

Approches hybrides pour des variantes du sac à dos et applications

Raïd Mansi

▶ **To cite this version:**

Raïd Mansi. Approches hybrides pour des variantes du sac à dos et applications. Informatique [cs]. Université de Valenciennes et du Hainaut-Cambrésis, 2009. Français. NNT : 2009VALE0026 . tel-03046120

HAL Id: tel-03046120
https://uphf.hal.science/tel-03046120

Submitted on 8 Dec 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Université de Valenciennes et du Hainaut-Cambrésis

Numéro d'ordre : 09/26

Approches Hybrides pour des Variantes du Sac à Dos et Applications

THÈSE

Présentée et soutenue publiquement le 12 novembre 2009 pour l'obtention du

Doctorat de l'université de Valenciennes et du Hainaut-Cambrésis

Spécialité Automatique et Informatique des Systèmes Industriels et Humains

Discipline : Informatique

Par

Raïd MANSI

Rapporteurs : M. Gérard Plateau, Professeur émérite, Université Paris XIII
M. Gilles Savard, Professeur, Ecole Polytechnique de Montréal

Examinateurs : M. Philippe Chrétienne, Professeur, Université de Paris VI
M. Jaques Teghem, Professeur, Faculté Polytechnique de Mons
M. Michel Vasquez, Enseignant chercheur – HDR, Ecole des Mines d'Alès

Invité : M. François Clautiaux, Maître de conférences, Université de Lille 1

Co-encadrants : Mme Luce Brotcorne, Maître de conférences, Université de Valenciennes, Chargé de Recherche, INRIA Lille Nord-Europe
M. Christophe Wilbaut, Maître de conférences, Université de Valenciennes

Directeur : M. Saïd Hanafi, Professeur, Université de Valenciennes

Laboratoire d'Automatique, de Mécanique et d'Informatique Industrielles et Humaines – UMR 8530

Résumé

Cette thèse porte sur l'étude de variantes du problème de sac à dos. Nous étudions d'abord des problèmes de sac à dos bi-niveaux pour lesquels nous proposons de nouvelles méthodes hybridant programmation dynamique et méthode de séparations et évaluations. Les expériences numériques montrent que les algorithmes présentés sont robustes et surpassent significativement les algorithmes de la littérature. Des heuristiques itératives hybrides basées sur des relaxations sont également conçues pour le problème du sac à dos multidimensionnel à choix multiples. Ce sont des variantes d'un schéma itératif qui convergent théoriquement vers une solution optimale du problème en résolvant une série de sous-problèmes de petite taille produits en exploitant l'information de relaxations. Nos méthodes sont enrichies par des techniques de fixation et l'ajout de coupes et pseudo-coupes induites par la recherche locale et les relaxations pour réduire l'espace de recherche. Les résultats montrent que ces heuristiques convergent rapidement vers des solutions élites et fournissent 13 nouvelles meilleures valeurs sur un ensemble de 33 instances de la littérature. Enfin, une stratégie d'oscillation combinant la programmation mathématique et des heuristiques constructives et destructives est proposée dans le but de résoudre un problème réel pour la gestion de perturbations dans le domaine aérien. Notre approche a été classée seconde d'un challenge international.

Mots clés : Sac à dos bi-niveaux, Sac à dos multidimensionnel à choix multiples, Domaine aérien, Programmation dynamique, Programmation mathématique, Hybridation, Relaxation, Oscillation

---

Laboratoire : LAMIH-UMR CNRS 8530, UVHC, « Le Mont Houy », 59313 Valenciennes Cedex 9, France

---

Abstract

In this thesis, we propose new methods hybridizing dynamic programming and integer programming exact methods to solve bilevel knapsack problems. The numerical experiments show that our algorithms are robust and surpass significantly the algorithms of the literature. Iterative hybrid heuristics based on relaxations are designed for solving multiple-choice multidimensional knapsack problem. These heuristics converge theoretically towards an optimal solution of the problem by solving a series of sub problems of small size produced by exploiting the information of relaxations. Our methods are enriched by using fixation and by the addition of cuts and pseudo-cuts induced by local search and relaxations. These added constraints are used to reduce the search space. The results show that the heuristics converge quickly to good solutions and provide 13 new better values from 33 instances of the literature. Lastly, an oscillation strategy combining the mathematical programming and constructive and destructive heuristics is proposed to solve a real problem for the management of disruptions in the air industry. Our approach was ranked second of an international challenge.

Keywords : Bilevel knapsack problem, Multiple-choice multidimensional knapsack problem, Airline industry, Dynamic programming, Mathematical programming, Hybridization, Relaxation, Oscillation

---

Laboratory : LAMIH-UMR CNRS 8530, UVHC, « Le Mont Houy », 59313 Valenciennes Cedex 9, France

---

Table des Matières

TABLE DES MATIERES ..... **IV**

LISTE DES FIGURES ..... **VIII**

LISTE DES ALGORITHMES ..... **X**

LISTE DES TABLEAUX ..... **XII**

INTRODUCTION **1**

PARTIE 1: PROBLEMES DE SAC A DOS BI-NIVEAUX **5**

1.1 Problème d'optimisation linéaire à deux niveaux **5**

1.2 Problème du sac à dos bi-niveaux **16**
1.2.1 Conditions d'existence de solutions du SDB 19
1.2.2 Programmation Dynamique pour le SDB 22
1.2.2.1 Procédure de descente 23
1.2.2.2 Procédure de remontée 24
1.2.2.3 Illustration 26
1.2.2.4 Programmation dynamique creuse pour le SDB 28
1.2.3 Résultats numériques 28

1.3 Reformulation du problème du sac à dos unidimensionnel bi-niveaux **33**
1.3.1 Phase 1 : Programmation Dynamique pour le problème du suiveur 33
1.3.2 Phase 2 : Reformulation du SDUB 34
1.3.3 Illustration 39
1.3.4 Résultats numériques 40

1.4 Heuristique pour le sac à dos multidimensionnel bi-niveaux **47**
1.4.1 Relaxation surrogate 48
1.4.2 Formulation approchée du SDMB 50

1.5 Conclusions et perspectives **52**

PARTIE 2 : SAC A DOS MULTIDIMENSIONNEL A CHOIX MULTIPLES **55**

2.1 Formulation et Applications du MMKP 55
2.1.1 Formulation 0-1 MIP du MMKP et notations 55
2.1.2 Applications du MMKP 57

2.2 Variantes et Méthodes pour le MMKP 59
2.2.1 Connexion avec des variantes du sac à dos 59
2.2.2 Réduction et Fixation 61
2.2.3 Méthodes existantes pour le MCKP et le MMKP 63

2.3 Relaxations du MMKP 66
2.3.1 Relaxation et Décomposition Lagrangiennes 67
2.3.2 Relaxations Surrogate et Composite 70
2.3.3 Relaxations en Nombres Entiers Mixtes et Semi-Continue 72

2.4 Recherche Locale pour le MMKP 74

2.5 Heuristiques Itératives basées sur les Relaxations pour le MMKP 80
2.5.1 Heuristique basée sur la Programmation Linéaire 81
2.5.2 Heuristique basée sur la Relaxation en Nombres Entiers Mixtes 83
2.5.3 Heuristique basée sur la Relaxation Semi-Continue 85

2.6 Résultats numériques 87

2.7 Conclusions et perspectives 94

PARTIE 3 : GESTION DE PERTURBATIONS DANS LE DOMAINE AERIEN 97

3.1 Description du Problème 98
3.1.1 La flotte des appareils 98
3.1.2 Le réseau d'aéroports 99
3.1.3 Les passagers 100
3.1.4 Perturbations et objectifs 100

3.2 Etude de l'existant 101

3.3 Rétablir la réalisabilité / Générer une solution initiale 104
3.3.1 Contraintes sur les avions 106
3.3.2 Contraintes sur les aéroports 110
3.3.3 Contraintes sur les passagers 112
3.3.4 Contraintes supplémentaires et bornes des variables 113
3.3.5 Fonction objectif 114
3.3.6 Méthodes de résolution 115
3.3.6.1 Relaxation des contraintes de maintenance 116
3.3.6.2 Résolution itérative 117
3.3.6.3 Procédure de réparation 121

3.4 Stratégie d'oscillation 125
3.4.1 Heuristiques constructives 126
3.4.1.1 Génération des routes pour les avions 126
3.4.1.2 Génération des itinéraires pour les passagers 128
3.4.1.3 Modèle de création des routes et d'affectation des passagers 130

3.4.2 Heuristique destructive 133

3.5 Résultats numériques 134

3.6 Conclusions 137

BIBLIOGRAPHIE : 139

Liste des figures

FIGURE 1.1 : RESOLUTION DU PROBLEME $CBLP_1$ 8
FIGURE 1.2-A : X ET Y SONT DES ENTIERS 10
FIGURE 1.2-B : X ET Y SONT DES ENTIERS ET (1-B') : $Y \geq 3/2$ 11
FIGURE 1.2-C : X ENTIER ET Y CONTINU 11
FIGURE 1.2-D : X CONTINU ET Y ENTIER 11
FIGURE 1.2-E : X CONTINU, Y ENTIER ET (1-B') : $Y \geq 3/2$ 12
FIGURE 1.3-A : ALGORITHME DE BARD ET MOORE (1992) POUR LE 0-1 MBLP 13
FIGURE 1.3-B : ALGORITHME DE MOORE ET BARD (1990) POUR LE MBLP 15
FIGURE 1.4: PROBLEME D'EXISTENCE D'UNE SOLUTION OPTIMALE 20
FIGURE 1.5 : ESPACE MEMOIRE POUR *PD* ET *PDD* 32
FIGURE 1.6 : INFLUENCE DE T SUR LE NOMBRE D'OBJETS ET LA CAPACITE 32
FIGURE 1.7-A : IMPACT DE A SUR P 43
FIGURE 1.7-B : IMPACT DE A SUR LE CPU_IP 43
FIGURE 1.8-A : IMPACT DE LA CORRELATION SUR P 44
FIGURE 1.8-B : IMPACT DE LA CORRELATION SUR CPU_IP 44
FIGURE 1.9 : EVOLUTION DE LA BORNE INFERIEURE DE LA REFORMULATION 46

FIGURE 2.1 : CONNEXIONS ENTRE LE MMKP ET D'AUTRES VARIANTES DU SAC A DOS 60
FIGURE 2.2 : ILLUSTRATION D'UN MOUVEMENT DANS LE VOISINAGE V_4 78
FIGURE 2.3 : BORNES SUPERIEURES POUR INS02 94

FIGURE 3.1 : CONTINUITE DES VOLS D'UN AVION. 106
FIGURE 3.2 : VOLS FICTIFS 107
FIGURE 3.3 : VOLS CREES 107
FIGURE 3.4 : LES DIFFERENTS VOLS DU RESEAU 108
FIGURE 3.5 : CAPACITES DES AEROPORTS 110
FIGURE 3.6 : TRANSIT DES PASSAGERS 112
FIGURE 3.7 : SOLUTION REALISABLE INITIALE 124
FIGURE 3.8 : GENERATION DES CHEMINS D'AVIONS 131
FIGURE 3.9 : SUPPRESSION DES CIRCUITS D'UN AVION 134

