U' \cup V'; E_d \cup E_a)$ associé à un nœud v, où $U' \subseteq U$ et $V' \subseteq V$ les ensembles de sommets qui ne sont pas incidents aux arcs imposés, et E_d et E_a les arcs directs et arrières. Le graphe d'écart est valué par les coûts réduits $\bar{c}_{ij}$ qui correspondent à la solution optimale x^{k-1} du nœud séparé.

Ici nous commentons l'algorithme 4.4 dont la partie la plus importante est la recherche de plus courts chemins que nous mettons en œuvre comme l'algorithme de Dijkstra, c'est-à-dire par *fixation d'étiquettes*. Le principe de l'algorithme 4.4 est d'associer deux familles d'étiquettes aux sommets de la sous-tournée $\mathcal{C}$. Ces étiquettes donnent les valeurs de deux plus courts chemins connus, l'un est quelconque et l'autre sort de la sous-tournée $\mathcal{C}$. De la même façon que pour l'algorithme de Dijkstra, une étiquette est fixée à chaque itération de l'algorithme et lorsqu'elle est associée à un chemin sortant de la sous-tournée $\mathcal{C}$, les étiquettes de la seconde famille sont mises à jour. Nous donnons dans ce qui suit plus de détails sur l'algorithme 4.4.

Soit π_j l'étiquette associée à un sommet j de V' et qui correspond à la valeur d'un plus court chemin connu entre le sommet i^* et le sommet j. Soit l'étiquette $\pi_j^{\mathcal{C}}$ associée à tout sommet j de l'ensemble $V_c = V' \cap \mathcal{C}$. L'étiquette $\pi_j^{\mathcal{C}}$ donne la valeur d'un plus court chemin connu entre le sommet i^* et le sommet j, tel que ce chemin parcourt au moins un sommet qui n'appartient pas à la sous-tournée $\mathcal{C}$.

Comme c'est le cas dans l'algorithme de Dijkstra, les étiquettes $\pi^{\mathcal{C}}$ et π sont initialisées à $+\infty$ dans les étapes 2 et 3 de l'algorithme 4.4.

Dans un graphe d'écart, pour tout arc arrière $(j, i) \in E_a$ on a les marques $\pi_i = \pi_j$ et $\pi_i^{\mathcal{C}} = \pi_j^{\mathcal{C}}$. C'est pour éviter cette redondance que nous ne représentons ici que les étiquettes de l'ensemble de sommets V'. Nous discutons de ce point dans les commentaires après la présentation de l'algorithme 4.4.

À chaque itération de l'algorithme (boucle **tant que** de l'étape 4), l'étiquette de valeur minimale parmi les étiquettes non fixées π ou $\pi^{\mathcal{C}}$ sera définitivement fixée. Cela signifie qu'un plus court chemin entre le sommet i^* et le sommet j correspondant à l'étiquette fixée a bien été trouvé et sa valeur est la valeur de cette étiquette (i.e. $\pi_{i^*j} = \pi_j$ ou $\pi^{\mathcal{C}}_{i^*j} = \pi^{\mathcal{C}}_j$).

Soit j' et j_c les sommets correspondant aux étiquettes minimales de π et $\pi^{\mathcal{C}}$ respectivement. Dans le cas où l'étiquette fixée est parmi les étiquettes π (i.e. $\pi_{j'} \leq \pi^{\mathcal{C}}_{j_c}$), alors le plus court chemin est quelconque, c'est-à-dire que ce chemin n'est pas contraint, il est donc forcément élémentaire. Dans le cas inverse, l'étiquette fixée appartient à $\pi^{\mathcal{C}}$ (i.e. $\pi_{j'} > \pi^{\mathcal{C}}_{j_c}$), alors le plus court chemin parcourt des sommets qui n'appartiennent pas à la sous-tournée $\mathcal{C}$.

Algorithme 4.4 : Plus court chemin sortant de la sous-tournée $\mathcal{C}$

1 $V_c \leftarrow V' \cap \mathcal{C}$; $i \leftarrow i^*$; $j' \in V' \setminus \{j^*\}, j_c \in V_c \setminus \{j^*\}$; $\delta \leftarrow 0$; $chem_{am} \leftarrow faux$; $fin \leftarrow faux$
2 **pour tous les** $j \in V'$ **faire** $\pi_j \leftarrow \infty$
3 **pour tous les** $j \in V_c$ **faire** $\pi^{\mathcal{C}}_j \leftarrow \infty$
4 **tant que** $fin = faux$ **faire**
5 **si** $\pi_{j'} \leq \pi^{\mathcal{C}}_{j_c}$ **alors**
6 `MajEtiq`(i, V', π, δ)
7 Déterminer j' tel que : $\pi_{j'} = \min \pi_j : j \in V'$
8 $V' \leftarrow V' \setminus \{j'\}$
9 **si** $i \notin \mathcal{C}$ **alors**
10 `MajEtiq`$(i, V_c, \pi^{\mathcal{C}}, \delta)$
11 Déterminer j_c tel que : $\pi^{\mathcal{C}}_{j_c} = \min \pi^{\mathcal{C}}_j : j \in V_c$
12 **sinon**
13 $V_c \leftarrow V_c \setminus \{j_c\}$
14 `MajEtiq`$(i, V_c, \pi^{\mathcal{C}}, \delta)$
15 Déterminer j_c tel que : $\pi^{\mathcal{C}}_{j_c} = \min \pi^{\mathcal{C}}_j : j \in V_c$
16 **si** $\pi_{j'} \leq \pi^{\mathcal{C}}_{j_c}$ **alors**
17 $\delta \leftarrow \pi_{j'}$
18 Déterminer i tel que : $x^{k-1}_{ij'} = 1$
19 **sinon**
20 $\delta \leftarrow \pi^{\mathcal{C}}_{j_c}$
21 Déterminer i tel que : $x^{k-1}_{ij_c} = 1$
22 **si** $f(x^{k-1}) + \delta \geq f(x^*)$ **alors** `// Le nœud` v `est coupé avec une borne`
23 $fin \leftarrow vrai$ `// Coupe de` $f(v)$ `si` $chem_{am} = faux$`, et coupe de` $\tilde{B}(v)$ `sinon`
24 **sinon**
25 **si** $j' = j^*$ *ou* $j_c = j^*$ **alors** `// Un plus court chemin` $P_{i^*j^*}$ `a été détecté`
26 **si** $\pi_{j'} \leq \pi^{\mathcal{C}}_{j_c}$ **alors** `//` $P_{i^*j^*}$ `est non-contraint`
27 $chem_{am} \leftarrow vrai$; $f(v) \leftarrow f(x^{k-1}) + \delta$
28 **sinon** `//` $P_{i^*j^*}$ `sort de la sous-tournée` $\mathcal{C}$`, (i.e.` $P^{\mathcal{C}}_{i^*j^*} = P_{i^*j^*}$`)`
29 $fin \leftarrow vrai$; $\tilde{B}(v) \leftarrow f(x^{k-1}) + \delta$; $\mathcal{L} \leftarrow \mathcal{L} \cup \{v\}$

La mise à jour des étiquettes est réalisée grâce à la procédure MajEtiq décrite dans l'algorithme 4.5.

Algorithme 4.5 : Procédure de mise à jour des étiquettes

```
1 Procédure MajEtiq(i, V, π, δ)
2     pour tous les j ∈ V faire
3         si δ + c̄_ij < π_j alors  π_j ← δ + c̄_ij
```

Les étiquettes π évoluent comme les étiquettes de l'algorithme de Dijkstra, c'est-à-dire qu'elles sont mises à jour chaque fois que l'étiquette minimale appartient à la famille des étiquettes π. La nouvelle étiquette minimale est repérée puis le sommet j' correspondant à la nouvelle étiquette minimale est extrait de l'ensemble V', et c'est ainsi que cette étiquette est rendue définitive. La preuve qu'une telle étiquette ne peut plus être améliorée s'appuie principalement sur le fait que les valuations du graphe d'écart sont positives.

Les étiquettes $\pi^{\mathcal{C}}$ ont une évolution différente de celles de la famille π. En effet, les étiquettes $\pi^{\mathcal{C}}$ sont mises à jour dans deux situations. La première est naturelle et correspond à l'étape 10, où les étiquettes de $\pi^{\mathcal{C}}$ sont mises à jour lorsque $\pi_{j'} \leq \pi^{\mathcal{C}}_{j_c}$, et que le sommet j' qui correspond à l'étiquette minimale dans le vecteur $\pi_{j'}$ n'appartient pas à la sous-tournée $\mathcal{C}$. La seconde situation où les étiquettes de $\pi^{\mathcal{C}}$ sont mises à jour survient lorsque $\pi_{j'} > \pi^{\mathcal{C}}_{j_c}$, i.e. l'étiquette minimale correspond à un chemin qui parcourt des sommets qui n'appartiennent pas à $\mathcal{C}$. Par définition les composantes du vecteur $\pi^{\mathcal{C}}$ sont les valeurs des chemins qui parcourent un sommet qui n'appartient pas à la composante $\mathcal{C}$. Par conséquent, un chemin construit par le prolongement d'un tel chemin passe forcément par un sommet qui n'appartient pas à la composante $\mathcal{C}$. Contrairement à la première situation, ici l'étiquette est minimale, elle peut donc être considérée comme définitive, aussi le sommet qui lui correspond est extrait de l'ensemble V_c. Il est possible de prouver que cette étiquette correspond dans cette situation à un plus court chemin puisque les valuations du graphe sont positives.

Si nous avions défini des étiquettes pour les deux sous-ensembles de sommets U' et V' du graphe d'écart, à chaque itération de l'algorithme 4.4 deux étiquettes seraient fixées, à savoir, l'étiquette de j' ou j_c selon le cas et l'étiquette du sommet i qui est le seul sommet suivant de j' ou de j_c dans le graphe d'écart. En effet, par la construction du graphe d'écart tout sommet j de V' a un seul sommet suivant i, tel que (j, i) est un arc arrière. Aussi le sommet i peut être obtenu grâce à la solution du nœud séparé x^{k-1}. C'est donc à partir de ce sommet i que les mises à jour des étiquettes sont effectuées.

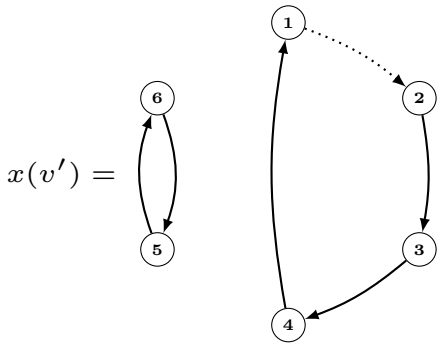
La valeur δ dans l'algorithme est la valeur de l'étiquette minimale parmi toutes les étiquettes non fixées. C'est donc une borne inférieure sur les valeurs des deux plus courts chemins $P_{i^*j^*}$ et $P^{\mathcal{C}}_{i^*j^*}$, si toutefois ces chemins n'ont pas encore été trouvés. Cela explique le test de l'étape 22 qui est une anticipation sur les coupes des bornes $f(v)$ et $\tilde{B}(v)$. En effet, la variable booléenne $chem_{am}$ vaut $vrai$ si le plus court chemin améliorant (non

contraint) a été trouvé, et si le test de l'étape 22 coupe le nœud v avant d'avoir déterminé la valeur de la solution optimale cela stoppe l'algorithme 4.4 et le nœud est coupé grâce à la borne $f(v)$, sinon le nœud v est coupé avec la valeur de la borne $\tilde{B}(v)$.

Si l'algorithme ne s'arrête pas sur une coupe de borne, alors il s'arrête lorsqu'un plus court chemin sortant de la sous-tournée $\mathcal{C}$ a été trouvé (étape 28), et cela après avoir détecté le plus court chemin améliorant à l'étape 26. Le chemin améliorant est détecté avant le chemin sortant grâce aux inégalités larges des étapes 5 et 16.

Les étapes les plus difficiles pour chaque itération de l'algorithme 4.4 sont les mises à jour des étiquettes qui se font avec une complexité en $O(n)$ (voir l'algorithme 4.5). Ceci donne la complexité de l'algorithme 4.4 en $O(n^2)$ ($2n$ étiquettes fixées dans le pire des cas, avec une complexité en $O(n)$ à chaque itération).