Liste des algorithmes

ALGORITHME 1.1 : PROCEDURE DE DESCENTE 24
ALGORITHME 1.2 : PROCEDURE DE REMONTEE 26

ALGORITHME 2.1 : ALGORITHME DE DESCENTE 75
ALGORITHME 2.2 : MEILLEUR VOISIN DANS V_4 79
ALGORITHME 2.3 : HEURISTIQUE ITERATIVE BASEE SUR LA PROGRAMMATION LINEAIRE (HIPL) 82
ALGORITHME 2.4 : HEURISTIQUE ITERATIVE BASEE SUR LA RELAXATION MIP (HIM) 84
ALGORITHME 2.5 : HEURISTIQUE ITERATIVE BASEE SUR LA RELAXATION SEMI-CONTINUE (RSC) 86

ALGORITHME 3.1 : LA PREMIERE PHASE 124
ALGORITHME 3.2 : ALGORITHME D'OSCILLATION 125
ALGORITHME 3.3 : PROCEDURE DE CREATION DE TOURNEES 128

Liste des tableaux

TABLEAU 1.1: VALEURS DE $f_k^2(x)$ ET $\widetilde{f}_k^1(x)$ 27
TABLEAU 1.2 : PHASE DE REMONTEE 28
TABLEAU 1.3 : TEMPS D'EXECUTION DES ALGORITHMES EN SECONDES POUR LA TAILLE 100 29
TABLEAU 1.4 : TEMPS D'EXECUTION DES ALGORITHMES EN SECONDES POUR LA TAILLE 1000 30
TABLEAU 1.5 : TEMPS D'EXECUTION DE *PD* ET *PDD* EN SECONDES 31
TABLEAU 1.6 : PHASE DE PROGRAMMATION DYNAMIQUE 39
TABLEAU 1.7 : COMPARAISON ENTRE LES ALGORITHMES *PD_SDUB* ET *M&B* 41
TABLEAU 1.8 : EFFETS DE LA TAILLE DU PROBLEME DU SUIVEUR 42
TABLEAU 1.9 : IMPACT DES CONTRAINTES DU MENEUR 45
TABLEAU 1.10 : QUALITE DE LA BORNE SUPERIEURE DE L'ALGORITHME DE *PD_SDUB* 47

TABLEAU 2.1 : INSTANCES POUR LE MMKP 87
TABLEAU 2.2 : VALEURS REFERENCES POUR L'EVALUATION DE NOS APPROCHES 88
TABLEAU 2.3 : RESULTATS NUMERIQUES DE HIPL 90
TABLEAU 2.4 : RESULTATS NUMERIQUES DE HIM 91
TABLEAU 2.5 : RESULTATS NUMERIQUES DE RSC 92
TABLEAU 2.6 : SYNTHESE DES RESULTATS 93

TABLEAU 3.1 : RESULTATS POUR LES INSTANCES A 135
TABLEAU 3.2 : RESULTATS POUR LES INSTANCES B ET X 136

Introduction

La présente thèse est réalisée au sein de l'équipe SIADE (Systèmes d'Information, d'Aide à la Décision et Embarqués) qui fait partie du LAMIH (Laboratoire d'Automatique, de Mécanique et d'Informatique Industrielles et Humaines) de l'Université de Valenciennes. L'ensemble des travaux présentés dans ce mémoire a été réalisé entre Octobre 2006 et Octobre 2009 et rentre dans le cadre du thème recherche opérationnelle et aide à la décision (ROAD).

La résolution de problèmes d'optimisation complexes de grandes tailles a motivé et motive encore à l'heure actuelle le développement de méthodes de résolutions approchées, également appelées heuristiques. Le but de ces méthodes est l'obtention d'une solution proche de l'optimum dans un temps de calcul raisonnable tout en limitant l'espace mémoire utilisé. L'évaluation d'une heuristique repose sur la rapidité d'obtention des solutions initiales et la convergence vers une solution optimale (Rardin et Uzsoy (2001)). Les métaheuristiques (telles que la recherche Tabou, le recuit simulé, les algorithmes génétiques, la recherche dispersée,...) ont déjà prouvé leur capacité à fournir des solutions de bonne qualité en des temps raisonnables.

Dans cette thèse, nous nous intéressons à des techniques d'hybridation (Blum, Aguilera, Roli et Sampels (2008)). Les méthodes hybrides, qui combinent méthodes exactes et/ou approchées, sont également réputées pour leur efficacité dans la résolution de problèmes d'optimisation difficiles. De nombreuses approches récentes combinent ainsi la programmation mathématique (programmation dynamique, algorithme de séparations et évaluations, relaxations), la programmation par contraintes et les heuristiques ou métaheuristiques.

Plusieurs classes de méthodes hybrides peuvent être recensées. Un premier groupe est composé de méthodes hybrides dites classiques, où des techniques de recherches locales sont employées pour améliorer les solutions produites par des heuristiques basées sur la génération de populations de solutions (par exemple combiner une recherche locale avec un algorithme génétique). Une branche importante des méthodes hybrides vise à améliorer les

métaheuristiques à l'aide de techniques additionnelles pour minimiser le temps d'exécution, améliorer les résultats, ou les deux à la fois. Durant les dernières années, de nombreuses méthodes proposées dans la littérature sont ainsi basées sur une hybridation entre méthodes exactes et heuristiques (par exemple le branchement local (Fischetti et Lodi (2003)). Ainsi, pour concevoir un algorithme de résolution pour un problème d'optimisation, nous pouvons décomposer le processus de résolution en plusieurs tâches, puis appliquer pour chacune d'elles la technique de résolution la plus performante selon le critère choisi (le temps d'exécution, la qualité de l'information obtenue). Chaque technique peut être une méthode exacte, une heuristique, une métaheuristique. Elle peut par exemple utiliser la programmation par contraintes ou des techniques d'intelligence artificielle.

Nos travaux s'intéressent à la mise en œuvre de telles méthodes pour résoudre des variantes de problèmes de sac à dos (Kellerer, Pferschy et Pisinger (2004)) et une application réelle. Les problèmes de la famille du sac à dos sont des problèmes fondamentaux en optimisation combinatoire qui modélisent de nombreuses applications industrielles, en particulier avec l'ajout de contraintes supplémentaires. Ces problèmes difficiles sont aussi devenus des problèmes tests pour les nouvelles approches de résolution et apparaissent souvent comme sous-problèmes de problèmes réels. Les problèmes de type sac à dos consistent à trouver une meilleure sélection d'un sous-ensemble d'objets pour les mettre dans un ou plusieurs sacs en respectant la ou les capacités et les contraintes supplémentaires éventuelles. Plusieurs variantes du sac à dos peuvent être distinguées selon la distribution des objets et des sacs. Nous citons ici quelques variantes : le problème du sac à dos lui-même, le plus simple par sa formulation, dans lequel chaque objet peut être au maximum choisi une seule fois ; le sac à dos borné où chaque objet peut être choisi plusieurs fois ; le sac à dos non borné où chaque objet peut être choisi une infinité de fois ; le sac à dos à choix multiples où les objets sont regroupés dans des sous-ensembles de sorte qu'un seul objet peut être choisi dans chaque sous-ensemble. Chacun de ces problèmes peut être étendu au cas comportant plusieurs contraintes de sac à dos : le sac à dos multidimensionnel ; le sac à dos multidimensionnel borné ; le sac à dos multidimensionnel non borné ; le sac à dos multidimensionnel à choix multiples. De nombreuses autres variantes peuvent être recensées dans la littérature, en changeant le type de la fonction objectif comme le problème du sac à dos quadratique, le sac à dos multi-objectif, le sac à dos max-min, le sac à dos non linéaire et le sac à dos bi-niveaux.

Cette thèse comprend trois parties principales, traitant plus particulièrement de deux variantes de problèmes de sac à dos : le problème du sac à dos bi-niveaux et le problème du

sac à dos multidimensionnel à choix multiples. Chaque partie est présentée de manière à pouvoir être lue indépendamment des autres.

Dans la première partie nous étudions des problèmes de sac à dos bi-niveaux qui permettent de modéliser des systèmes d'interactions hiérarchiques entre deux agents de décision, comme dans le problème de Stackelberg (1952). Le premier agent, appelé meneur, fixe d'abord ses décisions. Le deuxième agent, appelé suiveur, doit quant à lui prendre les meilleures décisions possibles en prenant en compte les valeurs fixées par le meneur. Le meneur contrôle les interactions et doit donc prendre les décisions qui vont engendrer la meilleure réponse du suiveur.

Le premier type de problème de sac à dos bi-niveaux (SDB) étudié comporte une seule variable pour le meneur (la capacité du sac) et une seule contrainte de sac à dos pour le suiveur. Le meneur a donc pour but de déterminer la valeur de capacité du sac maximisant son profit en prenant compte de la réaction du suiveur qui, pour une capacité fixée, sélectionne un sous-ensemble d'objets afin d'atteindre son propre objectif. Nous proposons un nouvel algorithme exact basé sur la programmation dynamique pour résoudre ce problème.

Le second type de problème de sac à dos bi-niveaux comprend plusieurs variables et contraintes au niveau du meneur et un problème de sac à dos pour le suiveur (SDUB). Pour ce problème nous proposons une méthode exacte hybride qui combine la programmation dynamique et la résolution d'un problème linéaire en nombres entiers. Plus précisément, l'application de la première phase de la programmation dynamique sur le problème du suiveur en ignorant les ressources consommées par le meneur permet de définir un domaine sous-jacent à la reformulation du SDUB sous la forme d'un programme linéaire en nombres entiers mixtes. Ce dernier peut être résolu de façon efficace par un algorithme classique de type séparations et évaluations.

Les expériences numériques montrent que nos algorithmes sont robustes et surpassent de manière significative les algorithmes proposés dans la littérature. Nous concluons la première partie de cette thèse en proposant une extension possible de la méthode proposée pour le SDUB au cas du problème de sac à dos bi-niveaux multidimensionnel. Cette extension résulte en une méthode de résolution approchée.

Dans la seconde partie de cette thèse, nous proposons des heuristiques itératives hybrides basées sur des relaxations pour le problème du sac à dos multidimensionnel à choix multiples. Cette variante du sac à dos multidimensionnel comporte des contraintes supplémentaires, dites de choix, qui renforcent la difficulté du problème. De nombreuses applications réelles de ce problème peuvent être recensées dans la littérature dans le domaine du multimédia ou de

l'affectation de ressources par exemple. Nous avons travaillé sur plusieurs variantes d'un schéma itératif qui convergent théoriquement vers une solution optimale du problème en résolvant une série de sous-problèmes de petite taille produits en exploitant l'information obtenue à partir de relaxations. Nous avons également intégré de nouveaux éléments : une nouvelle relaxation, des techniques de fixation, et l'ajout de coupes et pseudo-coupes induites par une recherche locale. Nos méthodes fournissent des bornes inférieure et supérieure sur la valeur optimale et fixent définitivement certaines variables à leurs valeurs optimales. Elles sont valides pour les programmes linéaires en variables binaires. Les résultats montrent que les heuristiques convergent rapidement vers des solutions élites et fournissent 13 nouvelles meilleures bornes inférieures sur un ensemble de 33 instances de la littérature.

La dernière partie de la thèse s'intéresse à la résolution d'un problème réel dans le domaine de la gestion de perturbations aériennes rencontré dans le cadre d'un challenge international organisé par la Société Française de Recherche Opérationnelle et Aide à la Décision (ROADEF). Un des principaux objectifs du problème était de mieux prendre en compte le ressenti passager lors de perturbations à travers des pénalités supplémentaires pour la compagnie aérienne. Devant la difficulté et la taille de ce problème, nous avons proposé une approche hybride combinant la programmation mathématique et un schéma d'oscillation. Après avoir construit une première solution réalisable à l'aide d'un modèle en nombres entiers mixtes, nous alternons des phases constructives et destructives permettant de créer un comportement oscillatoire entre la génération et la suppression de routes pour les avions et d'itinéraires pour les passagers dans le but d'améliorer la meilleure solution courante. Nous avons également pu modéliser un sous-problème comme un sac à dos multidimensionnel en nombres entiers et binaires, justifiant une nouvelle fois l'intérêt de ces problèmes pour la communauté de la recherche opérationnelle. Notre approche a été classée seconde et confirme que l'hybridation entre méthodes exactes et approchées est efficace pour des problèmes NP-Difficiles de grande taille.

Partie 1: Problèmes de sac à dos bi-niveaux

Les problèmes d'optimisation à deux niveaux, encore appelés problèmes bi-niveaux, permettent de modéliser les interactions hiérarchiques entre deux agents de décision. Dans cette partie nous considérons l'étude de problèmes bi-niveaux avec la structure de sac à dos. Dans la Section 1, nous présentons une introduction générale sur les problèmes d'optimisation bi-niveaux et la particularisons au cas des problèmes bi-niveaux linéaires avec variables discrètes. Dans la Section 2, nous étudions un problème de sac à dos qui est un problème bi-niveaux avec variables discrètes comportant une seule variable pour le meneur (la capacité du sac) et une seule contrainte de sac à dos pour le suiveur. Nous proposons un nouvel algorithme exact basé sur la programmation dynamique pour résoudre ce problème. La troisième section est consacrée à la résolution d'un problème de sac à dos bi-niveaux comprenant plusieurs variables et contraintes au niveau du meneur et un problème de sac à dos pour le suiveur. Pour ce problème nous proposons une méthode exacte hybride qui combine la programmation dynamique et la résolution d'un problème linéaire en nombres entiers. Dans la quatrième section nous étendons l'approche proposée à la Section 3 au cas du problème de sac à dos bi-niveaux multidimensionnel. Cette extension résulte en une méthode de résolution approchée. Finalement nous donnons des conclusions et perspectives dans la Section 5.

1.1 Problème d'optimisation linéaire à deux niveaux

Les problèmes bi-niveaux permettent de modéliser des systèmes d'interactions hiérarchiques entre deux agents de décision, comme dans le problème de Stackelberg (1952). Même si leur étude fut proposée en 1973 dans un article de Bracken et McGill (1973), la recherche sur ce type de problème n'a réellement commencé que dans les années 1980. Dans un problème bi-niveaux, le premier agent, appelé le meneur, fixe ses décisions en tenant compte explicitement du comportement d'un second agent, appelé le suiveur. Le meneur, qui contrôle les interactions, doit donc prendre les décisions engendrant la meilleure réponse du suiveur. Ainsi le problème à deux niveaux se formalise comme un problème d'optimisation

pour le meneur dont l'ensemble des contraintes contient le problème d'optimisation du suiveur. Une revue de la littérature détaillée des problèmes à deux niveaux est donnée dans Shimizu, Ishisuka et Bard (1997). Dans cette section, nous nous limitons à l'étude des problèmes bi-niveaux (**BLPP**) pour lesquels les fonctions objectif et les fonctions définissant les contraintes sont linéaires. Un problème BLPP peut être formulé de la façon suivante :

$$\text{(BLPP)} \begin{cases} Max_x f^1(x,y) = c^1 x + d^1 y \\ \quad \text{sous les contraintes} \\ \quad\quad B^1 x + B^2 y \le b^1 \\ \quad\quad x \in X \\ \quad\quad Max_y f^2(x,y) = c^2 x + d^2 y \\ \quad\quad\quad s.c. \\ \quad\quad\quad\quad A^1 x + A^2 y \le b^2 \\ \quad\quad\quad\quad y \in Y \end{cases}$$

où c^1, $c^2 \in \Re^{n^1}$, d^1, $d^2 \in \Re^{n^2}$, $b^1 \in \Re^{m^1}$, $b^2 \in \Re^{m^2}$, $B^1 \in \Re^{m^1 \times n^1}$, $A^1 \in \Re^{m^2 \times n^1}$, $B^2 \in \Re^{m^1 \times n^2}$, $A^2 \in \Re^{m^2 \times n^2}$. Le vecteur de variables du meneur est noté $x \in X \subseteq \Re^{n^1}$, et celui des variables du suiveur est noté $y \in Y \subseteq \Re^{n^2}$. Nous présentons ici un ensemble de notations associées au problème BLPP.

- La région admissible,

$$S = \left\{(x,y) \in X \times Y \middle| B^1 x + B^2 y \le b^1, A^1 x + A^2 y \le b^2\right\}.$$

- L'ensemble des solutions réalisables du suiveur, pour x fixé,

$$S(x) = \left\{y \in Y \middle| A^1 x + A^2 y \le b^2\right\}.$$

- La projection de S sur l'espace de décisions du meneur,

$$S(X) = \left\{x \in X \middle| \exists y \in Y : B^1 x + B^2 y \le b^1, A^1 x + A^2 y \le b^2\right\}.$$

- L'ensemble des réactions du suiveur, pour x fixé,

$$R(x) = \left\{y \in Y \middle| y \in \arg\max\left\{f^2(x,y) \middle| y \in S(x)\right\}\right\}.$$

- La région induite R ou domaine sur lequel optimise le meneur,

$$R = \left\{(x,y) \in S \middle| y \in R(x)\right\}.$$

La formulation BLPP n'est pas suffisante pour définir une solution optimale d'un problème bi-niveaux. En effet, dans le cas où le problème du suiveur admet plusieurs solutions optimales pour des variables du meneur fixées, deux approches peuvent être considérées pour définir une solution du problème BLPP (voir Loridan et Morgan (1989), Loridan et Morgan (1996)).

Approche optimiste : Dans ce cas nous considérons que le meneur est capable d'influencer le choix du suiveur afin d'être favorisé : $\underset{x}{Max}\left\{\underset{y}{Max}\left\{f^2(x,y): y \in \mathrm{R}(x)\right\}: (x,y) \in \mathrm{S}\right\}$. Une solution de ce problème est appelée une *solution forte.*

Approche pessimiste : Contrairement au cas optimiste, le choix du suiveur ne favorise pas les intérêts du meneur : $\underset{x}{Max}\left\{\underset{y}{Min}\left\{f^2(x,y): y \in \mathrm{R}(x)\right\}: (x,y) \in \mathrm{S}\right\}$. Une solution de ce problème est appelée une *solution faible.*

Un problème bi-niveaux linéaire avec variables continues, noté **CBLP**, est un problème BLPP où les fonctions objectif et les contraintes du problème sont linéaires et toutes les variables du problème sont continues (i.e. $X = R_+^{n^1}$ et $Y = R_+^{n^2}$). La difficulté majeure intervenant dans l'étude des problèmes à deux niveaux est qu'ils sont généralement non convexes et non différentiables (Calamai et Vicente (1993), Luo, Pang et Ralph (1996)). De plus, l'existence de contraintes de premier niveau peut engendrer un ensemble induit R discontinu ou vide (voir par exemple Edmunds et Bard (1991), Dempe (2002)).

Nous illustrons ces éléments à partir de l'exemple suivant.

Exemple 1 : Soit le CBLP suivant :

$$(\mathrm{CBLP}_1)\begin{cases} \underset{x}{Max}\ f^1 = x + 6y \\ \quad s.c. \\ \quad\quad x \geq 3/2 & (1\text{-a}) \\ \quad\quad y \geq 1/2 & (1\text{-b}) \\ \quad\quad \underset{y}{Max}\ f^2 = -y \\ \quad\quad\quad s.c. \\ \quad\quad\quad x - 2y \geq 5 & (1\text{-c}) \\ \quad\quad\quad 3x + 7y \leq 28 & (1\text{-d}) \\ \quad\quad\quad 2x + 5y \geq 10 & (1\text{-e}) \end{cases}$$

Sur la Figure 1.1, la région admissible S est définie par le polyèdre de sommets P^1, P^2, P^3, P^4 et P^5. Si le meneur ne tient pas compte de la réaction du suiveur et optimise sur S, la solution optimale est donnée par le point P^5. La projection de S sur l'espace de décision du meneur S(X) est [3/2, 15/4] ∪ [6, 7]. Pour une valeur $x_0 \in$ S(X) fixée par le meneur, le suiveur réagit en sélectionnant le point de plus petite ordonnée y_0 parmi les points du segment d'abscisse x_0 contenu dans S. La région induite, ou domaine sur lequel optimise le meneur, est donc définie par l'union des segments [P^1, P^2] et [P^3, P^4]. Cet ensemble est clairement non convexe. De

plus, il est discontinu à cause de l'existence de la contrainte (1-b). Notons que la contrainte (1-a) ne provoque pas une telle situation car elle ne dépend pas des variables de décision du suiveur. Le meneur peut donc assurer la satisfaction de cette contrainte lors de son action car la réaction du suiveur n'a aucune influence sur cette contrainte.

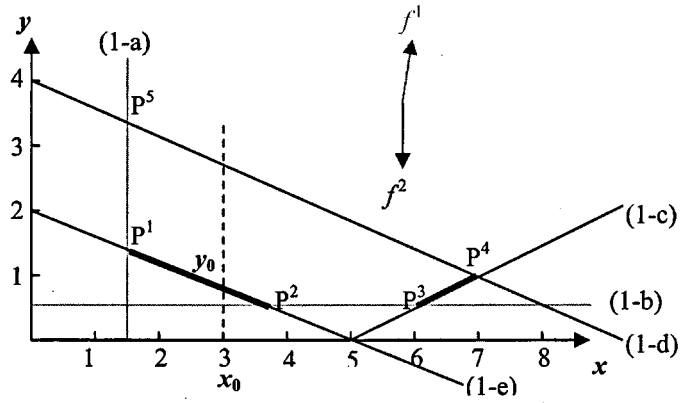


Figure 1.1 : Résolution du problème $CBLP_1$

Finalement, le problème $CBLP_1$ admet P^4 comme optimum global et P^1 comme optimum local.

Du point de vue de la complexité, CBLP est un problème fortement NP-Difficile (Hansen, Jaumard et Savard (1992)). De plus, la vérification de l'optimalité locale d'une solution du CBLP est aussi un problème NP-Difficile (Vicente, Savard et Judice (1994)). Parmi les propriétés les plus importantes du CBLP, nous pouvons citer :

(i) Une solution optimale du CBLP appartient à la région induite R (Bialas et Karwan (1984)).

(ii) Une solution optimale du CBLP est toujours un point extrême de la région admissible S (Shimizu, Ishisuka et Bard (1997)).

(iii) S'il existe un point extrême de la région induite R alors il est un point extrême de la région admissible S (Shimizu, Ishisuka et Bard (1997)).