Exemple. Nous montrons dans la figure 4.5 la solution optimale $x(v')$ du nœud séparé v' et la matrice des coûts réduits $\bar{c}$ qui lui est associée. La solution $x(v')$ est composée de deux sous-tournées. Supposons que la séparation soit basée sur les arcs de la sous-tournée $\{1, 2, 3, 4, 1\}$ et que l'arc $(1, 2)$, en pointillés dans la figure 4.5, soit l'arc interdit pour créer le premier fils v du nœud v'. Selon l'implémentation choisie pour l'algorithme 4.4, l'arc $(1, 2)$ est supprimé de l'ensemble des arcs arrières du graphe d'écart associé au nœud v ou bien le coût réduit $\bar{c}_{12}$ est mis à l'infini (i.e. $\bar{c}_{12} = \infty$). La figure 4.6 montre le déroulement

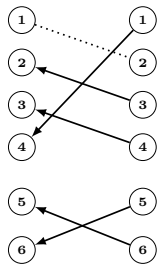


$$\bar{c} = \begin{pmatrix} \infty & \not{0}^{\infty} & 6 & 1 & 1 & 2 \\ 7 & \infty & 0 & 1 & 1 & 6 \\ 4 & 1 & \infty & 0 & 4 & 9 \\ 0 & 7 & 2 & \infty & 2 & 1 \\ 2 & 5 & 6 & 10 & \infty & 0 \\ 3 & 4 & 9 & 9 & 0 & \infty \end{pmatrix}$$

FIGURE 4.5 – Matrice des coûts réduits et solution du nœud séparé

de l'algorithme 4.4. Elle est composée à gauche du graphe d'écart associé au nœud v, pour lequel nous représentons uniquement les arcs arrières, et à droite du tableau dont chacune des lignes correspond à un sommet du graphe d'écart qui montre l'évolution des étiquettes π et $\pi^{\mathcal{C}}$. Les chemins améliorant et sortant doivent relier les extrémités de l'arc interdit $(i^*, j^*) = (1, 2)$. L'itération 0 (*it* 0) est une étape d'initialisation où toutes les composantes de π et $\pi^{\mathcal{C}}$ valent ∞. L'étiquette à développer ici est $i = 1$ et la valeur du plus court chemin est initialisée à $\delta = 0$.

Une étiquette encadrée dans la figure 4.6 indique qu'elle est minimale. Une telle étiquette est rendue définitive et ceci est indiqué dans la figure 4.6 en soulignant les étiquettes pour toutes les itérations qui suivent celles où elles ont été fixées (encadrées).



it 0 $i=1$ $\delta=0$		*it*1 $i=3$ $\delta=1$		*it* 2 $i=6$ $\delta=1$		*it* 3 $i=1$ $\delta=2$		*it* 4 $i=5$ $\delta=2$		*it* 5 $i=4$ $\delta=4$		*it* 6 $i=4$ $\delta=4$		*it* 7 $i=1$ $\delta=5$	
π	$\pi^{\mathcal{C}}$	π	$\pi^{\mathcal{C}}$	π	$\pi^{\mathcal{C}}$	π	$\pi^{\mathcal{C}}$	π	$\pi^{\mathcal{C}}$	π	$\pi^{\mathcal{C}}$	π	$\pi^{\mathcal{C}}$	π	$\pi^{\mathcal{C}}$
∞	∞	∞	∞	5	∞	4	4	4	4	$\boxed{4}$	4	$\underline{4}$	$\boxed{4}$	$\underline{4}$	$\underline{4}$
∞	∞	∞	∞	2	∞	$\boxed{2}$	5	$\underline{2}$	5	$\underline{2}$	5	$\underline{2}$	5	$\underline{2}$	$\boxed{5}$
∞	∞	6	∞	6	∞	6	10	6	10	6	8	6	8	6	6
∞	∞	$\boxed{1}$	∞	$\underline{1}$	∞	$\underline{1}$	10	$\underline{1}$	10	$\underline{1}$	10	$\underline{1}$	10	$\underline{1}$	10
∞		1		$\boxed{1}$		$\underline{1}$		$\underline{1}$		$\underline{1}$		$\underline{1}$		$\underline{1}$	
∞		2		2		2		$\boxed{2}$		$\underline{2}$		$\underline{2}$		$\underline{2}$	

FIGURE 4.6 – Exemple de déroulement de l'algorithme 4.4

Les étiquettes $\pi^{\mathcal{C}}$ sont mises à jour pour la première fois à l'itération 3, puisque le sommet développé à l'itération 3 est $i = 6$ qui n'appartient pas à la sous-tournée de séparation.

À l'itération 3, l'étiquette minimale correspond à l'extrémité terminale de l'arc interdit. Le plus court chemin améliorant a donc été trouvé et il vaut $\delta = 2$. La valeur de la solution optimale du nœud v est donnée $f(x(v)) = f(x(v')) + \delta = f(x(v')) + 2$

Le chemin sortant qui donne la valeur de la borne $\tilde{B}(v)$ n'est obtenu qu'à l'itération 7, dès lors que l'étiquette minimale correspond une nouvelle fois à l'extrémité terminale de l'arc interdit. La valeur de la borne est donnée par $\tilde{B}(v) = f(x(v')) + \delta = f(x(v')) + 5 > f(x(v))$.

Nous montrons dans la figure 4.7a les deux chemins améliorant (en ligne simple) et sortant (en ligne double) obtenus avec l'algorithme 4.4 dans le graphe d'écart et nous représentons dans les figures 4.7b et 4.7a les solutions obtenues à l'aide de ces deux chemins.

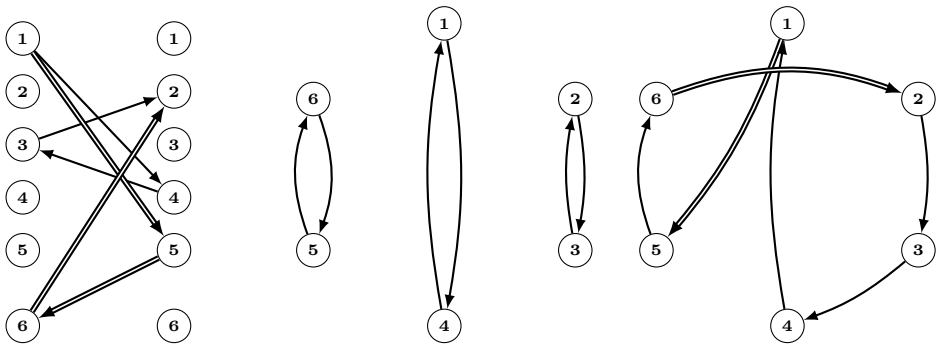


FIGURE 4.7 – Plus courts chemins résultant de l'algorithme 4.4

La solution optimale $x(v)$ du nœud v contient 3 sous-tournées alors que la solution obtenue avec le plus court chemin sortant correspond à une tournée qui de plus est de coût minimal parmi toutes les tournées du nœud v. Précisons tout de même que le chemin sortant ne conduit pas une solution réalisable s'il n'est pas élémentaire.

4.3.5 Gestion de l'ensemble des arêtes imposées

Lorsque des arêtes imposées $I(v)$ du nœud v forment un chemin de longueur inférieure à $n-1$ dans le graphe du problème du voyageur de commerce asymétrique, où n est le nombre de sommets du graphe, comme c'est le cas dans la figure 4.8, l'arête correspondant à l'arc retour (en pointillés sur la figure 4.8) allant de l'extrémité finale du chemin vers son extrémité initiale doit être interdite.

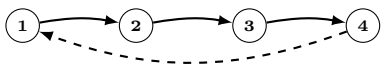


FIGURE 4.8 – Arc interdit induit par un chemin imposé

Ces arcs ne sont pas forcément dans l'ensemble $F(v)$. Afin d'améliorer l'utilisation de l'espace mémoire, nous proposons de stocker l'ensemble des arêtes imposées sous la forme d'un ensemble de paires qui correspondent aux extrémités initiales et finales des chemins induits par l'ensemble des arêtes imposées $I(v)$; la totalité du chemin peut être reconstituée à l'aide de la solution optimale $x(v)$. Les nouveaux arcs interdits peuvent être obtenus en inversant les paires correspondant à la représentation de l'ensemble $I(v)$.

4.4 Résultats expérimentaux

Nous présentons dans cette section les résultats expérimentaux que nous avons obtenus. Avant d'aborder les détails de ces résultats, nous commentons dans la section 4.4.1 les instances que nous résolvons et les variantes des algorithmes que nous obtenons par la combinaison de différents choix algorithmiques. Nous discutons aussi de la démarche et de l'organisation des expérimentations dans la section 4.4.1.

4.4.1 Remarques préliminaires sur la partie expérimentale

4.4.1.a Instances

Contrairement à la recherche de solution de compromis pour le problème d'affectation multiobjectif, le problème du voyageur de commerce asymétrique est très étudié, par conséquent, des instances réelles de différents types et différentes origines sont dispo-

nibles et sont rassemblées dans la librairie d'instances TSPLIB [1]. Ceci nous permet de vérifier l'efficacité de nos algorithmes en comparant nos résultats à ceux présents dans la littérature sur les mêmes instances.

Les instances de TSPLIB sont données sous la forme de matrices de coûts carrées. Les coefficients de ces matrices sont entiers avec des valeurs très grandes pour les éléments de la diagonale principale. Les noms de ces instances sont donnés par les initiales des auteurs qui les ont proposées. Nous discuterons plus en détail de l'origine de ces instances lorsque nous présenterons nos résultats expérimentaux. Nous avons procédé à une analyse statistique sommaire de ces instances à partir des valeurs présentées dans la table 4.1.

Instances	n	min	max	Δ	E	σ	$\frac{\sigma}{E}$	$cor_{c_{ij},c_{ji}}$	$E(c_{ij}-c_{ji})$
ftv33	34	7	332	325	128	59	0.46	0.67	2.61
ftv35	36	7	332	325	135	61	0.45	0.62	0.79
ftv38	39	7	332	325	133	60	0.45	0.65	1.40
ftv44	45	7	332	325	137	61	0.44	0.68	1.62
ftv47	48	7	348	341	142	63	0.44	0.63	0.03
ftv55	56	6	324	318	132	59	0.44	0.69	1.59
ftv64	65	5	348	343	135	61	0.45	0.68	-1.24
ftv70	71	5	348	343	138	62	0.45	0.67	-0.08
ftv170	171	4	368	364	154	68	0.44	0.76	-2.31
rbg323	323	0	33	33	19	7	0.39	0.05	-0.40
rbg358	358	0	33	33	19	8	0.42	0.02	-0.67
rbg403	403	0	33	33	19	8	0.43	0.04	-0.51
rbg443	443	0	33	33	19	8	0.42	0.03	-0.39
ft53	53	21	1834	1813	493	376	0.76	-0.28	219.78
ft70	70	331	2588	2257	1030	423	0.41	-0.10	239.21
kro124p	100	81	4545	4464	1910	923	0.48	0.98	-0.68
ry48p	48	54	2782	2728	1139	618	0.54	0.99	1.88
p43	43	0	5160	5160	594	1526	2.57	0.96	981.86

TABLE 4.1 – Statistiques sur les instances de la TSPLIB

La première et la deuxième colonnes de la table 4.1 donnent, respectivement, le nom et la taille des instances. Les autres mesures de la table 4.1 sont calculées sans prendre en compte les valeurs des coefficients de la diagonale de la matrice des coûts. Dans l'ordre, les colonnes min, max et Δ donnent le coefficient minimum, maximum et l'écart entre ces deux valeurs pour chaque instance. Les colonnes E, σ et $\frac{\sigma}{E}$ sont la moyenne, l'écart-type et l'*écart-type relatif*. Cette dernière mesure (écart-type relatif qui est aussi appelé *coefficient de variation*) est très appréciée lors de la comparaison de la dispersion de séries de données qui diffèrent par leur ordre de grandeur.

L'avant-dernière colonne de la table 4.1 $cor_{c_{ij},c_{ji}}$ est le coefficient de corrélation entre deux vecteurs, notons les CU et CL, qui contiennent respectivement les coefficients au-dessus et en-dessous de la diagonale de la matrice des coûts. Les vecteurs CU et CL sont

1. http ://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/

construits de telle sorte que la position du coût c_{ij} dans CU corresponde à celle du coût c_{ji} dans CL pour tout $i < j$. Le coefficient de corrélation est proche de 1 lorsque les coûts c_{ji} forment une fonction affine croissante des coûts c_{ij}, et cette mesure est proche de -1 dans le cas où ces coûts sont liés par une fonction affine décroissante. Les valeurs intermédiaires de $cor_{c_{ij},c_{ji}}$ renseignent sur le degré de dépendance linéaire entre les coûts des vecteurs CU et CL. Lorsque la valeur de $cor_{c_{ij},c_{ji}}$ est loin de 1, c'est-à-dire lorsque les vecteurs CU et CL ne sont pas positivement et fortement corrélés, alors l'instance est asymétrique. Dans le cas contraire, c'est-à-dire $cor_{c_{ij},c_{ji}}$ est proche de 1, les instances ne sont pas forcément symétriques ; c'est le cas par exemple lorsque $c_{ij} = 2 * c_{ji}$ pour tout $i < j$. La colonne $E(c_{ij} - c_{ji})$ sert à distinguer les instances positivement corrélées symétriques des asymétriques. $E(c_{ij} - c_{ji})$ représente la moyenne des différences entre les vecteurs CU et CL. Ainsi, une instance est considérée *quasi-symétrique* si $cor_{c_{ij},c_{ji}}$ est très proche de 1 et $E(c_{ij} - c_{ji})$ très petit. Les instances qui ont une moyenne des écarts très importante sont asymétriques.

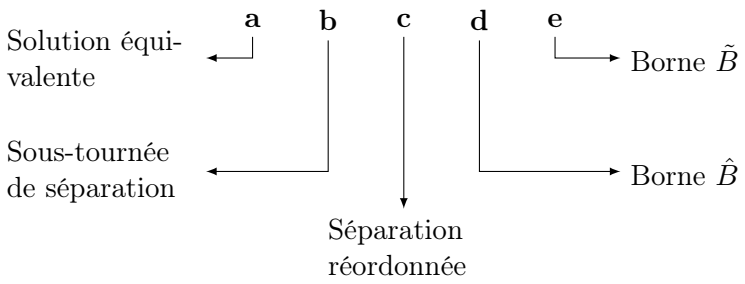
Il existe sans doute des mesures statistiques plus élaborées pour juger de la dispersion des coûts d'une instance ou de son asymétrie, mais nous souhaitons des mesures simples et faciles à interpréter. La finalité de cette étude statistique est de classifier les instances en familles et d'identifier les améliorations qui conviennent à chaque famille d'instances grâce à l'analyse des résultats expérimentaux.

4.4.1.b Variantes

Lors des expérimentations, nous avons observé que toutes les variantes utilisant la recherche d'une bonne tournée pour le nœud séparé suivant le principe du *patching algorithm* (voir section 4.3.1) sont meilleures que leurs équivalentes qui n'utilisent pas cette amélioration, et ce pour toutes les instances testées. C'est pour cette raison que toutes les variantes que nous présentons dans cette section utilisent la recherche d'une bonne tournée (étape 2 de l'algorithme 4.3).

La recherche d'un plus court chemin dans un graphe d'écart est le meilleur moyen pour résoudre les problèmes d'affectation associés aux nœuds de l'arborescence. Par conséquent, toutes les variantes utilisent ce mode de résolution. Le test de la borne f (étape 10 de l'algorithme 4.3) est lui aussi présent dans toutes nos versions, puisque le calcul de cette borne ne nécessite aucun effort (valeur optimale du nœud).

Une variante de notre procédure est définie par la donnée des 5 choix algorithmiques qui la caractérisent. Chaque variante est désignée par 5 symboles (**abcde**) qui informent sur ses caractéristiques.



Nous donnons dans ce qui suit les valeurs que peuvent prendre les caractères **a**, **b**, **c**, **d** et **e**.

a) Choix d'une solution équivalente (section 4.3.3) :

- $\mathbf{\not{E}}$ Ne pas chercher de solution équivalente
- $\mathbf{E_f}$ Fusionner les sous-tournées
- \- Énumération de solutions équivalentes et choix de la solution contenant :
  - $\mathbf{E_n}$ le moins de sous-tournées
  - $\mathbf{E_t}$ la plus petite sous-tournée
  - $\mathbf{E_l}$ la sous-tournée ayant le moins d'arcs libres
  - $\mathbf{E_c}$ la sous-tournée de coût min

b) Critère de choix de la sous-tournée de séparation (section 4.3.2) :

- $\mathbf{ST_t}$ la plus petite sous-tournée
- $\mathbf{ST_l}$ la sous-tournée ayant le moins d'arcs libres
- $\mathbf{ST_c}$ la sous-tournée de moindre coût

c) Séparation réordonnée (section 3.2.4) : $\mathbf{\not{R}}$ Non $\mathbf{R}$ Oui

d) Borne $\hat{B}$ (section 3.2.3) : $\mathbf{\not{\hat{B}}}$ Non $\mathbf{\hat{B}}$ Oui

e) Borne $\tilde{B}$ (section 4.3.4) : $\mathbf{\not{\tilde{B}}}$ Non $\mathbf{\tilde{B}}$ Oui

Il existe 144 ($6 \times 3 \times 2 \times 2 \times 2$) combinaisons possibles, par conséquent 144 variantes sont envisageables. Notons que la variante $\mathbf{E_f ST_l \not{R} \not{\hat{B}} \not{\tilde{B}}}$ correspond à l'algorithme proposé par (Carpaneto et al., 1995).

4.4.1.c Organisation des expérimentations

Compte tenu du nombre important de versions, ce qui rend les résultats expérimentaux difficiles à interpréter, nous procédons aux expérimentations sur chaque famille d'instances en deux temps. Dans un premier temps nous résolvons les instances d'une famille à l'aide de toutes les variantes. Chaque version de l'algorithme se verra associer un *vecteur temps* dont chaque composante correspond au temps CPU nécessaire pour la résolution d'une instance. À la manière d'un problème multicritère, on considère qu'une variante V_1 domine une autre variante V_2, si le vecteur temps associé à V_1 domine le vecteur temps associé à V_2 au sens de la minimisation (voir la section 1.1.4). À l'issue de cette première phase, seules les variantes non-dominées seront conservées pour le reste des expérimentations.

Dans un second temps, et après la réduction du nombre de variantes, nous évaluons l'efficacité de ces versions non-dominées en relevant d'autres mesures de performance telles que le nombre de solutions énumérées et la taille des arborescences ; ceci nous permettra de procéder à une analyse fine de ces variantes.

Nous disposons pour ces tests d'une machine dotée d'un processeur Intel® Core™ i7 CPU Q 840 @ 1.87GHz x 8 et d'une RAM de 16 GB qui tourne sous le système d'exploitation Ubuntu, version 12.04, 64 bits. Les algorithmes sont codés en langage C et les fichiers binaires sont générés en utilisant le compilateur gcc avec l'option d'optimisation -O3.

4.4.2 Résultats expérimentaux

Nous analysons dans la section courante les résultats obtenus lors de la résolution de chacune des familles d'instances de la TSPLIB. Nous nous référons à la table 4.1 lors de la description des instances.

4.4.2.a Résultats expérimentaux pour les instances ftv

Les 9 instances **ftv**, décrites initialement par Fischetti et al. (1994) et adaptées au problème du voyageur de commerce asymétrique dans Fischetti et Toth (1997), viennent d'un problème de livraison de produits pharmaceutiques dans le centre-ville de Bologne (Italie). Les sommets du graphe complet modélisant ce problème représentent des lieux pertinents de la ville (un dépôt, un client ou un grand carrefour). Un arc (i, j) est muni de la valuation c_{ij} valant la longueur d'un plus court chemin entre les deux lieux i et j. Le plus court chemin est obtenu en prenant en considération les sens interdits.

Par construction ces instances sont *faiblement* asymétriques. En effet, bien que la distance d'un plus court chemin d'un lieu i à un lieu j ne soit pas forcément équivalente à la longueur d'un plus court chemin dans le sens inverse, la différence entre les deux distances est souvent faible. Ceci est confirmé dans la table 4.1 par le coefficient de corrélation $cor_{c_{ij},c_{ji}}$ qui est modérément proche de 1 et par la moyenne des écarts qui est

très réduite. La dispersion des coûts des instances **ftv** est aussi intermédiaire par rapport aux autres instances comme on peut l'observer grâce au coefficient de dispersion ($\frac{\sigma}{E}$) et à l'écart entre le coût maximal et le coût minimal (Δ) dans la table 4.1.

Nous commentons ici les résultats obtenus à l'issue de la première étape des expérimentations pour les instances **ftv**. Ici les 9 instances **ftv** on été résolues à l'aide des 144 variantes de notre algorithme. L'analyse des temps de calcul a montré que l'on peut isoler 13 variantes non-dominées parmi toutes les versions possibles. Les temps de calcul en secondes de ces variantes et de la variante de Carpaneto et al. (1995) 14. $\mathbf{E_fST_lR\hat{B}\tilde{B}}$ sont présentés dans la table 4.2.

Variantes	**ftv33**	**ftv35**	**ftv38**	**ftv44**	**ftv47**	**ftv55**	**ftv64**	**ftv70**	**ftv170**
1. $\mathbf{E_fST_cR\hat{B}\tilde{B}}$	0.01	0.01	0.01	0.01	0.02	0.04	0.06	0.28	35.32
2. $\mathbf{E_fST_c\not{R}\hat{B}\tilde{B}}$	0.00	0.03	0.02	0.00	0.03	0.06	0.09	0.31	31.57
3. $\mathbf{E_fST_c\not{R}\hat{B}\not{\tilde{B}}}$	0.00	0.02	0.02	0.00	0.04	0.10	0.12	0.31	33.52
4. $\mathbf{E_fST_c\not{R}\not{\hat{B}}\tilde{B}}$	0.02	0.02	0.02	0.01	0.03	0.06	0.10	0.37	35.02
5. $\mathbf{E_fST_c\not{R}\not{\hat{B}}\not{\tilde{B}}}$	0.00	0.02	0.02	0.00	0.03	0.12	0.15	0.34	38.03
6. $\mathbf{E_fST_cR\not{\hat{B}}\tilde{B}}$	0.01	0.02	0.02	0.00	0.03	0.06	0.08	0.37	39.50
7. $\mathbf{E_fST_tR\hat{B}\tilde{B}}$	0.00	0.01	0.01	0.00	0.01	0.13	0.03	0.20	47.96
8. $\mathbf{E_fST_t\not{R}\hat{B}\tilde{B}}$	0.00	0.01	0.02	0.00	0.02	0.12	0.06	0.22	56.81
9. $\mathbf{E_fST_lR\hat{B}\tilde{B}}$	0.00	0.00	0.01	0.01	0.01	0.16	0.03	0.25	53.83
10. $\mathbf{E_fST_l\not{R}\not{\hat{B}}\not{\tilde{B}}}$	0.00	0.02	0.00	0.01	0.02	0.17	0.07	0.30	56.05
11. $\mathbf{\not{E}ST_cR\not{\hat{B}}\not{\tilde{B}}}$	0.00	0.02	0.02	0.00	0.04	0.10	0.10	1.14	80.28
12. $\mathbf{\not{E}ST_cR\hat{B}\tilde{B}}$	0.00	0.02	0.01	0.01	0.04	0.09	0.11	1.22	81.81
13. $\mathbf{\not{E}ST_tR\not{\hat{B}}\not{\tilde{B}}}$	0.00	0.00	0.01	0.00	0.03	0.33	0.05	0.75	104.80
14. $\mathbf{E_fST_l\not{R}\not{\hat{B}}\not{\tilde{B}}}$ [a]	0.00	0.02	0.02	0.00	0.04	0.19	0.09	0.30	60.99

a. version de Carpaneto et al. (1995)

TABLE 4.2 – Temps CPU des variantes non-dominées pour les instance **ftv** (secondes)

Les résultats de la variante de Carpaneto et al. (1995) (14. $\mathbf{E_fST_l\not{R}\not{\hat{B}}\not{\tilde{B}}}$) sont consultables dans Roberti et Toth (2012). Nous préférons considérer les temps de calcul que nous obtenons avec nos codes, exécutés sur notre machine. Ceci donne plus d'homogénéité aux temps CPU. De plus les résultats que nous obtenons pour la version de Carpaneto et al. (1995) sont bien meilleurs que ceux présentés par Roberti et Toth (2012). Cependant, nos résultats montrent que cette variante est dominée par les variantes 7. $\mathbf{E_fST_tR\hat{B}\tilde{B}}$ et 8. $\mathbf{E_fST_t\not{R}\hat{B}\tilde{B}}$.