Les méthodes de résolution développées pour résoudre le problème CBLP sont principalement basées sur la présentation spécifique de l'ensemble R. Elles peuvent être classées en quatre catégories : i) *Algorithmes énumératifs* (Bialas et Karwan(1984)), ii) *Approches par fonction de pénalité* (Anandalingam et White (1990)), iii) *Approches basées sur la complémentarité* (Bialas, Karwan et Shaw (1980), Júdice et Faustino (1992)) et iv) *Algorithmes basés sur la méthode de séparations et évaluations* (Bard et Moore (1990), Hansen, Jaumard et Savard (1992), Savard et Zghal (2007)).

Une reformulation équivalente du problème CBLP (voir Shimizu, Ishisuka et Bard (1997)) est basée sur la propriété suivante : si $X = R_+^{n^1}$ et $Y = R_+^{n^2}$ alors une condition

nécessaire pour que la solution (x^*, y^*) soit une solution optimale du problème CBLP est qu'il existe un vecteur u^* tel que (u^*, x^*, y^*) soit une solution optimale du problème suivant :

$$(\text{CBLP}^*)\begin{cases} Max \;\; c^1x + d^1y \\ \quad s.c. \\ \qquad B^1x + B^2y \le b^1 \\ \qquad A^1x + A^2y \le b^2 \\ \qquad y(d^2 - uA^2) = 0 \qquad (2-\text{a}) \\ \qquad u(b^2 - A^1x - A^2y) = 0 \qquad (2-\text{b}) \\ \qquad x, y, u \ge 0 \end{cases}$$

Les variables u sont les variables duales du problème du suiveur. Les contraintes (2-a) et (2-b) sont les conditions nécessaires et suffisantes d'optimalité du problème du suiveur appelées conditions d'optimalité de *Karuch-Khun-Tuker* (KKT). Ces conditions sont à la base d'une transformation du CBLP en un programme linéaire avec variables mixtes (Fortuny-Amat et McCarl (1981), Audet, Hansen, Jaumard et Savard (1997)). Le principe de cette transformation consiste à remplacer le problème du second niveau par ces conditions d'optimalité. Des variables binaires α, β et une constante M suffisamment grande sont introduites pour linéariser les contraintes bilinéaires (2-a) et (2-b). Ainsi, le problème CBLP* peut être reformulé sous la forme du problème linéaire en nombres entiers mixtes suivant :

$$(\text{CBLP}^{\text{MIP}})\begin{cases} Max \;\; c^1x + d^1y \\ \quad s.c. \\ \qquad B^1x + B^2y \le b^1 \\ \qquad A^1x + A^2y \le b^2, \qquad uA^2 \ge d^2 \\ \qquad b^2 - A^1x - A^2y \le M(e-\alpha), \quad u \le M\alpha \\ \qquad y \le M(e-\beta), \qquad uA^2 - d^2 \le M\beta \\ \qquad x, y, u \ge 0, \quad \alpha \in \{0,1\}^{m^2}, \beta \in \{0,1\}^{n^2} \end{cases}$$

où e est un vecteur de composantes égales à 1 de dimension m^2.

Par la suite, ce modèle CBLP$^{\text{MIP}}$ peut être résolu à l'aide de méthodes d'optimisation classiques (par exemple de type séparations et évaluations (ou *Branch-and-Bound*)). Nous renvoyons le lecteur au livre de Shimizu, Ishisuka et Bard (1997) ou au survey de Colson, Marcotte et Savard (2005 et 2007) pour une présentation complète des méthodes de résolution du CBLP.

Le but de cette partie de la thèse étant l'étude de problèmes bi-niveaux avec la structure de sac à dos, nous nous concentrons par la suite sur la présentation des propriétés et des

méthodes de résolution des problèmes bi-niveaux linéaires avec variables entières ou mixtes (**MBLP**).

La présence de variables entières dans un problème bi-niveaux pose des problèmes pour l'existence d'une solution optimale. En effet, nous sommes assurés de l'existence d'une solution optimale pour le MBLP uniquement dans le cas où la région admissible S est compacte et non vide et où

- soit les variables du meneur x et les variables du suiveur y sont continues,
- soit les variables du meneur x et les variables du suiveur y sont des entiers,
- soit les variables du meneur x sont mixtes et les variables du suiveur sont continues.

Dans le cas où les variables du meneur x sont continues et celles du suiveur y sont entières, le problème MBLP n'admet pas toujours de solution optimale car la région induite n'est pas compacte.

Nous illustrons ces propriétés ainsi que le lien entre un problème bi-niveaux linéaire mixte et sa relaxation sur l'exemple $CBLP_1$ où nous imposons des contraintes d'intégrité sur les variables du meneur et / ou du suiveur. Considérons d'abord le cas où x et y sont des entiers.

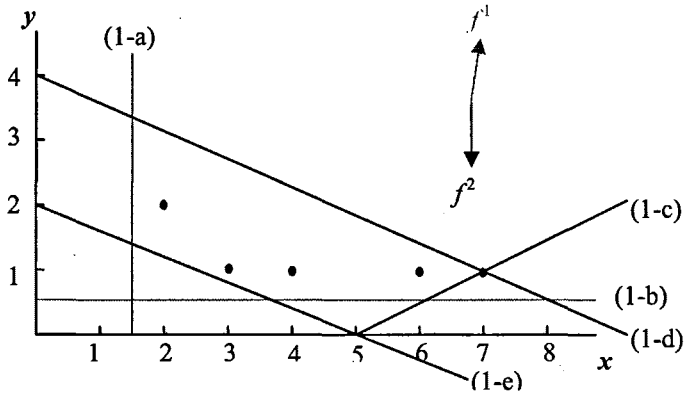


Figure 1.2-a : x et y sont des entiers

La solution optimale est (2,2). Notons que la solution optimale (7,1) de la relaxation ne fournit pas une borne supérieure pour le problème.

Dans le cas où les variables x et y sont des entiers et la contrainte (1-b) est remplacée par (1-b') $y \geq 3/2$, la relaxation du problème n'admet pas de solution, même si le problème d'origine admet une solution optimale (2,2).

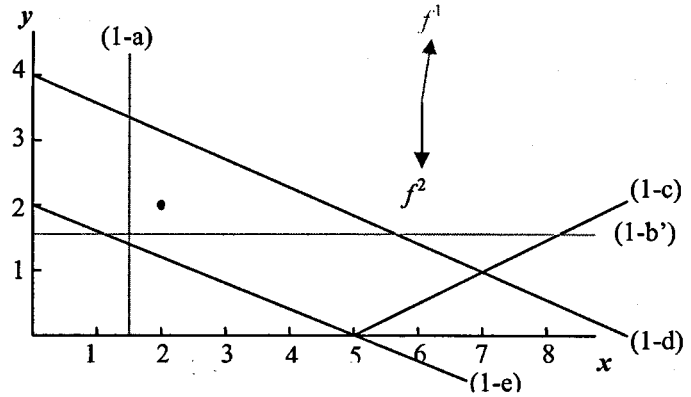


Figure 1.2-b : x et y sont des entiers et (1-b') : $y \geq 3/2$

Dans le cas où x est entier et y est continu, la région induite du MBLP est contenue dans la région induite de sa relaxation (Figure 1.2-c). La solution optimale de la relaxation définit donc une borne supérieure sur la valeur optimale du problème d'origine.

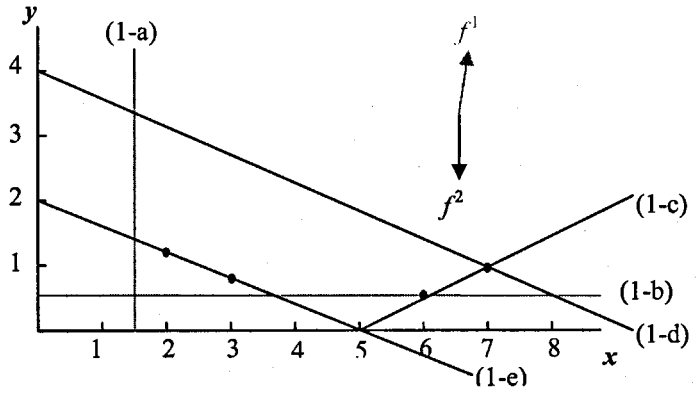


Figure 1.2-c : x entier et y continu

Dans le cas où x est continu et y est entier, une approximation de la valeur optimale est donnée par $\lim_{x \to 5/2^{>}} f^1(x,2)$. De plus, la solution optimale de la relaxation ne définit pas une borne supérieure pour le problème d'origine.

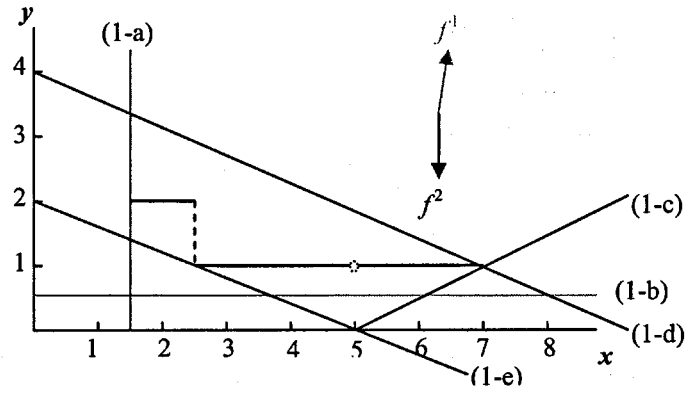


Figure 1.2-d : x continu et y entier

Finalement dans le cas où x est continu, y est entier et la contrainte (1-b) est remplacée par (1-b') ($y \geq 3/2$), la région induite de la relaxation est vide alors qu'elle est non-vide pour le problème d'origine. L'approximation de la valeur optimale peut être donnée par $\lim_{x \to 5/2^{>}} f^1(x,2)$.

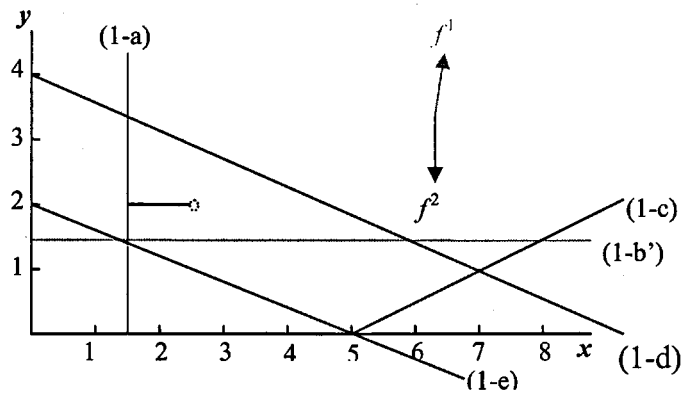


Figure 1.2-e : x continu, y entier et (1-b') : $y \geq 3/2$

En conclusion, il suit que la résolution d'un problème bi-niveaux mixte est plus complexe lorsqu'il existe des variables continues pour le meneur, des variables entières pour le suiveur et des contraintes au premier niveau avec des variables du suiveur. D'autres résultats théoriques concernant l'existence de solutions optimales selon la nature des variables peuvent être trouvés dans Vicente, Savard et Judice (1996), Dempe (2002).

Nous terminons cette section par une revue de la littérature des méthodes de résolution pour le MBLP. Nous présentons en détails deux algorithmes qui serviront de base pour les comparaisons des performances d'algorithmes lors des expérimentations numériques.