Les petites instances **ftv33**, **ftv35**, **ftv38**, **ftv44** et **ftv47** sont résolues dans des temps très faibles (entre 0.00 et 0.04 secondes). Nous avons procédé à des tests, dont les résultats ne sont pas reportés ici, qui consistent à faire tourner une variante quelconque de notre algorithme dix fois sur une des petites instances citées précédemment. Ces nouveaux tests nous ont permis de relever les erreurs dans les valeurs des temps de

calcul obtenus par l'exécution d'une variante plusieurs fois sur la même instance. Nous avons constaté que pour ces instances la moyenne des temps CPU est du même ordre de grandeur que l'erreur qui entache ces valeurs, c'est-à-dire que la valeur du temps CPU varie d'une quantité équivalente -voire même supérieure- à la moyenne des valeurs observées. Nos observations n'ont pas changé lorsqu'on a utilisé des variantes ou des instances différentes pour ces tests. En raison de la forte variation et des faibles valeurs des temps CPU observés pour les instances **ftv33**, **ftv35**, **ftv38**, **ftv44** et **ftv47**, nous avons décidé de les écarter de l'analyse des résultats des instances **ftv**.

Nous nous sommes donc focalisés uniquement sur les instances **ftv55**, **ftv64**, **ftv70** et **ftv170** et seules 4 variantes demeurent non-dominées : 1. $\mathbf{E_fST_cR\hat{B}\tilde{B}}$, 2. $\mathbf{E_fST_cR\hat{B}\tilde{B}}$, 7. $\mathbf{E_fST_tR\hat{B}\tilde{B}}$ et 8. $\mathbf{E_fST_tR\hat{B}\tilde{B}}$.

Nous présentons dans la table 4.3 les temps CPU, le nombre de solutions énumérées et les tailles des arborescences de recherche des quatre variantes non-dominées et la variante de Carpaneto et al. (1995). Afin de ne pas encombrer la table 4.3 nous nous sommes contentés des résultats des plus grandes instances **ftv70** et **ftv170**, puisque ces résultats suffisent pour montrer les performances des variantes retenues.

	Temps CPU (s.)		Solutions énumérées		Taille arborescence	
Variantes	**ftv70**	**ftv170**	**ftv70**	**ftv170**	**ftv70**	**ftv170**
1. $\mathbf{E_fST_cR\hat{B}\tilde{B}}$	0.28	35.32	3 349	102 045	8 554	241 137
2. $\mathbf{E_fST_cR\hat{B}\tilde{B}}$	0.31	31.57	2 427	83 383	9874	203 338
7. $\mathbf{E_fST_tR\hat{B}\tilde{B}}$	0.20	47.96	1 759	135 105	5 717	345 110
8. $\mathbf{E_fST_tR\hat{B}\tilde{B}}$	0.22	56.81	1 659	149 894	6 378	382 630
14. $\mathbf{E_fST_lR\hat{B}\tilde{B}}$ [a]	0.30	60.99	2 628	156 912	13 046	438 199

a. version de Carpaneto et al. (1995)

TABLE 4.3 – Résultats pour les instances **ftv**

La variante 2. $\mathbf{E_fST_cR\hat{B}\tilde{B}}$ semble être la plus performante ; le meilleur temps de calcul pour la plus grande instance **ftv170** est obtenu avec cette variante 2. $\mathbf{E_fST_cR\hat{B}\tilde{B}}$. Le nombre de solutions énumérées et la taille des arborescences expliquent en grande partie la performance de cette variante.

La variante 8. $\mathbf{E_fST_tR\hat{B}\tilde{B}}$ ne semble pas compétitive bien qu'elle soit non-dominée. En effet, elle est battue par la variante 7. $\mathbf{E_fST_tR\hat{B}\tilde{B}}$ sur huit parmi les neuf instances résolues (voir la table 4.2), et ne bat la variante 7. $\mathbf{E_fST_tR\hat{B}\tilde{B}}$ que d'un centième de seconde sur la l'instance **ftv55**.

Nous pouvons remarquer que les quatre variantes non-dominées contiennent les deux tests de bornes $\hat{\mathbf{B}}$ et $\tilde{\mathbf{B}}$. La séparation réordonnée apparaît dans deux variantes non-dominées. Nous concluons que globalement, nos propositions améliorent l'énumération ordonnée pour les instances **ftv**. L'impact de nos améliorations est remarquable par

comparaison avec la version de Carpaneto et al. (1995) 14. $\mathbf{E_f ST_l}\overline{\overline{RBB}}$, qui en plus d'être plus rapides, utilisent moins d'espace de stockage.

Notons aussi qu'aucune variante qui utilise la recherche de solutions équivalentes par l'énumération de couplages ($\mathbf{E_n}$, $\mathbf{E_t}$, $\mathbf{E_l}$ et $\mathbf{E_c}$) n'est retenue dans les variantes non-dominées. Cela s'explique par le fait que ces modes de recherche de solutions équivalentes sont trop laborieux par rapport à la difficulté des instances **ftv**. En revanche, toutes les variantes non-dominées utilisent la fusion des sous-tournées ($\mathbf{E_f}$) proposée par Carpaneto et al. (1995).

4.4.2.b Résultats expérimentaux pour les instances rbg

Les quatre instances **rbg** proviennent d'un problème d'ordonnancement de tâches d'un *transtockeur*. Ce problème a été modélisé par Ascheuer (1986) comme la recherche d'un *plus court chemin hamiltonien dans un graphe complet*. Chaque sommet du graphe représente un déplacement du transtockeur d'une étagère *origine* à une autre étagère *cible* d'un *palettier*. Le coût c_{ij} est la distance séparant l'étagère cible de la tâche i et l'étagère origine de la tâche j, autrement dit, c'est la distance que le transtockeur parcourt pour se placer à l'origine de la tâche j après avoir réalisé la tâche i, d'où l'asymétrie des instances **rbg** et l'absence de corrélation entre le coût c_{ij} et le coût c_{ji} comme le montre le coefficient de corrélation ($cor_{c_{ij},c_{ji}}$) dans la table 4.1.

Par ailleurs, le transtockeur est un dispositif d'optimisation de l'espace de stockage, par conséquent les distances entre les étagères sont très réduites. Ainsi, les coûts c_{ij} appartiennent à des intervalles très restreints ($[0, 33]$) et le coefficient de variation des instances **rbg** est faible. Toutes ces particularités associées à la taille des instances **rbg** ($300 \leq n \leq 450$) engendrent des instances possédant un nombre important de solutions équivalentes pour toute valeur de la fonction objectif.

Notons que le problème de gestion de transtockeur est résolu en temps réel (*on-line*). Ce mode de résolution justifie la recherche d'un algorithme très rapide et donne de l'intérêt à toute milliseconde économisée dans les temps de calcul. La table 4.4 donne les temps de calcul des variantes non-dominées pour les instances **rbg**.

Ces instances sont très faciles à résoudre. La raison est que la bonne tournée initiale que nous calculons avec le *patching algorithm* (étape 2 de l'algorithme 4.3) est systématiquement une tournée optimale pour le problème.

Ces instances ne sont pas appropriées pour tester l'efficacité des algorithmes par énumération ordonnée, néanmoins, elles montrent l'utilité de la recherche d'une bonne tournée initiale. Nous supposons à la lumière de ces résultats que la faible variation des coûts de ces instances est à l'origine des temps de calcul très réduits. Nous aborderons cet aspect avec les instances **ft** qui sont à forte variation (section 4.4.2.c).

Nous remarquons aussi que les instances faciles sont résolues de manière plus efficace avec les variantes les plus simples.

Variantes	**rbg323**	**rbg358**	**rbg403**	**rbg443**
1. $\not{\mathbf{E}}\mathbf{ST_t}\not{\mathbf{R}}\not{\hat{\mathbf{B}}}\not{\tilde{\mathbf{B}}}$	0.01	0.02	0.04	0.05
2. $\not{\mathbf{E}}\mathbf{ST_t}\not{\mathbf{R}}\hat{\mathbf{B}}\tilde{\mathbf{B}}$	0.02	0.01	0.04	0.05
3. $\not{\mathbf{E}}\mathbf{ST_t}\mathbf{R}\not{\hat{\mathbf{B}}}\tilde{\mathbf{B}}$	0.02	0.01	0.03	0.06
4. $\not{\mathbf{E}}\mathbf{ST_t}\mathbf{R}\hat{\mathbf{B}}\not{\tilde{\mathbf{B}}}$	0.01	0.02	0.04	0.05
5. $\not{\mathbf{E}}\mathbf{ST_t}\mathbf{R}\hat{\mathbf{B}}\tilde{\mathbf{B}}$	0.01	0.02	0.04	0.05
6. $\not{\mathbf{E}}\mathbf{ST_l}\not{\mathbf{R}}\not{\hat{\mathbf{B}}}\tilde{\mathbf{B}}$	0.01	0.02	0.03	0.06
7. $\not{\mathbf{E}}\mathbf{ST_l}\not{\mathbf{R}}\hat{\mathbf{B}}\tilde{\mathbf{B}}$	0.01	0.01	0.04	0.06
8. $\not{\mathbf{E}}\mathbf{ST_l}\mathbf{R}\not{\hat{\mathbf{B}}}\not{\tilde{\mathbf{B}}}$	0.02	0.01	0.04	0.05
9. $\not{\mathbf{E}}\mathbf{ST_l}\mathbf{R}\hat{\mathbf{B}}\tilde{\mathbf{B}}$	0.02	0.01	0.04	0.05
10. $\mathbf{E_f ST_t}\mathbf{R}\not{\hat{\mathbf{B}}}\tilde{\mathbf{B}}$	0.02	0.01	0.04	0.05
11. $\mathbf{E_f ST_l}\not{\mathbf{R}}\not{\hat{\mathbf{B}}}\not{\tilde{\mathbf{B}}}$ [a]	0.01	0.01	0.05	0.07

a. version de Carpaneto et al. (1995)

TABLE 4.4 – Temps CPU pour les instances **rbg** (secondes)

Une dernière remarque concerne le nombre de solutions énumérées et la taille des arborescences de recherche qui sont extrêmement faibles et quasiment les mêmes pour toutes les variantes, y compris les variantes dominées (1 solution énumérée, et entre 1 et 7 nœuds dans l'arborescence pour toutes les instances).

4.4.2.c Résultats expérimentaux pour les instances ft

Les 2 instances **ft** ont été introduites par Fischetti et Toth (1992) pour un problème d'ordonnancement de tâches d'un procédé industriel de coloration de résine. La partie triangulaire supérieure de la matrice des coûts de ces instances est corrélée négativement avec la partie inférieure (voir $cor_{c_{ij},c_{ji}}$ table 4.1). Les auteurs ne donnent pas de précisions sur la problématique industrielle étudiée pour comprendre les raisons de la présence de cette forme de conflit entre les coûts c_{ij} et c_{ji}.

Ces instances sont donc asymétriques et l'écart-type relatif ainsi que l'étendue des intervalles auxquels appartiennent les coûts de ces instances indiquent une variation très élevée des coûts de ces instances.

Nous avons identifié cinq variantes non-dominées. Nous présentons à la fois les temps de calcul, le nombre de solutions énumérées et la taille des arborescences des variantes non-dominées et de la variante de Carpaneto et al. (1995) dans la table 4.5.

Précisons que les résultats expérimentaux pour l'instance **ft53** n'ont pas pu être tous relevés pour toutes les variantes. Le calcul a été stoppé pour 80 variantes à cause de la saturation de la mémoire vive (16 GB). Ces cas de saturation de la RAM sont indiqués dans la table 4.5 par le signe _.

Variantes	Temps CPU (s.)		Solutions énumérées		Taille arborescence	
	ft53	**ft70**	**ft53**	**ft70**	**ft53**	**ft70**
1. $\not{\mathbf{EST_c}}\mathbf{R\hat{B}\tilde{B}}$	50.33	0.25	709 692	1 292	1 787 675	3 251
2. $\mathbf{E_l ST_c R\hat{B}\tilde{B}}$	55.15	0.19	719 847	1 482	1 811 624	3 721
3. $\mathbf{E_n ST_c R\hat{B}\tilde{B}}$	59.84	0.18	690 404	1 247	1 741 468	3 160
4. $\not{\mathbf{EST_c}}\not{\mathbf{R}}\not{\hat{\mathbf{B}}}\tilde{\mathbf{B}}$	124.9	0.16	1 603 457	1 347	6 130 536	6 300
5. $\not{\mathbf{EST_t}}\mathbf{R\hat{B}\tilde{B}}$	165.56	0.13	2 684 464	1 445	6 362 806	3 786
6. $\mathbf{E_f ST_l}\not{\mathbf{R}}\not{\hat{\mathbf{B}}}\not{\tilde{\mathbf{B}}}$ [a]	_	0.23	_	1 856	_	7 930

a. version de Carpaneto et al. (1995)

TABLE 4.5 – Résultats pour les instances **ft**

Bien qu'elle soit plus petite, il apparaît que l'instance **ft53** soit plus difficile à résoudre que l'instance **ft70**. L'instance **ft53** se distingue de l'instance **ft70** par une variation des coûts très importante (voir la table 4.1 : $\frac{\sigma}{E}$=0.76 pour **ft53** et $\frac{\sigma}{E}$=0.41 pour **ft70**). Cette remarque rejoint nos remarques sur les instances **rbg**, à savoir que la forte variation des coûts constitue une caractéristique commune aux instances difficiles, et qu'à l'inverse l'énumération ordonnée est plus efficace lorsque les coefficients de la matrice des coûts sont faiblement dispersés.

Les variantes 4. $\not{\mathbf{EST_c}}\not{\mathbf{R}}\not{\hat{\mathbf{B}}}\tilde{\mathbf{B}}$ et 5. $\not{\mathbf{EST_t}}\mathbf{R\hat{B}\tilde{B}}$ présentent de mauvais résultats pour l'instance **ft53**, de plus le gain qu'elles permettent de réaliser sur l'instance **ft70** par rapport aux autres variantes est très faible. Les versions 4. $\not{\mathbf{EST_c}}\not{\mathbf{R}}\not{\hat{\mathbf{B}}}\tilde{\mathbf{B}}$ et 5. $\not{\mathbf{EST_t}}\mathbf{R\hat{B}\tilde{B}}$ sont donc de mauvaises variantes non-dominées.

Le test de la borne $\tilde{B}$ est nécessaire pour résoudre efficacement l'instance **ft53**. En effet, nous avons constaté grâce aux résultats complets que très peu de variantes parviennent à résoudre **ft53** sans recourir au test de la borne $\tilde{B}$, et ce avec un temps CPU qui dépasse les 300 secondes.

Les résultats obtenus montrent que les tests de la borne $\hat{B}$ et la séparation réordonnée permettent d'améliorer les performances de notre procédure, comme le montre la comparaison des résultats relatifs aux deux versions 1. $\not{\mathbf{EST_c}}\mathbf{R\hat{B}\tilde{B}}$ et 4. $\not{\mathbf{EST_c}}\not{\mathbf{R}}\not{\hat{\mathbf{B}}}\tilde{\mathbf{B}}$. En effet, par l'introduction de $\mathbf{R}$ et $\hat{\mathbf{B}}$ à la version 4. $\not{\mathbf{EST_c}}\not{\mathbf{R}}\not{\hat{\mathbf{B}}}\tilde{\mathbf{B}}$ nous observons une nette amélioration du temps CPU, du nombre de solutions énumérées et de la taille des arborescences de recherche.

La recherche d'une solution équivalente par énumération des solutions équivalentes est retenue pour deux variantes non-dominées. Ceci affirme l'utilité de ce mode de recherche de solutions équivalentes pour les instances difficiles.

Les variantes non-dominées les plus performantes contiennent donc toutes nos améliorations et se basent toutes sur la tournée de coût minimal pour la séparation.

Comme pour les instances précédemment testées, la variante de Carpaneto et al. (1995) est dominée et ne retourne pas de solution optimale pour **ft53**.

Pour ce type d'instance, il semble que les versions 1. $\not{\textbf{E}}\textbf{ST}_\textbf{c}\textbf{R}\hat{\not{\textbf{B}}}\tilde{\not{\textbf{B}}}$, 2. $\textbf{E}_\textbf{l}\textbf{ST}_\textbf{c}\textbf{R}\hat{\not{\textbf{B}}}\tilde{\not{\textbf{B}}}$ ou 3. $\textbf{E}_\textbf{n}\textbf{ST}_\textbf{c}\textbf{R}\hat{\not{\textbf{B}}}\tilde{\not{\textbf{B}}}$ sont équivalentes, puisqu'elles présentent des performances voisines pour les instances **ft**.

4.4.2.d Résultats expérimentaux pour les instances ry48p et kro124p

Les instances **ry48p** et **kro124p** se rapportent à un problème de voyageur de commerce symétrique. Ces instances sont rendues asymétriques par de légères perturbations aléatoires de la matrice des coûts (voir Fischetti et Toth (1997)). Les sommets du graphe complet décrivent ici des points sur le plan. Un coût c_{ij} est la distance euclidienne entre le point i et le point j. Des quantités ont ensuite été ajoutées ou retranchées des coefficients de la matrice des coûts. Le coefficient de corrélation $cor_{c_{ij},c_{ji}}$ dans la table 4.1 est en effet très proche de 1 et la moyenne des écarts entre les coefficients de la partie supérieure et inférieure de la matrice des coûts prouvent bien la quasi-symétrie de ces deux instances. Par ailleurs, les coûts des instances **ry48p** et **kro124p** appartiennent à des intervalles larges par rapport aux coûts des instances **ftv**, **rbg** et **ft**.

Deux variantes non-dominées (1. $\not{\textbf{E}}\textbf{ST}_\textbf{c}\textbf{R}\hat{\not{\textbf{B}}}\tilde{\not{\textbf{B}}}$ et 2. $\not{\textbf{E}}\textbf{ST}_\textbf{t}\textbf{R}\hat{\not{\textbf{B}}}\tilde{\not{\textbf{B}}}$) ont été identifiées. Les valeurs du temps de calcul, du nombre de solutions énumérées et la taille des arborescences de ces variantes non-dominées et de la variante dominée de Carpaneto et al. (1995) 3. $\textbf{E}_\textbf{f}\textbf{ST}_\textbf{l}\textbf{R}\hat{\not{\textbf{B}}}\tilde{\not{\textbf{B}}}$ sont présentées dans la table 4.6.

Les cas où les résultats n'ont pas pu être relevés pour cause de saturation de la mémoire vive sont indiqués dans la table 4.6 par le signe _.

Variantes	Temps CPU (secondes)		Solutions énumérées		Taille arborescence	
	ry48p	**kro124p**	**ry48p**	**kro124p**	**ry48p**	**kro124p**
1. $\not{\textbf{E}}\textbf{ST}_\textbf{c}\textbf{R}\hat{\not{\textbf{B}}}\tilde{\not{\textbf{B}}}$	18.37	167.29	572 906	1 741 122	1 196 951	3 227 608
2. $\not{\textbf{E}}\textbf{ST}_\textbf{t}\textbf{R}\hat{\not{\textbf{B}}}\tilde{\not{\textbf{B}}}$	13.38	_	396 538	_	1 094 315	_
3. $\textbf{E}_\textbf{f}\textbf{ST}_\textbf{l}\textbf{R}\hat{\not{\textbf{B}}}\tilde{\not{\textbf{B}}}$ [a]	30.05	183.84	728 506	1 482 024	1 759 429	3 840 339

a. version de Carpaneto et al. (1995)

TABLE 4.6 – Résultats pour les instances **ry48p** et **kro124p**

Ces instances sont difficiles. En effet, un très grand nombre de variantes ne résolvent pas **kro124p**, c'est le cas de la variante non-dominée 2. $\not{\textbf{E}}\textbf{ST}_\textbf{t}\textbf{R}\hat{\not{\textbf{B}}}\tilde{\not{\textbf{B}}}$. La variante de Carpaneto et al. (1995) est dominée, cependant elle retourne des solutions optimales, ce qui est suffisamment remarquable pour ces instances difficiles.

Dans la table complète des résultats que nous ne montrons pas ici, nous avons remarqué que toutes les variantes qui utilisent la séparation réordonnée (**R**) ne parviennent pas à résoudre l'instance **kro124p**. Nous avons aussi constaté dans la même table que

les variantes proche de la version 1. $\acute{\textbf{EST}}_{\textbf{c}}\textbf{R}\hat{\textbf{B}}\hat{\tilde{\textbf{B}}}$ qui n'utilisent pas (**R**) (c'est-à-dire les variantes qui utilisent la séparation réordonnée et la majorité des paramètres de la version 1. $\acute{\textbf{EST}}_{\textbf{c}}\textbf{R}\hat{\textbf{B}}\hat{\tilde{\textbf{B}}}$) ont des performances équivalentes à celles de la variante 1. $\acute{\textbf{EST}}_{\textbf{c}}\textbf{R}\hat{\textbf{B}}\hat{\tilde{\textbf{B}}}$. Cela pour dire que ce n'est pas l'utilisation ou la mise à l'écart des bornes $\hat{\textbf{B}}$ ou $\tilde{\textbf{B}}$ qui influe sur les performances de la variante 1. $\acute{\textbf{EST}}_{\textbf{c}}\textbf{R}\hat{\textbf{B}}\hat{\tilde{\textbf{B}}}$, mais l'utilisation de la séparation réordonnée (**R**). Il est donc remarquable que contrairement aux autres familles d'instances, la séparation réordonnée (**R**) n'améliore pas les performances de notre procédure pour les instances **ry48p** et **kro124p**.

Nous avons observé que les solutions optimales des nœuds de l'arborescence pour les instances symétriques contiennent un nombre important de sous-tournées de petites tailles, notamment des sous-tournées composées de deux arcs. Nous avons fait les mêmes constatations pour les instances quasi-symétriques comme **kro124p** et **ry48p**. Nous présumons que si une arête (i, j) appartient à la solution optimale d'une instance symétrique d'un problème d'affectation, alors l'arête (j, i) possède de fortes chances aussi de faire partie de la solution optimale ($c_{ij} = c_{ji}$). Nous pensons donc que la multiplicité des sous-tournées de petites tailles est due à cette conjecture. Nous pensons aussi que la symétrie est à l'origine de la difficulté de l'instance **kro124p** et qu'en règle générale, les algorithmes par énumération ordonnée ne sont pas adaptés pour la résolution du problème de voyageur de commerce symétrique.

4.4.2.e Cas de l'instance p43

L'instance **p43** provient d'un problème industriel d'ordonnancement en ingénierie chimique. C'est une instance asymétrique qui possède néanmoins une partie symétrique, c'est-à-dire qu'en supprimant certains sommets du graphe de cette instance on obtient une instance symétrique. Les valeurs observées dans la table 4.1, confirment l'asymétrie de cette instance par le coefficient de corrélation $cor_{c_{ij},c_{ji}}$ proche de 1 et la moyenne des écarts $E(c_{ij}, c_{ji})$ très élevée, et nous informe sur la forte dispersion des coûts grâce au coefficient de variation $\frac{\sigma}{E} = 2.57$ et à l'étendue de l'intervalle des valeurs Δ.