L'algorithme de Bard et Moore (1992) est basé sur la méthode de séparations et évaluations. Il fut développé pour la résolution de problèmes MBLP avec des variables binaires pour le meneur et le suiveur, que nous notons par 0-1 MBLP.

$$\text{(0-1 MBLP)} \begin{cases} Max_x f^1(x,y) = c^1 x + d^1 y \\ \quad \text{s.c.} \\ \quad\quad x \in \{0,1\}^{n^1} \\ \quad\quad Max_y f^2(x,y) = c^2 x + d^2 y \\ \quad\quad\quad \text{s.c.} \\ \quad\quad\quad\quad A^1 x + A^2 y \leq b^2 \\ \quad\quad\quad\quad y \in \{0,1\}^{n^2} \end{cases}$$

Dans cet algorithme, le 0-1 MBLP est reformulé de sorte que la fonction objectif du meneur devienne une contrainte avec un paramètre $\beta \leq f^1(x,y)$, et en ajoutant une contrainte sur les variables du meneur :

$$\text{(0-1 MBLP) } (k,\beta) \begin{cases} Max_x\ f^2(x,y) = c^2x + d^2y \\ \text{s.c.} \\ A^1x + A^2y \le b^2 \\ f^1(x,y) = c^1x + d^1y \ge \beta & (3-a) \\ \sum_{i=1}^{n^1} x_i \ge k & (3-b) \\ x \in \{0,1\}^{n^1},\quad y \in \{0,1\}^{n^2} \end{cases}$$

Les paramètres β et k sont initialisés à $\beta = -\infty$, $k = 0$. Notons que k représente le nombre minimum de variables devant être fixées à 1. L'algorithme est décrit dans la Figure 1.3-a.

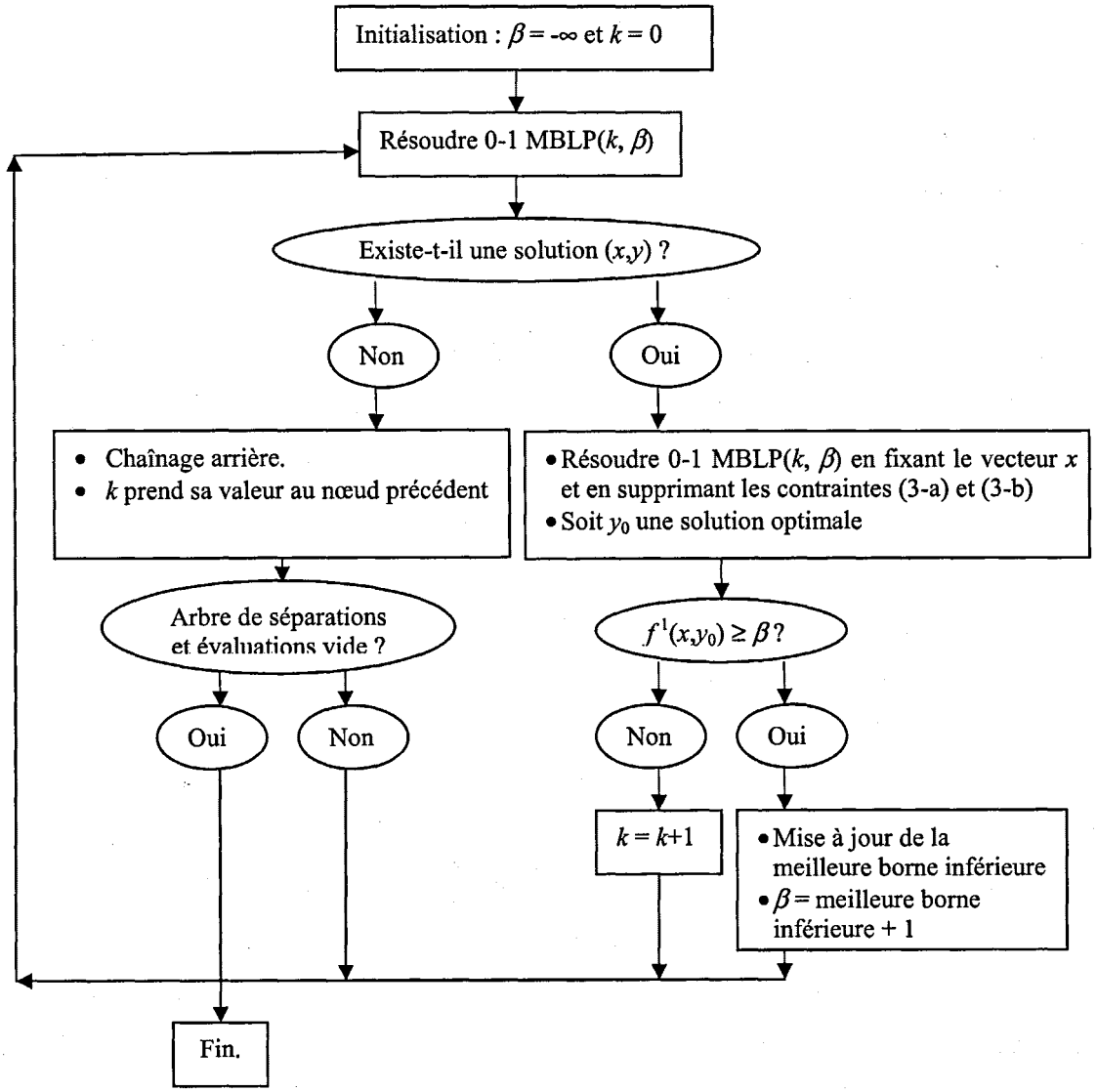


Figure 1.3-a : Algorithme de Bard et Moore (1992) pour le 0-1 MBLP

Le premier pas consiste à résoudre le modèle (0-1 MBLP) (k, β). Si une solution optimale est obtenue, les valeurs des variables du meneur x_i sont fixées à la valeur de cette solution et le

problème 0-1 MBLP est résolu pour trouver une solution appartenant à la région induite R. Le second pas consiste à faire les mises à jour suivantes :

- β est fixé à la meilleure valeur de la fonction objectif du meneur trouvée, plus 1.
- k est fixé au nombre de variables du meneur fixées à 1, plus 1, et l'algorithme crée une nouvelle branche sur une variable x_i non fixée.

De façon classique, un chaînage arrière (retour au nœud père dans l'arbre de séparations et évaluations) est appliqué si aucune solution respectant les contraintes (3-a) et (3-b), paramétrées par k et β, n'est trouvée.

L'algorithme de Moore et Bard (1990) fait partie des premières méthodes proposées pour résoudre le MBLP. Cet algorithme ne permet pas de résoudre des problèmes MBLP avec des contraintes de premier niveau incluant des variables du suiveur. Le principe de base est d'effectuer une énumération implicite en s'appuyant sur le principe de séparations et évaluations. La borne supérieure considérée pour l'élagage dans l'arbre est la valeur optimale du problème MBLP obtenue en relâchant les contraintes d'intégrité et en ignorant la fonction objectif du suiveur. Le problème résultant est appelé HPP (*High Point Problem*) et formulé comme suit :

$$\text{(HPP)} \begin{cases} Max_x f^1(x,y) = c^1 x + d^1 y \\ \quad \text{s.c.} \\ \quad\quad B^1 x \le b^1 \\ \quad\quad A^1 x + A^2 y \le b^2 \\ \quad\quad x \ge 0 \\ \quad\quad y \ge 0 \end{cases}$$

Notons que si les contraintes d'intégrité sont conservées, la solution obtenue définit également une borne, mais ces conditions sont généralement supprimées pour des raisons de rapidité d'exécution.

L'algorithme de Moore et Bard (1990) peut être décrit de la façon suivante (Figure 1.3-b) : la fonction objectif du meneur f^1_{opt} est initialisée à -∞. A chaque itération, l'algorithme résout la relaxation HPP. Si la valeur optimale v(HPP) est meilleure que f^1_{opt}, l'algorithme résout ensuite la relaxation en continu CBLP (i.e. en ignorant les contraintes d'intégrité). Si la solution optimale du CBLP générée respecte les conditions d'intégrité, les variables du meneur sont fixées et le problème du suiveur est résolu afin de mettre à jour la meilleure

valeur f_{opt}^{1}. Dans le cas contraire (i.e., la solution optimale du CBLP obtenue ne respecte pas les conditions d'intégrité), un branchement est fait sur une des variables. L'algorithme fait un chaînage arrière s'il n'y a pas de solution ou si la valeur de la fonction objectif du meneur trouvée dans une branche est de moins bonne qualité que f_{opt}^{1}. Récemment, DeNegre et Ralphs (2008) ont proposé une amélioration de cet algorithme où la borne supérieure est améliorée par l'ajout de coupes valides.

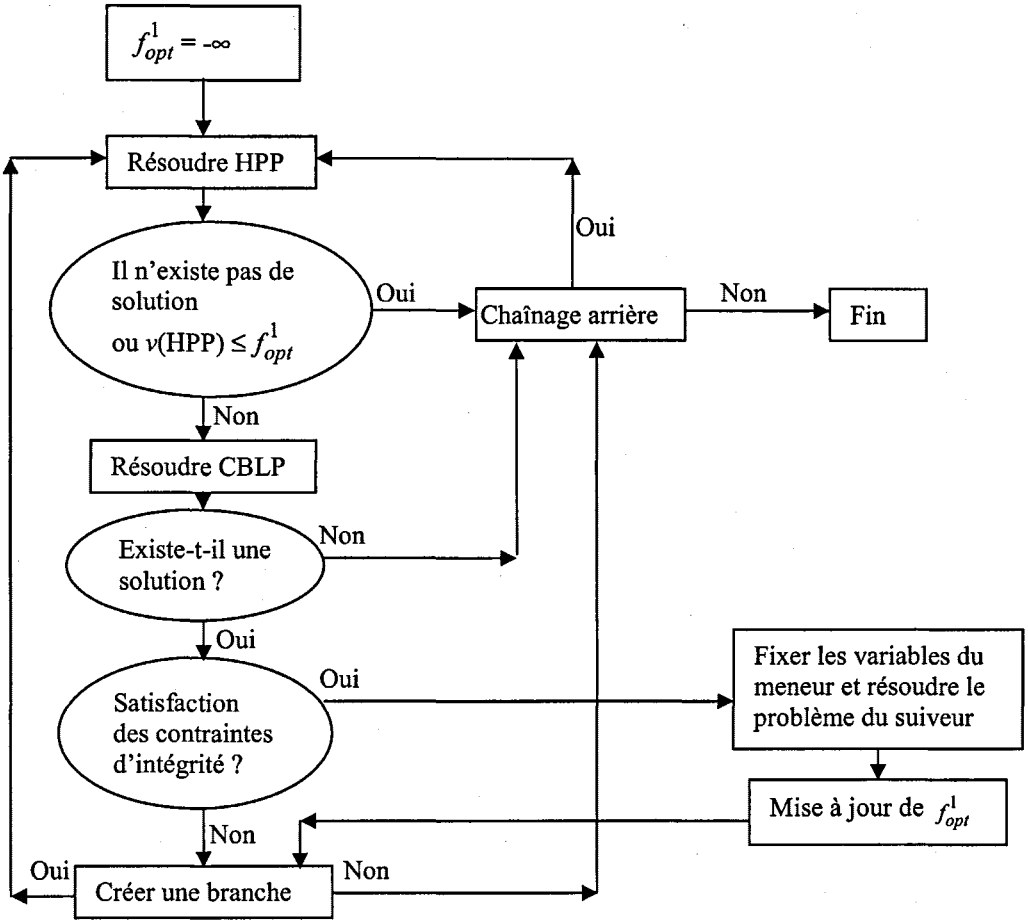


Figure 1.3-b : Algorithme de Moore et Bard (1990) pour le MBLP

D'autres références pour résoudre le problème MBLP existent dans la littérature comme l'algorithme de *Branch-and-Bound* proposé par Wen et Yang (1990) où les variables du suiveur sont continues et celles du meneur sont entières. Les informations duales sont utilisées pour trouver une solution optimale ou presque optimale pour le problème. Dans ce cas, le MBLP peut être écrit sous la forme d'un programme linéaire en nombres mixtes avec un seul niveau en linéarisant les conditions d'optimalité de *Karuch-Khun-Tuker*. Thirwani et Arora (1997) appliquent la méthode "*k^{th} best algorithm*" pour résoudre le MBLP. Une approche de pénalisation pour le MBLP avec un petit nombre de variables entières est aussi présentée par Kalashnikov et Rios-Mercado (2002). Köppe, Queyranne et Ryan (2009) exploitent les

récents résultats d'optimisation paramétrique pour résoudre efficacement un cas particulier de problèmes MBLP. Finalement des algorithmes spécifiques pour résoudre des cas particuliers du MBLP se trouvent dans Dempe (1996) et Dempe et Richter (2000) (pour la structure de sac à dos). L'algorithme de Dempe et Richter (2000) est décrit dans la Section 1.2.

Nous pouvons conclure de cette étude que la résolution d'un problème bi-niveaux linéaire avec variables discrètes est très difficile. De plus, le problème d'obtention d'une solution réalisable est également difficile. Il résulte aussi de notre étude qu'il est nécessaire de prendre en considération la structure spéciale des problèmes traités. Dans la section suivante, nous traitons un cas particulier des problèmes bi-niveaux linéaires avec variables discrètes (Brotcorne, Hanafi et Mansi (2009)). Notre objectif est de développer une méthode de résolution efficace pour ce problème.

1.2 Problème du sac à dos bi-niveaux

Cette section s'intéresse au problème du sac à dos bi-niveaux (**SDB**). Nous commençons par définir le problème et le formuler. Ensuite, nous présentons un nouvel algorithme basé sur la programmation dynamique pour le résoudre.

Dans le cas du problème du sac à dos bi-niveaux, le meneur détermine la capacité du sac de façon à maximiser son profit en tenant compte explicitement de la réaction du suiveur. Ce dernier sélectionne, pour une capacité donnée, un sous-ensemble d'objets de façon à maximiser son profit en respectant la contrainte de capacité. Notons que si le meneur fixe sa décision, le suiveur doit résoudre un problème de sac à dos avec variables binaires (**PSD**). L'objectif du PSD est de trouver un sous-ensemble d'objets de façon à maximiser un profit total tout en satisfaisant une contrainte de capacité (Martello et Toth (1990), Kellerer, Pferschy et Pisinger (2004)).

Le SDB permet par exemple de modéliser un problème de gestion de portefeuille dans lequel un individu (le meneur) partage son capital entre un compte épargne avec un taux de rendement fixe et un investissement plus risqué, pour lequel il passe par un intermédiaire telle qu'une banque. L'intermédiaire (le suiveur) (i) achète des parts pour maximiser son revenu, tout en respectant la contrainte du budget du meneur (contrainte de sac à dos) et (ii) obtient un retour de ses propres investissements. Des applications semblables peuvent être trouvées dans le domaine de la gestion de revenus (Talluri et Van Ryzin (2004)).

Le SDB peut être formulé sous forme d'un problème bi-niveaux linéaire en nombres entiers mixtes. Soit un ensemble de n objets tel qu'à chaque objet j est associé un poids a_j, un

coût c_j pour le suiveur et un coût d_j pour le meneur. Dans le problème SDB, le meneur contrôle la capacité du sac définie par la variable $x \in X = \Re$ ($n^1 = 1$) pour le meneur. Un coût t est associé à la capacité du sac. Le vecteur des variables binaires $y \in Y = \{0,1\}^{n^2}$ représente la sélection d'un sous-ensemble d'objets à mettre dans le sac par le suiveur. Nous considérons par la suite $n = n^2$. Le problème SDB peut être formulé comme suit :

$$\text{(SDB)} \begin{cases} \underset{x}{Max}\, f^1(x,y) = tx + dy \\ \quad s.c. \\ \quad\quad \underline{b} \le x \le \overline{b} \\ \quad\quad \underset{x}{Max}\, f^2(y) = cy \\ \quad\quad\quad s.c. \\ \quad\quad\quad\quad ay \le x \\ \quad\quad\quad\quad y \in \{0,1\}^n \end{cases}$$

où a, c et d sont des vecteurs de dimension n. Nous supposons que les coefficients des vecteurs a et c sont des entiers positifs, que d est un vecteur de coefficients entiers et que t est un réel. Les constantes entières et positives $\underline{b}$ et $\overline{b}$ représentent respectivement la borne inférieure et supérieure de la capacité du sac. La fonction objectif du meneur est notée $f^1(x,y): \Re \times \{0,1\}^n \to \Re$ et celle du suiveur $f^2(y): \{0,1\}^n \to \Re$.

Le problème SDB fait partie des problèmes NP-Difficiles (voir par exemple Dempe et Richter (2000)). En effet, si $\underline{b} = \overline{b}$ alors le problème SDB est un problème de sac à dos classique.

A notre connaissance, dans la littérature, le seul algorithme proposé pour résoudre le SDB est celui proposé par Dempe et Richter (2000). Il est basé sur un principe énumératif avec des règles de dominance permettant d'éliminer une partie des solutions. Une solution est dominée s'il existe une autre solution qui consomme moins de ressources et qui engendre plus de profit. Avant de présenter les règles de dominances utilisées par Dempe et Richter (2000), nous introduisons les définitions suivantes. Soient $M^0 = \{(0,0,\ldots,0)^T\}$ et e_k le $k^{ème}$ vecteur unitaire. Si M^{k-1} est l'ensemble des solutions réalisables et non dominées à l'étape k-1, alors à l'étape k, l'ensemble des solutions réalisables et non dominées M^k est inclus dans l'ensemble $M^{k-1} \cup \{x + e_k, \text{avec } x \in M^{k-1}\}$.

A chaque étape k, ils appliquent les règles de dominances suivantes sur l'ensemble $M^{k-1} \cup \{x + e_k$, avec $x \in M^{k-1}\}$ pour obtenir l'ensemble M^k (dans le cas optimiste) :

R1) Eliminer $\bar{x}$ de M^k si $a\bar{x} > b$.

R2) Soient $\bar{x}, \hat{x} \in M^k$.

Si $c\bar{x} < c\hat{x}$ et $a\bar{x} \geq a\hat{x}$ alors supprimer $\bar{x}$ de M^k.

Si $c\bar{x} = c\hat{x}$, $a\bar{x} \geq a\hat{x}$ et $d\bar{x} \leq d\hat{x}$ alors supprimer $\bar{x}$ de M^k.

La règle R1 élimine $\bar{x}$ si elle n'est pas réalisable. La règle R2 élimine $\bar{x}$ s'il existe une autre solution $\hat{x}$ qui la domine. Autrement dit, si le profit du suiveur associé à la solution $\hat{x}$ est meilleur que celui de $\bar{x}$ avec une capacité résiduelle du sac associée plus grande pour la solution $\hat{x}$. Dans le cas où les fonctions objectif du suiveur admettent la même valeur et la capacité résiduelle du sac associée à la solution $\hat{x}$ est plus grande que celle de $\bar{x}$, la solution $\hat{x}$ domine la solution $\bar{x}$ si le profit du meneur associé à la solution $\hat{x}$ est plus grand que celui associé à la solution $\bar{x}$.

L'algorithme de Dempe et Richter (2000) consiste à appliquer les règles d'élimination R1 et R2 pour générer une solution forte pour le SDB. L'application des règles de dominances pour le cas optimiste permet de définir l'ensemble de solutions réalisables non-dominées $M^n = \{x^1, ..., x^q\}$. Dans le cas où $t \leq 0$, une solution optimale (x^*, y^*) vérifie $dx^* + tax^* = \max_{1 \leq i \leq q} \{dx^i + tax^i\}$, avec $y^* = ax^*$.

Dans le cas pessimiste, la règle R2 doit être remplacée par la règle R'2:

R'2) Soient $\bar{x}, \hat{x} \in M^k$.

Si $c\bar{x} < c\hat{x}$ et $a\bar{x} \geq a\hat{x}$ alors supprimer $\bar{x}$ de M^k.

Si $c\bar{x} = c\hat{x}$, $a\bar{x} \geq a\hat{x}$ et $d\bar{x} \geq d\hat{x}$ alors supprimer $\bar{x}$ de M^k.

Notons que dans le cas où $t > 0$, une solution optimale est directement obtenue en posant $y^* = \bar{b}$ (voir Théorème 1 ci-après).

Dans la section suivante, nous étudions d'abord le problème d'existence d'une solution optimale, ensuite nous examinons les relations entre les données du problème et une solution optimale, si elle existe.

1.2.1 Conditions d'existence de solutions du SDB

Le problème SDB n'admet pas toujours de solution optimale. Néanmoins les caractéristiques particulières du SDB peuvent nous aider à introduire des conditions d'existence d'une solution optimale. Le théorème suivant, énoncé par Dempe et Richter (2000), donne ces conditions d'existence, dans le cas de deux approches optimiste et pessimiste.

Théorème 1 : Conditions nécessaires et suffisantes d'existence d'une solution optimale

- Si le coût associé à la capacité du sac pour le meneur est négatif ou nul, c'est-à-dire $t \leq 0$, alors il existe une solution forte et une solution faible du problème SDB.
- Sinon, deux cas sont possibles :
  - Soit il existe des solutions optimales $\{(\bar{b}, y^*)\}$ respectant les deux approches (optimiste et pessimiste) pour un certain $y^* \in R(\bar{b})$, où $R(\bar{b})$ est l'ensemble des solutions du problème du sac à dos pour $x = \bar{b}$. L'ensemble des solutions optimales est alors défini par $\{(\bar{b}, y^*) \mid y^* \in R(\bar{b})\}$.
  - Soit le problème SDB n'admet pas de solution optimale.

Nous donnons ci-dessous un exemple de problème SDB qui n'admet pas de solution optimale lorsque t est strictement positif.

Exemple 2 : Soit

$$(\mathrm{SDB}_1)\begin{cases} \underset{y,x}{Max}\ f^1(x,y) = 5y_1 + y_2 + y_3 + y_4 + x \\ \quad s.c.\ \ 1 \leq x \leq 4 \\ \quad\quad \underset{x}{Max}\ f^2(x) = 4y_1 + 5y_2 + 10y_3 + 15y_4 \\ \quad\quad\quad s.c.\ \ y_1 + 2y_2 + 3y_3 + 4y_4 \leq x \\ \quad\quad\quad\quad y \in \{0,1\}^4 \end{cases}$$

Nous remarquons que $f^1(x, y)$ est une fonction linéaire par morceaux et discontinue par rapport à la variable x. En effet, pour une décision du meneur x dans l'intervalle [1,2[, le suiveur a toujours la même réaction $y = (1,0,0,0)$, avec $dy = 5 + x$. La valeur de la fonction objectif du meneur sur la région induite, pour $x \in [1,2[$, est donnée par $f^1(x,y) = 5 + x$. De la même manière pour $x \in [2,3[$ et $[3,4]$ nous obtenons (voir Figure 1.4)

$$f^1(x,y)=\begin{cases}5+x & \text{pour } x\in[1,2[\\ 1+x & \text{pour } x\in[2,4]\end{cases}$$

Bien que la fonction objectif du meneur $f^1(x, y)$ soit bornée par la valeur $7=\lim_{\substack{<\\ x\to 2}} f(x,y)$, elle n'admet pas de maximum.