Nous ne présentons aucun résultat numérique pour l'instance **p43** car nous n'avons pas réussi à la résoudre, bien que sa taille soit raisonnable ($n = 40$). Cette instance cumule les deux caractéristiques d'une instance difficile, à savoir, sa partie symétrique et une forte variation des coûts.

En effet, nous avons observé que l'instance **ft53**, plus petite que l'instance **ft70**, s'est avérée plus difficile à résoudre. Nous pensons que cela est dû à une variation des coûts plus élevée pour **ft53** ($\frac{\sigma}{E}$ table 4.1). Par ailleurs l'instance **kro124p** est difficile, bien que sa variation soit du même ordre de grandeur que celle des instances faciles **ftv**. Nous pensons que c'est son origine symétrique qui pénalise cette instance.

4.4.2.f Résultats pour toutes les familles

Nous présentons dans la table 4.7 les temps CPU des variantes non-dominées obtenues en prenant en considération 7 instances choisies dans toutes les familles des instances de TSPLIB. Pour ces tests, nous avons repris les résultats des deux plus grandes instances de la famille **ftv** (**ftv70** et **ftv170**), de l'instance **rbg443** et des instances **ft53**, **ft70**, **ry48p** et **kro124p**. En effet, nous avons écarté les instances non discriminantes qui sont soit trop faciles comme certaines instances des familles **ftv** ou **rbg**, soit trop difficiles comme c'est le cas de l'instance **p43** qui n'est résolue par aucune des variantes.

La variante dominée de Carpaneto et al. (1995) est présentée à la ligne (32.) de la table 4.7. Le signe _ dans la table 4.7 indique que la valeur du temps CPU pour une variante n'est pas relevé à cause de la saturation de la mémoire vive. Les variantes sont classées dans la table 4.7 dans un ordre décroissant du nombre d'instances résolues, puis dans un ordre croissant de la somme des temps CPU. Le fait que l'instance **kro124p** soit plus difficile à résoudre que **ft53** privilégie les variantes qui résolvent plus efficacement l'instance **kro124p**.

La prise en compte de toutes les familles des instances fait émerger un nouveau critère de performance pour les variantes testées qui consiste à préférer les variantes qui retournent une solution optimale pour toutes les instances. Ici on compte 4 variantes non-dominées dans ce cas (les quatre premières lignes de la table 4.7). Les variantes qui résolvent à l'optimum toutes les instances testées utilisent toutes la séparation selon la sous-tournée de coût minimum ($\mathbf{ST_c}$) et le test de borne $\tilde{\mathbf{B}}$ et n'utilisent pas l'amélioration $\mathbf{R}$. La séparation réordonnée $\mathbf{R}$ est rejetée certainement à cause de l'instance **kro124p** (voir la section 4.4.2.d).

Les variantes qui utilisent le test de la borne $\hat{\mathbf{B}}$ sont au nombre de 22 parmi les 31 variantes non-dominées (i.e. 70 %). La même proportion de variantes non-dominées (70 %) utilisent la borne $\tilde{\mathbf{B}}$. Presque la totalité des variantes non-dominées (i.e. 30 parmi 31) utilisent une des règles d'élagage $\hat{\mathbf{B}}$ ou $\tilde{\mathbf{B}}$ au moins, ce qui justifie globalement l'intérêt de ces règles d'élagage. Par ailleurs, près de la moitié des variantes non-dominées (14 parmi 31) utilisent les tests de bornes $\hat{\mathbf{B}}$ et $\tilde{\mathbf{B}}$ simultanément, et cela montre leur complémentarité.

4.4.3 Synthèse des résultats expérimentaux

Grâce aux résultats que nous avons obtenus par la résolution des instances de la TSPLIB, nous pouvons expliquer quelques-uns des comportements de nos algorithmes.

Avant d'aborder l'analyse des résultats qui se rapportent aux variantes de l'algorithme, nos premières conclusions concernent les instances. Il semble que la forte variation des coûts a un impact négatif sur les performances des algorithmes par énumération ordonnée. De même, la symétrie des instances entrave le bon déroulement de nos algorithmes. La combinaison de ces deux caractéristiques (forte variation et symétrie) donne lieu aux instances les plus difficiles à résoudre, ce qui est le cas de l'instance **p43**.

Variantes	**ftv70**	**ftv170**	**rbg443**	**ft53**	**ft70**	**ry48p**	**kro124p**
1. $\not{\mathbf{E}}\mathbf{ST_c}\not{\mathbf{R}}\hat{\mathbf{B}}\tilde{\mathbf{B}}$	0.95	79.65	0.06	105.07	0.19	18.79	168.13
2. $\mathbf{E_f ST_c}\not{\mathbf{R}}\hat{\mathbf{B}}\tilde{\mathbf{B}}$	0.31	31.57	0.07	131.76	0.19	24.17	219.78
3. $\not{\mathbf{E}}\mathbf{ST_c}\not{\mathbf{R}}\not{\hat{\mathbf{B}}}\tilde{\mathbf{B}}$	1.09	86.94	0.06	124.90	0.16	22.86	194.86
4. $\mathbf{E_f ST_c}\not{\mathbf{R}}\not{\hat{\mathbf{B}}}\tilde{\mathbf{B}}$	0.37	35.02	0.06	152.76	0.22	25.25	226.41
5. $\mathbf{E_f ST_c R}\hat{\mathbf{B}}\tilde{\mathbf{B}}$	0.28	35.32	0.06	64.21	0.20	28.76	_
6. $\mathbf{E_f ST_c}\not{\mathbf{R}}\hat{\mathbf{B}}\tilde{\mathbf{B}}$	0.37	39.50	0.06	61.05	0.20	31.56	_
7. $\not{\mathbf{E}}\mathbf{ST_c R}\hat{\mathbf{B}}\tilde{\mathbf{B}}$	1.22	81.81	0.06	50.33	0.25	22.07	_
8. $\mathbf{E_f ST_t}\not{\mathbf{R}}\hat{\mathbf{B}}\tilde{\mathbf{B}}$	0.22	56.81	0.06	159.84	0.19	17.33	_
9. $\mathbf{E_f ST_t}\not{\mathbf{R}}\not{\hat{\mathbf{B}}}\tilde{\mathbf{B}}$	0.23	55.58	0.07	184.46	0.25	18.39	_
10. $\not{\mathbf{E}}\mathbf{ST_c}\not{\mathbf{R}}\hat{\mathbf{B}}\not{\tilde{\mathbf{B}}}$	0.92	81.13	0.06	_	0.21	18.37	167.29
11. $\mathbf{E_f ST_l}\not{\mathbf{R}}\not{\hat{\mathbf{B}}}\not{\tilde{\mathbf{B}}}$	0.30	60.99	0.07	_	0.23	30.05	183.84
12. $\mathbf{E_f ST_c}\not{\mathbf{R}}\hat{\mathbf{B}}\not{\tilde{\mathbf{B}}}$	0.31	33.52	0.06	_	0.27	23.21	220.22
13. $\not{\mathbf{E}}\mathbf{ST_t}\not{\mathbf{R}}\hat{\mathbf{B}}\tilde{\mathbf{B}}$	0.70	126.51	0.05	135.56	0.18	15.00	_
14. $\mathbf{E_f ST_l}\not{\mathbf{R}}\hat{\mathbf{B}}\not{\tilde{\mathbf{B}}}$	0.30	56.05	0.10	_	0.19	28.21	194.38
15. $\mathbf{E_f ST_t R}\hat{\mathbf{B}}\tilde{\mathbf{B}}$	0.20	47.96	0.06	218.95	0.20	12.78	_
16. $\mathbf{E_f ST_l}\not{\mathbf{R}}\hat{\mathbf{B}}\tilde{\mathbf{B}}$	0.30	55.42	0.06	_	0.17	29.13	196.23
17. $\not{\mathbf{E}}\mathbf{ST_t R}\hat{\mathbf{B}}\tilde{\mathbf{B}}$	0.75	106.77	0.05	165.56	0.13	13.50	_
18. $\mathbf{E_f ST_l}\not{\mathbf{R}}\not{\hat{\mathbf{B}}}\tilde{\mathbf{B}}$	0.29	57.23	0.07	_	0.21	30.24	203.62
19. $\not{\mathbf{E}}\mathbf{ST_t}\not{\mathbf{R}}\not{\hat{\mathbf{B}}}\tilde{\mathbf{B}}$	0.65	117.41	0.06	166.51	0.26	16.24	_
20. $\not{\mathbf{E}}\mathbf{ST_t}\not{\mathbf{R}}\not{\hat{\mathbf{B}}}\tilde{\mathbf{B}}$	0.82	105.97	0.06	181.44	0.15	13.49	_
21. $\mathbf{E_f ST_t}\not{\mathbf{R}}\not{\hat{\mathbf{B}}}\tilde{\mathbf{B}}$	0.22	49.21	0.05	241.90	0.20	14.69	_
22. $\not{\mathbf{E}}\mathbf{ST_l}\not{\mathbf{R}}\hat{\mathbf{B}}\not{\tilde{\mathbf{B}}}$	0.85	109.63	0.06	_	0.14	37.79	231.58
23. $\not{\mathbf{E}}\mathbf{ST_l}\not{\mathbf{R}}\hat{\mathbf{B}}\tilde{\mathbf{B}}$	0.84	111.76	0.06	_	0.13	40.34	246.89
24. $\not{\mathbf{E}}\mathbf{ST_t R}\hat{\mathbf{B}}\not{\tilde{\mathbf{B}}}$	0.75	104.80	0.05	321.15	0.20	11.55	_
25. $\mathbf{E_f ST_c R}\hat{\mathbf{B}}\not{\tilde{\mathbf{B}}}$	0.29	36.79	0.06	_	0.49	28.34	_
26. $\mathbf{E_f ST_l R}\hat{\mathbf{B}}\not{\tilde{\mathbf{B}}}$	0.25	54.09	0.06	_	0.18	26.60	_
27. $\mathbf{E_f ST_l R}\hat{\mathbf{B}}\tilde{\mathbf{B}}$	0.25	53.83	0.08	_	0.16	27.21	_
28. $\mathbf{E_l ST_c R}\hat{\mathbf{B}}\tilde{\mathbf{B}}$	19.69	_	0.42	55.15	0.19	26.93	_
29. $\mathbf{E_n ST_c R}\hat{\mathbf{B}}\tilde{\mathbf{B}}$	1.82	_	0.48	59.84	0.18	41.32	_
30. $\not{\mathbf{E}}\mathbf{ST_l R}\hat{\mathbf{B}}\not{\tilde{\mathbf{B}}}$	1.02	99.95	0.05	_	0.14	22.59	_
31. $\not{\mathbf{E}}\mathbf{ST_l R}\hat{\mathbf{B}}\tilde{\mathbf{B}}$	0.92	102.02	0.05	_	0.13	23.01	_
32. $\mathbf{E_f ST_l}\not{\mathbf{R}}\not{\hat{\mathbf{B}}}\not{\tilde{\mathbf{B}}}$ [a]	0.30	60.99	0.07	_	0.23	30.05	183.84
Min	0.2	31.57	0.05	50.33	0.13	11.55	167.29
Max	19.69	_	0.48	_	0.49	41.32	_
Moyenne	1.21	_	0.09	_	0.20	23.86	_

a. version de Carpaneto et al. (1995)

TABLE 4.7 – Temps CPU des variantes non-dominées pour toutes les familles d'instances (secondes)

Nous avons souvent remarqué que les instances faciles sont résolues de manière plus efficace avec les variantes simples. Aussi nos améliorations n'ont pas été d'une grande utilité pour résoudre les petites instances de **ftv** et les instances **rbg**.

Les avantages des tests des bornes $\hat{\mathbf{B}}$ et $\tilde{\mathbf{B}}$ sont certains. Les règles d'élagage améliorent les performances de l'énumération ordonnée pour tous les types d'instances. Cette remarque est confirmée par les résultats de la table 4.7 où la quasi-totalité des variantes non-dominées utilisent les deux tests des bornes $\hat{\mathbf{B}}$ et $\tilde{\mathbf{B}}$. Le choix de la sous-tournée de coût minimum pour la séparation est aussi le critère qui a le plus été retenu pour les meilleures variantes. Cela renforce l'intuition de combiner une séparation étroite à la division des sous-tournées fortement liées.