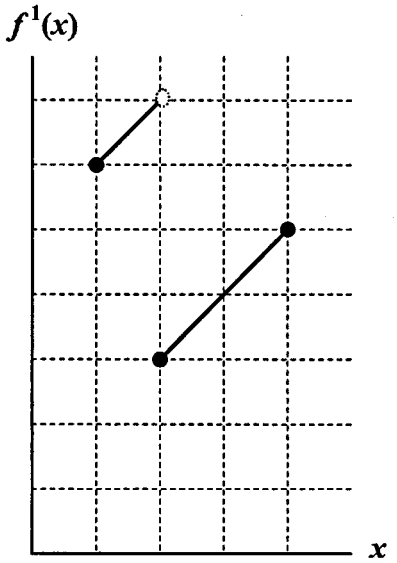


Figure 1.4: Problème d'existence d'une solution optimale

Proposition 1 : Supposons que la borne inférieure $\underline{b}=0$ et que le coefficient de capacité du sac soit strictement négatif, $t < 0$. Si la valeur absolue de t est supérieure à la valeur de tous les ratios (profit associé au meneur / poids de l'objet), alors la capacité sélectionnée par le meneur est nulle et le suiveur ne sélectionne aucun objet. Plus formellement, si $|t| > \max_{1\le i\le n}(d_j/a_j)$ alors la solution $x^* = 0$ et $y^* = 0$ est une solution optimale (forte et faible) pour le problème SDB.

Preuve : Soit (x, y) une solution réalisable du SDB. En multipliant la contrainte $ay \le x$ par $t < 0$ et en lui ajoutant dy, il suit que $(ta+d)\, y \ge tx + dy = f^1(x, y)$. Par hypothèse $ta_j+d_j < 0$ pour $j = 1,.., n$, or $y\in\{0,1\}^n$, ainsi $(ta+d)\, y \le 0$. Donc pour toute solution réalisable (x, y), nous avons $f^1(x, y) \le 0$, puisque $(ta+d)\, y \ge tx + dy = f^1(x, y)$. De plus, $x = 0$ et $y = 0$ est une solution réalisable pour le problème SDB, donc $(x^*, y^*) = (0,0)$ est une solution optimale pour le SDB. □

Proposition 2 : Soient le coût associé à la capacité du sac $t \geq 0$ et α une constante positive. Si $d = \alpha \times c$ alors il existe des solutions optimales respectant les deux approches (optimiste et pessimiste) et appartenant à l'ensemble $\{(y^*, \bar{b}) \mid y^* \in R(\bar{b})\}$.

Preuve : Supposons par l'absurde qu'il existe une solution (x',y') avec $y' \notin R(\bar{b})$ et $f^1(x',y') = tx' + dy' > f^1(\bar{b},x^*) = t\bar{b} + dy^*$. Par conséquent $dy' > dy^*$ car $tx' \leq t\bar{b}$. Or $d = \alpha c$, ainsi $\alpha c y' > \alpha c y^*$, ce qui implique que $f^2(y') > f^2(y^*)$. Ce dernier résultat est contradictoire avec l'hypothèse $y^* \in R(\bar{b})$ et $y' \notin R(\bar{b})$, car $f^2(y^*) > f^2(y) \; \forall \, y \in R(x)$ et $x \in [\underline{b}, \bar{b}]$. □

Proposition 3 : Soit SDB (resp. SDB_d) le problème de sac à dos bi-niveaux continu (resp. discret au premier niveau meneur), c'est-à-dire x continu (resp. entier). Si $t \leq 0$ alors toute solution optimale (x^*, y^*) du SDB_d est optimale pour le SDB.

Preuve : Soit $R(SDB_d)$ l'ensemble des solutions réalisables du SDB avec x entier et R(SDB) l'ensemble des solutions réalisables du SDB continu au premier niveau (meneur). Il est évident que $R(SDB_d) \subset R(SDB)$. De plus, il existe une solution optimale de SDB (x^*, y^*) qui appartient à SDB_d, sachant qu'à l'optimum x^* prend une valeur entière, donc (x^*, y^*) est aussi optimale pour le SDB_d. □

Proposition 4 : Si $t > 0$ et s'il existe une solution optimale pour le problème SDB alors elle est également optimale pour le SDB_d.

Preuve : La preuve est semblable à celle de la condition nécessaire dans la Proposition 3. □

La Proposition 3 nous montre que la résolution du SDB est équivalente à la résolution du SDB_d dans le cas où $t \leq 0$. D'après le Théorème 1, nous savons qu'une solution optimale du SDB, si elle existe lorsque $t > 0$, est $(y^*, \bar{b})$ avec $y^* \in R(\bar{b})$. Notons qu'il existe toujours une solution optimale pour le problème SDB_d. Nous déduisons alors que si le problème SDB admet une solution optimale, nous pouvons le reformuler comme une série de problèmes de sac à dos avec variables binaires, associés aux valeurs prises par la variable du meneur x. Les deux propositions précédentes nous permettent d'utiliser la programmation dynamique comme base d'un algorithme de résolution pour le SDB. Cet algorithme est présenté dans la section suivante.

Notons que la programmation dynamique fait partie des méthodes de résolution les plus efficaces pour les problèmes de sac à dos.

1.2.2 Programmation Dynamique pour le SDB

Le problème de sac à dos est un problème d'optimisation combinatoire très largement étudié depuis plusieurs décennies. Ce problème peut être résolu efficacement par la programmation dynamique (voir Toth (1980), Martello et Toth (1990), Kellerer, Pferschy et Pisinger (2004)). Dans cette section, nous rappelons d'abord les relations de récurrences pour l'algorithme de la programmation dynamique appliqué au sac à dos classique, ensuite nous considérons leur extension au cas des problèmes bi-niveaux avec la structure de sac à dos.

La méthode de programmation dynamique consiste à plonger le problème initial dans une famille de sous-problèmes de même nature :

$$f_k(x) = \max\left\{\sum_{j=1}^{k} c_j y_j : \sum_{j=1}^{k} a_j y_j \le x, y \in \{0,1\}^k\right\},$$

puis à relier par une relation de récurrences les solutions optimales de ces sous-problèmes

$$f_k(x) = \begin{cases} f_{k-1}(x) & \text{pour } x = 0,..,a_{k-1} \\ \max\left(f_{k-1}(x), f_{k-1}(x-a_k)+c_k\right) & \text{pour } x = a_k,....,b \end{cases}$$

L'application directe de la programmation dynamique sur le problème du suiveur n'est pas suffisante pour la résolution du SDB. En effet, il est nécessaire d'y intégrer des procédures de dominance liées aux valeurs des variables de décision et à la fonction objectif du meneur. Nous décrivons ci-dessous le principe de l'algorithme proposé. Deux tables sont considérées : la première permet de mémoriser les valeurs optimales des sous-problèmes associés au suiveur $f_k^2(x)$ et la deuxième contient les valeurs optimales des sous-problèmes associés au meneur $\tilde{f}_k^1(x)$ avec

$$\tilde{f}_k^1(x) = \max\left\{\sum_{j=1}^{k} d_j y_j : \sum_{j=1}^{k} a_j y_j \le x, y \in \{0,1\}^k \text{ et } y \in \mathrm{R}(x)\right\},$$

$$f_k^2(x) = \max\left\{\sum_{j=1}^{k} c_j y_j : \sum_{j=1}^{k} a_j y_j \le x, y \in \{0,1\}^k\right\}.$$

L'algorithme est composé de deux phases : une phase de descente et une phase de remontée.

1.2.2.1 Procédure de descente

La phase de descente consiste à appliquer la programmation dynamique sur le SDB pour définir les réactions associées à chaque action possible du meneur. La résolution du premier sous-problème associé au premier objet est triviale : le suiveur sélectionne le premier objet si la capacité du sac est suffisante :

$$f_1^2(x)=\begin{cases}0 & \text{pour} \quad x=0,...,a_1-1\\ c_1 & \text{pour} \quad x=a_1,...,\bar{b}\end{cases} \qquad \tilde{f}_1^1(x)=\begin{cases}0 & \text{pour} \quad x=0,...,a_1-1\\ d_1 & \text{pour} \quad x=a_1,...,\bar{b}\end{cases}$$

Dans les autres sous-problèmes, nous appliquons les relations de récurrences de la programmation dynamique de la façon suivante. Dans le cas où la décision du suiveur peut entraîner un scénario différent, il favorise son profit sans considération du profit du meneur. D'une manière formelle, le suiveur prend ses décisions selon la relation de récurrences suivante :

$$f_k^2(x)=\begin{cases}f_{k-1}^2(x) & \text{pour} \quad x=0,..,a_k-1\\ \max\left(f_{k-1}^2(x), f_{k-1}^2(x-a_k)+c_k\right) & \text{pour} \quad x=a_k,....,\bar{b}\end{cases}$$

Dans ce cas, il suffit que le meneur suive la trace du suiveur. Autrement dit,

$$\tilde{f}_k^1(x)=\begin{cases}\tilde{f}_{k-1}^1(x) & \text{si} \quad f_k^2(x)=f_{k-1}^2(x)\\ \tilde{f}_{k-1}^1(x-a_k)+d_k & \text{si} \quad f_k^2(x)=f_{k-1}^2(x-a_k)+c_k\end{cases}$$

Si les décisions sont équivalentes par rapport au profit du suiveur pour une valeur de x, alors la décision du suiveur favorise le meneur dans le cas optimiste. C'est-à-dire que si

$$f_k^2(x) = f_{k-1}^2(x) = f_{k-1}^2(x-a_k)+c_k,$$

alors la valeur optimale du sous-problème associée au meneur doit être prise selon la relation de récurrences suivante :

$$\tilde{f}_k^1(x)=\max\left(\tilde{f}_{k-1}^1(x), \tilde{f}_{k-1}^1(x-a_k)+d_k\right).$$

Notons que dans ce cas $x \geq a_k$, sinon le suiveur ne peut pas avoir plusieurs choix car il ne peut pas sélectionner le $k^{ème}$ objet. Dans le cas pessimiste, le choix doit être défavorable au meneur :

$$\tilde{f}_k^1(x)=\min\left(\tilde{f}_{k-1}^1(x), \tilde{f}_{k-1}^1(x-a_k)+d_k\right).$$

L'Algorithme 1.1 décrit la procédure de descente.