Les autres améliorations dépendent des instances résolues. Lorsque les instances sont asymétriques, il est conseillé d'utiliser la séparation réordonnée ($\mathbf{R}$), alors que c'est contre-indiqué pour résoudre les instances quasi-symétriques (voir les instances **ft53** et **kro124p**).

L'étape de la recherche d'une solution équivalente ne doit pas prendre beaucoup de temps, notamment lorsque les instances ne sont pas très difficiles. En effet, une forte majorité des variantes non-dominées utilisent soit la fusion de sous-tournées ($\mathbf{E_f}$), ou bien éludent complètement l'étape de recherche d'une solution équivalente ($\not{\mathbf{E}}$). L'énumération de solutions équivalentes ($\mathbf{E_l}$ et $\mathbf{E_n}$) a été retenue uniquement pour l'instance difficile **ft53**.

Aucune variante ne s'est détachée des autres pour incarner la meilleure version pour résoudre le problème du voyageur de commerce asymétrique par l'énumération ordonnée. Il semble préférable d'analyser l'instance, pour la classer dans une des familles décrites précédemment, et choisir une variante adéquate pour la résoudre.

Pour finir, nous avons comparé nos résultats avec ceux présentés par Roberti et Toth (2012) qui se rapportent à d'autres approches essentiellement basées sur le *Branch and cut*. N'ayant pas pu récupérer les codes sources de ces algorithmes, il est difficiles de tirer des conclusions. Néanmoins, nos algorithmes semblent être très compétitifs.

4.5 Conclusion

Nous avons proposé un algorithme par énumération ordonnée pour résoudre le problème du voyageur de commerce asymétrique. Nous nous sommes basés sur l'algorithme de Carpaneto et al. (1995) auquel nous avons apporté quelques améliorations. Nos améliorations se présentent comme des alternatives aux choix algorithmiques de Carpaneto et al. (1995), c'est le cas de l'étape de la recherche d'une solution équivalente ou de celle du choix de la sous-tournée de séparation. Nos améliorations prennent aussi la forme d'ajouts ou de nouveaux mécanismes introduits dans l'algorithme de base, comme la séparation réordonnée et les bornes $\hat{B}$ et $\tilde{B}$.

L'analyse des résultats expérimentaux que nous avons obtenus pour les instances de la TSPLIB, qui peut être transposée sur d'autres instances, nous a permis de mieux

cerner l'impact positif ou négatif des différentes améliorations en fonction de l'instance à résoudre.

Conclusion

Nous avons proposé dans cette thèse des outils pour la résolution de problèmes d'optimisation combinatoire difficiles. Les algorithmes que nous avons proposés sont basés sur le principe de l'énumération ordonnée des solutions d'un problème relâché, qui lui aussi est un problème d'optimisation combinatoire. Nous avons centré nos efforts sur l'amélioration des performances de ces algorithmes par l'introduction de divers perfectionnements. Nous avons formalisé une méthode générique qui ne dépend pas du problème résolu, puis nous avons instancié nos procédures sur deux problèmes d'optimisation combinatoire, à savoir, la recherche d'une solution de compromis pour le problème d'affectation multiobjectif et le problème du voyageur de commerce asymétrique.

Les perfectionnements que nous avons proposés, qui peuvent être génériques ou spécifiques à un problème particulier, se présentent sous la forme de tests de bornes et de choix algorithmiques. Ils visent à améliorer nos procédures en diminuant le nombre de solutions énumérées ou en agissant sur la complexité de l'énumération de chacune de ces solutions.

Les résultats expérimentaux que nous avons obtenus pour la recherche d'une solution de compromis pour le problème d'affectation multiobjectif montrent dans un premier temps que l'énumération ordonnée, dénuée des améliorations que nous proposons, est dominée par des méthodes classiques (solver CPLEX) pour la résolution des programmes linéaires mixtes en nombres entiers. Suite à l'introduction des tests de bornes et la réorganisation de la séparation, nous avons observé que les nouvelles versions sont significativement plus performantes que les solvers, et permettent de résoudre des instances de plus en plus grandes. Une seconde version (version avec minoration variable) repousse les limites de notre procédure et nous permet de résoudre des instances avec un grand nombre d'objectifs.

Nous avons aussi présenté des résultats pour le problème du voyageur de commerce asymétrique. Contrairement au problème précédent, des algorithmes par énumération ordonnée et des instances issues de problèmes réels sont disponibles. Nous avons donc testé nos procédures sur ces instances et comparé nos résultats à ceux des algorithmes de la littérature ce qui a montré l'efficacité de notre méthode. Les résultats que nous avons obtenus pour le problème du voyageur de commerce asymétrique sont différents de ceux que nous avons pu constater pour la recherche d'une solution de bon compro-

mis où tous les perfectionnements améliorent la procédure. En effet, nous avons constaté qu'il est préférable d'analyser la nature de l'instance du problème du voyageur de commerce asymétrique avant de la résoudre, et ce afin de choisir la bonne combinaison des améliorations qui engendrera la meilleure variante pour résoudre cette instance.

La principale force des méthodes par énumération ordonnée réside dans la simplicité du principe général sur lequel elles se fondent. Cette simplicité permet de modifier aisément ces méthodes pour les améliorer ou pour les adapter pour résoudre d'autres problèmes. Par ailleurs, l'ajout de perfectionnements -tout aussi intelligibles que le principe général- nous a permis d'obtenir des algorithmes efficaces pour des problèmes difficiles.

Dans la continuité de ces travaux nous pourrions adapter l'approche par énumération ordonnée à la recherche de solution de bon compromis pour le problème du voyageur de commerce asymétrique multiobjectif. Pour ce problème la relaxation est sur deux niveaux, puisque la fonction objectif est non-linéaire et l'ensemble réalisable est plus restreint que l'ensemble admissible du problème d'affectation. Nous pensons qu'il serait intéressant de comparer deux approches par énumération ordonnée ; une qui énumère des affectations et une autre qui énumère des tournées.

Par ailleurs, les algorithmes de recherche de solution de compromis peuvent être intégrés comme des outils élémentaires dans des générateurs de frontière de Pareto pour les problèmes d'optimisation combinatoire multiobjectif. Une approche spécifique aux problèmes bi-objectifs, qui est fondée sur l'exploration récursive de zones de recherche par la recherche de solutions de compromis, a été proposée par Ralphs et al. (2006). La généralisation de cette approche est une extension intéressante de notre travail, notamment en combinant la recherche de solutions de compromis et le principe de l'énumération ordonnée.

Le problème d'affectation quadratique est connu pour être très difficile et c'est pour cela qu'il semble être une bonne application pour l'énumération ordonnée. Nous avons adapté notre algorithme pour ce problème et nous avons opté pour la minoration de la fonction objectif à l'aide de la borne de Gilmore-Lawler (voir Gilmore (1962)). Cette borne est appréciée car elle est linéaire, mais aussi parce que sa validité n'est pas soumise à la condition d'ajouter des contraintes dans le problème d'affectation. Nous avons mis en œuvre plusieurs variantes de notre procédure et les résultats expérimentaux obtenus ont montré que la qualité de l'approximation que fournit la borne de Gilmore-Lawler n'est pas suffisante pour engendrer un processus d'énumération efficace, et ce même si la fonction minorante est ajustée durant la progression du Branch and Bound comme c'est le cas dans les versions avec minoration variable. Il serait intéressant d'explorer d'autres minorations, même si elles nécessitent l'introduction de contraintes supplémentaires ce qui implique que le problème d'affectation linéaire soit une relaxation sur deux niveaux (minoration de la fonction quadratique par une fonction linéaire et suppression de ces contraintes supplémentaires). Il est aussi nécessaire dans ce cas de réfléchir à des règles d'élagage propres au problème d'affectation quadratique.

Bibliographie

R. Ahuja, T. Magnanti et J. Orlin. 1993, *Network Flows : Theory, Algorithms and Applications*, Prentice Hall, New Jersey.

H. Aissi, C. Bazgan et D. Vanderpooten. 2009, «Min–max and min–max regret versions of combinatorial optimization problems : A survey», *European Journal of Operational Research*, vol. 197, n° 2, p. 427 – 438.

N. Ascheuer. 1986, «Hamiltonian path problems in the on-line optimization of flexible manufacturing systems», cahier de recherche TR 96-3, ZIB.

L. Belhoul, L. Galand et D. Vanderpooten. 2014, «An efficient procedure for finding best compromise solutions to the multi-objective assignment problem», *Computers & Operations Research*, vol. 49, p. 97–106.

M. Bellmore et J. Malone. 1971, «Pathology of traveling-salesman subtour-elimination algorithms», *Operations Research*, vol. 19, n° 2, p. 278–307.

V. Bowman Jr. 1976, «On the relationship of the Tchebycheff norm and efficient frontier of multiple-criteria objectives», dans *Multiple Criteria Decision Making*, édité par H. Thiriez et S. Zionts, LNEMS 130, Springer-Verlag, Berlin, p. 76–85.

R. Burkard, M. Dell'Amico et S. Martello. 2009, *Assignment Problems*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

G. Carpaneto, M. Dell'Amico et P. Toth. 1995, «Exact solution of large-scale, asymmetric traveling salesman problems», *ACM Trans. Math. Softw.*, vol. 21, n° 4, p. 394–409.

G. Carpaneto et P. Toth. 1980, «Some new branching and bounding criteria for the asymmetric travelling salesman problem», *Management Science*, vol. 26, n° 7, p. 736–743.

C. Chegireddy et H. Hamacher. 1987, «Algorithms for finding k-best perfect matchings», *Discrete Applied Mathematics*, vol. 18, n° 2, p. 155–165.

K. Dächert, J. Gorski et K. Klamroth. 2012, «An augmented weighted tchebycheff method with adaptively chosen parameters for discrete bicriteria optimization problems», *Computers & Operations Research*, vol. 39, n° 12, p. 2929–2943.

E. Dijkstra. 1959, «A note on two problems in connexion with graphs», *Numerische Mathematik*, vol. 1, n° 1, p. 269–271.

Z. Drezner et S. Nof. 1984, «On optimizing bin picking and insertion plans for assembly robots», *IIE Transactions*, vol. 16, n° 3, p. 262–270.

J. Edmonds et R. Karp. 1972, «Theoretical improvements in algorithmic efficiency for network flow problems», *Journal of the ACM*, vol. 19, n° 2, p. 248–264.

M. Ehrgott. 2005, *Multicriteria Optimization (2nd ed.)*, Springer.

M. Ehrgott et A. Skriver. 2003, «Solving biobjective combinatorial max-ordering problems by ranking methods and a two-phases approach», *European Journal of Operational Research*, vol. 147, n° 3, p. 657–664.

M. Fischetti et P. Toth. 1992, «An additive bounding procedure for the asymmetric travelling salesman problem», *Mathematical Programming*, vol. 53, n° 1-3, p. 173–197.

M. Fischetti et P. Toth. 1997, «A polyhedral approach to the asymmetric traveling salesman problem», *Management Science*, vol. 43, n° 11, p. 1520–1536.

M. Fischetti, P. Toth et D. Vigo. 1994, «A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs», *Operations Research*, vol. 42, n° 5, p. 846–859.

K. Fukuda et T. Matsui. 1992, «Finding all minimum-cost perfect matchings in bipartite graphs», *Networks*, vol. 22, n° 5, p. 461–468.

K. Fukuda et T. Matsui. 1994, «Finding all the perfect matchings in bipartite graphs», *Applied Mathematics Letters*, vol. 7, n° 1, p. 15 – 18.

H. Gabow. 1983, «Scaling algorithms for network problems», dans *Foundations of Computer Science, 1983., 24th Annual Symposium on*, p. 248–258.

H. Gabow, Z. Galil, T. H. Spencer et R. Tarjan. 1986, «Efficient algorithms for finding minimum spanning trees in undirected and directed graphs», *Combinatorica*, vol. 6, n° 2, p. 109–122.

H. Gabow et R. Tarjan. 1991, «Faster scaling algorithms for general graph matching problems», *J. ACM*, vol. 38, n° 4, p. 815–853.

L. Galand et P. Perny. 2006, «Search for compromise solutions in multiobjective state space graphs», dans *17th European Conference on Artificial Intelligence*, p. 93–97.

L. Galand, P. Perny et O. Spanjaard. 2010, «Choquet-based optimisation in multiobjective shortest path and spanning tree problems», *European Journal of Operational Research*, vol. 204, n° 2, p. 303–315.

M. Garey et D. Johnson. 1979, *Computers and Intractability : A Guide to the Theory of NP-Completeness*, W. H. Freeman.

P. Gilmore. 1962, «Optimal and suboptimal algorithms for the quadratic assignment problem», *Journal of the Society for Industrial and Applied Mathematics*, vol. 10, n° 2, p. 305–313.

M. Gondran et M. Minoux. 2009, *Graphes et algorithmes (4e ed.)*, Lavoisier.

H. Hamacher. 1995, «A note on k best network flows», *Annals of Operations Research*, vol. 57, n° 1, p. 65–72.

H. Hamacher et M. Queyranne. 1985, «K best solutions to combinatorial optimization problems», *Annals of Operations Research*, vol. 4, n° 1, p. 123–143.

H. Hamacher et G. Ruhe. 1994, «On spanning tree problems with multiple objectives», *Annals of Operations Research*, vol. 52, n° 4, p. 209–230.

J. Hopcroft et R. Karp. 1973, «An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs», *SIAM Journal on Computing*, vol. 2, n° 4, p. 225–231.

R. Karp. 1979, «A patching algorithm for the nonsymmetric traveling-salesman problem», *SIAM Journal on Computing*, vol. 8, n° 4, p. 561–573.

N. Katoh, T. Ibaraki et H. Mine. 1981, «An algorithm for finding k minimum spanning trees», *SIAM Journal on Computing*, vol. 10, n° 2, p. 247–255.

J. Knowles et D. Corne. 2001, «Benchmark problem generators and results for the multiobjective degree-constrained minimum spanning tree problem», dans *Proceedings of the Genetic and Evolutionary Computation Conference*, (GECCO-2001), Morgan Kaufmann, p. 424–431.

P. Kouvelis et G. Yu. 1997, *Robust discrete optimization and its applications*, Kluwer Academic, Dordrecht.

E. Lawler. 1972, «A procedure for computing the k best solutions to discrete optimization problems and its application to the shortest path problem», *Management Science*, vol. 18, n° 7, p. 401–405.

J. Little, K. Murty, D. Sweeney et C. Karel. 1963, «An algorithm for the traveling salesman problem», *Operations Research*, vol. 11, n° 6, p. 972–989.

M. Luque, K. Miettinen, A. Ruiz et F. Ruiz. 2012, «A two-slope achievement scalarizing function for interactive multiobjective optimization», *Computers & Operations Research*, vol. 39, n° 7, p. 1673 – 1681.

K. Miettinen et M. Mäkelä. 2002, «On scalarizing functions in multiobjective optimization», *OR Spectrum*, vol. 24, n° 2, p. 193–213.

M. Miller, H. Stone et I. Cox. 1997, «Optimizing Murty's ranked assignment method», *IEEE Transactions on Aerospace and Electronic Systems*, vol. 33, n° 3, p. 851–862.

K. Murty. 1968, «An algorithm for ranking all the assignments in order of increasing cost», *Operations Research*, vol. 16, n° 3, p. 682–687.

K. Murty, C. Karel et J. Little. 1962, «Original article on branch and bound method for the traveling salesman problem», cahier de recherche, Institute of Technology Cleveland.

Y. Nikulin, K. Miettinen et M. Mäkelä. 2012, «A new achievement scalarizing function based on parameterization in multiobjective optimization», *OR Spectrum*, vol. 34, n° 1, p. 69–87.

J. Paixão, E. Martins, M. Rosa et J. Dos Santos. 2003, «The determination of the path with minimum-cost norm value», *Networks*, vol. 41, n° 4, p. 184–196.

M. Pascoal, M. Captivo et J. Clímaco. 2003, «A note on a new variant of Murty's ranking assignments algorithm», *Quarterly Journal of the Belgian, French and Italian Operations Research Societies*, vol. 1, n° 3, p. 243–255.

C. Pedersen, L. Nielsen et K. Andersen. 2008, «An algorithm for ranking assignments using reoptimization», *Computers & Operations Research*, vol. 35, n° 11, p. 3714–3726.

J. Pekny et D. Miller. 1992, «A parallel branch and bound algorithm for solving large asymmetric traveling salesman problems», *Math. Program.*, vol. 55, n° 1-6, p. 17–33.

A. Przybylski, X. Gandibleux et M. Ehrgott. 2008, «Two phase algorithms for the biobjective assignment problem», *European Journal of Operational Research*, vol. 185, n° 2, p. 509–533.

A. Przybylski, X. Gandibleux et M. Ehrgott. 2010, «A two phase method for multiobjective integer programming and its application to the assignment problem with three objectives», *Discrete Optimization*, vol. 7, n° 3, p. 149–165.

T. Ralphs, M. Saltzman et M. Wiecek. 2006, «An improved algorithm for solving biobjective integer programs», *Annals of Operations Research*, vol. 147, n° 1, p. 43–70.

R. Roberti et P. Toth. 2012, «Models and algorithms for the asymmetric traveling salesman problem : an experimental comparison», *EURO Journal on Transportation and Logistics*, vol. 1, n° 1-2, p. 113–133.

B. Roy. 1970, *Algèbre moderne et théorie des graphes orientées vers les sciences économiques et sociales. Tome 2 : Applications et problèmes spécifiques*, Dunod, Paris.

F. Ruiz, M. Luque, F. Miguel et M. del Mar Muñoz. 2008, «An additive achievement scalarizing function for multiobjective programming problems», *European Journal of Operational Research*, vol. 188, n° 3, p. 683–694.

M. Sakarovitch. 1984, *Optimisation combinatoire : Programmation discrète*, Collection Enseignement des sciences, Hermann.

N. Shor. 1985, *Minimization Methods for Non-differentiable Functions*, Springer series in computational mathematics, Springer-Verlag, New York, NY, USA.

R. Tarjan. 1972, «Depth-first search and linear graph algorithms», *SIAM Journal on Computing*, vol. 1, n° 2, p. 146–160.

R. Tarjan. 1983, *Data Structures and Network Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

J. Teghem. 2012, *Recherche opérationnelle - Tome 1, Méthode d'optimisation*, ellipses, Paris.

N. Tomizawa. 1971, «On some techniques useful for solution of transportation network problems», *Networks*, vol. 1, n° 2, p. 173–194.

T. Uno. 1997, «Algorithms for enumerating all perfect, maximum and maximal matchings in bipartite graphs», dans *Algorithms and Computation*, *Lecture Notes in Computer Science*, vol. 1350, édité par H. Leong, H. Imai et S. Jain, Springer Berlin Heidelberg, p. 92–101.

T. Uno. 2001, «A fast algorithm for enumerating bipartite perfect matchings», dans *ISAAC*, p. 367–379.

D. Vanderpooten. 1989, «The interactive approach in MCDA : a technical framework and some basic conceptions», *Mathematical and Computer Modelling*, vol. 12, n° 10-11, p. 1213–1220.

D. Vanderpooten et P. Vincke. 1989, «Description and analysis of some representative interactive multicriteria procedures», *Mathematical and Computer Modelling*, vol. 12, n° 10-11, p. 1221–1238.

A. Wierzbicki. 1986a, «A methodological approach to comparing parametric characterizations of efficient solutions», dans *Large-Scale Modelling and Interactive Decision Analysis*, *Lecture Notes in Economics and Mathematical Systems*, vol. 273, édité par G. Fandel, M. Grauer, A. Kurzhanski et A. Wierzbicki, Springer Berlin Heidelberg, p. 27–45.

A. Wierzbicki. 1986b, «On the completeness and constructiveness of parametric characterizations to vector optimization problems», *OR Spectrum*, vol. 8, n° 2, p. 73–87.

Résumé : Notre objectif dans cette thèse est de proposer des algorithmes efficaces pour résoudre des problèmes d'optimisation combinatoire difficiles. Nos algorithmes se fondent sur une démarche générique par énumération ordonnée. Ils sont mis en œuvre sous la forme de méthodes de Branch and Bound, que nous améliorons à différents niveaux de la séparation et de l'évaluation. Nous accordons une importance particulière à la validation de l'efficacité de nos algorithmes par l'expérimentation, et ce par la résolution d'instances générées aléatoirement ou qui proviennent de problèmes réels et par la confrontation de nos résultats expérimentaux à ceux des algorithmes concurrents présents dans la littérature.

Dans un premier temps, nous établissons le principe de l'énumération ordonnée qui consiste à générer dans un ordre adéquat les solutions d'un problème relâché associé au problème principal jusqu'à l'obtention de la preuve d'optimalité d'une solution réalisable. Nous construisons ensuite une procédure générique dans le cadre général des problème d'optimisation combinatoire. En maintenant le même niveau de généralité sur les problèmes résolus, nous spécifions les divers perfectionnements que nous introduisons sous la forme de règles d'élagage de l'arborescence de recherche.

Dans un second temps nous abordons les applications de notre algorithme sur des problèmes qui admettent le problème d'affectation comme relaxation. Le premier cas particulier que nous étudions est la recherche d'une solution de bon compromis pour le problème d'affectation multiobjectif. Le défi pour ce problème -et pour les problèmes similaires- est de minorer la fonction objectif non-linéaire qui modélise le compromis par une fonction linéaire et d'énumérer les affectations par ordre non-décroissant de cette fonction minorante. Nous proposons des perfectionnements pour nos algorithmes qui améliorent sensiblement leurs performances au vu des résultats expérimentaux. La seconde application se rapporte au problème du voyageur de commerce asymétrique qui présente la difficulté de comporter des contraintes qui interdisent les sous-tournées, en plus des contraintes du problème d'affectation. Ici encore, nous proposons des améliorations que nous incorporons dans le schéma de base de l'énumération ordonnée. L'étude expérimentale montre que les perfectionnements doivent être sélectionnés en prenant en considération les caractéristiques des instances résolues.

Mots-clés : Énumération ordonnée, Branch and Bound, problèmes d'optimisation combinatoire mono et multi-objectifs, problème d'affectation, problème du voyageur de commerce asymétrique.

Abstract: Our aim in this thesis is to propose efficient algorithms for solving difficult combinatorial optimization problems. Our algorithms are based on a generic method of ordered enumeration. We propose an implementation as Branch and Bound methods where several improvements are introduced at various levels. We verify the efficiency of our algorithms by solving randomly generated or real-world instances and by comparing our computational results with those obtained by competing algorithms known in the literature.

Initially, we describe the principle of ordered enumeration which consists in generating in a specific order solutions of a relaxed problem associated to the difficult main problem, until meeting a proof of the optimality of a feasible solution. We construct a generic procedure in the general context of combinatorial optimization problems. Maintaining the same level of generality, we specify various improvements that we introduce to improve our procedure.

In a second step we discuss applications of our algorithm on some difficult problems which admit the assignment problem as relaxation. The first special case we study is the search for a compromise solution to the multiobjective assignment problem. The challenge for this problem - and similar problems- is to find a linear lower bounding function for the non-linear function which defines the compromise, and enumerate feasible assignments in non decreasing order of the lower bounding function. We propose refinements to our algorithms which, in light of the experimental results, improve significantly their performances. The second application is the asymmetric travelling salesman problem, which contains sub-tour constraints in addition to the constraints of the assignment problem. Here again, we propose improvements to the basic scheme of ordered enumeration, which induces a large number of variants. The experimental study shows that the improvements should be selected taking into account the characteristics of the instances.

Key words: Ordered Enumeration, Branch and Bound, Mono and Multi-objective Combinatorial Optimization Problems, Assignment Problem, Asymmetric Travelling Salesman Problem.