Algorithme 1.1 : Procédure de descente

// Initialisation

pour $x = 0$ **à** a_1-1 **faire** $f_1^2(x) = 0$; $\tilde{f}_1^1(x) = 0$; **fin pour**

pour $x = a_1$ **à** $\bar{b}$ **faire** $f_1^2(x) = c_1$; $\tilde{f}_1^1(x) = d_1$; **fin pour**

// Itérations

pour $k = 2$ **à** n **faire**

pour $x = 0$ **à** $\bar{b}$ **faire**

si $x < a_k$ **alors** $f_k^2(x) = f_{k-1}^2(x)$; $\tilde{f}_k^1(x) = \tilde{f}_{k-1}^1(x)$; **fin si**

si $x \geq a_k$ **alors**

$$f_k^2(x) = \max\left(f_{k-1}^2(x), f_{k-1}^2(x - a_k) + c_k\right) \quad \text{(4-a)}$$

si $f_{k-1}^2(x) \neq f_{k-1}^2(x - a_k) + c_k$ **alors**

$$\tilde{f}_k^1(x) = \begin{cases} \tilde{f}_{k-1}^1(x) & si \quad f_k^2(x) = f_{k-1}^2(x) \\ \tilde{f}_{k-1}^1(x - a_k) + d_k & si \quad f_k^2(x) = f_{k-1}^2(x - a_k) + c_k \end{cases} \quad \text{(4-b)}$$

fin si

si $f_{k-1}^2(x) = f_{k-1}^2(x - a_k) + c_k$ **alors**

$$\tilde{f}_k^1(x) = \max\left(\tilde{f}_{k-1}^1(x), \tilde{f}_{k-1}^1(x - a_k) + d_k\right) \quad \textit{(Cas optimiste)} \text{ (4-c)}$$

$$\tilde{f}_k^1(x) = \min\left(\tilde{f}_{k-1}^1(x), \tilde{f}_{k-1}^1(x - a_k) + d_k\right) \quad \textit{(Cas pessimiste)} \text{(4-d)}$$

fin si

fin si

fin pour

fin pour

1.2.2.2 Procédure de remontée

La phase de descente permet de générer la valeur optimale du SDB, le but de la phase de remontée est de générer les valeurs optimales des variables de décision. À la fin de la construction des deux tables par la procédure de descente, la $n^{ème}$ colonne de la table du meneur nous permet d'obtenir les valeurs optimales de la fonction objectif du meneur associées à chaque valeur de x dans $[\underline{b}, \bar{b}]$.

Proposition 5 : Soit (x^*, y^*) une solution optimale de SDB_d.

- Si $t \leq 0$ alors $\tilde{f}_n^1(x^*) + tx^* = \underset{\underline{b} \leq x \leq \bar{b}}{Max}\left\{\tilde{f}_n^1(x) + tx\right\}$
- Si $t > 0$ alors
  - Si $\underset{\underline{b} \leq x < \bar{b}}{Max}\left\{\tilde{f}_n^1(x) + t(x+1)\right\} \leq \tilde{f}_n^1(\bar{b}) + t\bar{b}$ alors $(\bar{b}, y^*)$ est une solution optimale du SDB avec $y^* \in R(x^*)$ et $x^* = \bar{b}$.
  - Sinon le problème SDB n'admet pas de solution optimale.

Preuve : Nous considérons deux cas.

Cas 1 : $t \leq 0$. D'après la Proposition 3, nous savons que x^* appartient à $\{\underline{b}, \underline{b}+1,..,\overline{b}\}$. Evidemment, la valeur optimale de x est celle associée à la valeur maximale de $\tilde{f}_n^1(x)+tx$. Notons que $\tilde{f}_n^1(x)+tx = f^1(x,y)$ avec $x \in \{\underline{b}, \underline{b}+1,..,\overline{b}\}$ et $y \in R(x)$.

Cas 2 : $t > 0$. Nous savons que $\forall(x, y) \in R(SDB_d)$, avec $x \in \{\underline{b}, \underline{b}+1,\ldots, \overline{b}-1\}$, nous avons $y \in R(x + \varepsilon)$ avec $\varepsilon \in [0,1]$. La valeur de la fonction objectif du meneur du SDB associée est $f^1(x+\varepsilon, y) = \tilde{f}_n^1(x)+t(x+\varepsilon)$. Nous ne prendrons pas en compte le cas où $x = \overline{b}$ car la valeur de ε associée est nulle. Nous pouvons remarquer que la valeur optimale locale pour $x \in \{\underline{b}, \underline{b}+1,..,\overline{b}-1\}$ n'existe pas. Nous pouvons l'estimer par $\lim_{\varepsilon \to 1}(f_n^1(x)+t(x+\varepsilon))$. D'autre part, nous savons que la solution optimale locale pour $x = \overline{b}$ existe. Pour vérifier si cette solution est optimale globalement, il suffit de vérifier si elle est meilleure que toutes les approximations des solutions optimales locales pour $x \in \{\underline{b}, \underline{b}+1,..,\overline{b}-1\}$. S'il existe une approximation qui la domine alors il n'existe pas d'optimum global pour le SDB. □

Notons que s'il existe plusieurs valeurs optimales pour x^*, nous appliquons la procédure de remontée pour chacune d'elles pour récupérer toutes les solutions optimales. Nous remontons à partir de la valeur x^* et nous appliquons les instructions de la programmation dynamique sur le problème du suiveur en respectant la relation de récurrences suivante :

$$\text{Pour } k = n \text{ à } 2 \text{ faire } y_k^* = \begin{cases} 0 & \text{si } f_k^2(x) = f_{k-1}^2(x) \\ 1 & \text{si } f_k^2(x) = f_{k-1}^2(x-a_k)+c_k \end{cases}$$

A chaque étape k de la procédure de remontée, si le suiveur a des alternatives équivalentes (i.e., $f_{k-1}^2(x) = f_{k-1}^2(x-a_k)+c_k$), nous appliquons la relation de récurrences suivante liée au problème du meneur pour déterminer la valeur optimale de y_k^*:

$$y_k^* = \begin{cases} 0 & \text{si } \tilde{f}_k^1(x) = \tilde{f}_{k-1}^1(x) \\ 1 & \text{si } \tilde{f}_k^1(x) = \tilde{f}_{k-1}^1(x-a_k)+d_k \end{cases}$$

Si les deux choix sont équivalents pour le meneur alors la variable y_k^* peut prendre les deux valeurs. Notons qu'à chaque itération, si $y_k^* = 1$, la valeur de x est mise à jour par $x - a_k$. La valeur optimale de la dernière variable y_1^*, qui ne dépend pas des relations de récurrences, est déterminée par :

$$y_1^* = \begin{cases} 0 & \text{si} \quad f_1^2(x) = 0 \\ 1 & \text{si} \quad f_1^2(x) = c_1 \end{cases}$$

La procédure de remontée est définie par l'algorithme suivant.

Algorithme 1.2 : Procédure de remontée

$x = x^*$;
pour $k = n$ à 2 **faire**
  si $f_{k-1}^2(x) \neq f_{k-1}^2(x - a_k) + c_k$ **alors**
    si $f_k^2(x) = f_{k-1}^2(x)$ **alors** $y_k^* = 0$; **fin si**
    si $f_k^2(x) = f_{k-1}^2(x - a_k) + c_k$ **alors** $y_k^* = 1$; $x = x - a_k$; **fin si**
  sinon
    si $\tilde{f}_k^1(x) = \tilde{f}_{k-1}^1(x)$ **alors** $y_k^* = 0$; **fin si**
    si $\tilde{f}_k^1(x) = \tilde{f}_{k-1}^1(x - a_k) + d_k$ **alors** $y_k^* = 1$; $x = x - a_k$; **fin si**
  fin si
fin pour
si $f_1^2(x) = 0$ **alors** $y_1^* = 0$ **sinon** $y_1^* = 1$; **fin si**

Comme l'algorithme basé sur la programmation dynamique pour le problème du sac à dos classique, l'algorithme présenté ci-dessus pour le SDB a une complexité en $O(n\bar{b})$. Nous illustrons le déroulement de cet algorithme dans la section suivante.

1.2.2.3 Illustration

Soit le problème SDB_2 comprenant 4 variables binaires pour le suiveur :

$$(SDB_2) \begin{cases} Max\, f^1(x,y) = -2x + 3y_1 + 5y_2 + y_3 + y_4 \\ \quad s.c. \\ \quad\quad 0 \le x \le 6 \\ \quad\quad Max\, f^1(y) = y_1 + y_2 + y_3 + y_4 \\ \quad\quad\quad s.c. \\ \quad\quad\quad\quad 5y_1 + 3y_2 + 2y_3 + y_4 \le x \\ \quad\quad\quad\quad y \in \{0,1\}^4 \end{cases}$$

Phase de descente : Le Tableau 1.1 donne les valeurs optimales des sous-problèmes associés au meneur et au suiveur lors de la phase de descente. Une cellule (x, k) contient les valeurs $\left(f_k^2(x), \tilde{f}_k^1(x)\right)$. Le suiveur sélectionne le premier objet (k = 1) uniquement si la capacité du

sac est suffisante ($x \geq 5$), c'est à dire $f_1^2(x)$ et $\tilde{f}_1^1(x)$ sont nulles pour $x < a_1 = 5$ et $\left(f_1^2(x), \tilde{f}_1^1(x)\right) = (c_1, d_1) = (1,3)$ pour $x \geq 5$.

k / x	1		2		3		4	
	$f_1^2(x)$	$\tilde{f}_1^1(x)$	$f_2^2(x)$	$\tilde{f}_2^1(x)$	$f_3^2(x)$	$\tilde{f}_3^1(x)$	$f_4^2(x)$	$\tilde{f}_4^1(x)$
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	9
2	0	0	0	0	1	1	1	9
3	0	0	1	5	1	1	2	10
4	0	0	1	5	1	1	2	10
5	1	3	1	5	2	6	2	10
6	1	3	1	5	2	6	3	15

Tableau 1.1: Valeurs de $f_k^2(x)$ et $\tilde{f}_k^1(x)$

Le deuxième objet ($k = 2$) peut être sélectionné si la capacité du sac est suffisante. Donc, pour $x < a_2 = 3$ il n'y a aucun changement dans le sac, $\left(f_2^2(x), \tilde{f}_2^1(x)\right) = \left(f_1^2(x), \tilde{f}_1^1(x)\right)$. Pour $x \geq 3$, la relation de récurrences (4-a) définit : $f_2^2(x) = \max\left(f_1^2(x), f_1^2(x-a_2)+c_2\right)$ pour $x = a_2, \ldots, \bar{b}$. Ainsi $f_2^2(3) = \max\left(f_1^2(3), f_1^2(0)+1\right) = \max\{0,1\} = 1$. Comme la valeur de $f_2^2(3)$ est obtenue par $f_1^2(3-a_2)+c_2$, le meneur doit prendre sa valeur à partir de $\tilde{f}_2^1(3) = \tilde{f}_1^1(3-a_2)+d_2 = \tilde{f}_1^1(0) + 5 = 5$ (Règle (4-b)).

Le même principe peut être appliqué pour $k = 3$ et 4. Le cas où les décisions du suiveur sont équivalentes intervient uniquement pour $k = 4$ et $x = 5$. En effet, $f_4^2(5) = \max\left(f_3^2(5), f_3^2(4)+1\right) = \max(2, 2) = 2$. Dans ce cas, le suiveur favorise l'objectif du meneur dans le cas optimiste en appliquant la Règle (4-c) $\tilde{f}_4^1(5) = \max\left(\tilde{f}_4^1(5), \tilde{f}_4^1(5-a_4)+d_4\right) = \max(6,1+9) = 10$.

A la fin de cette phase, nous définissons la valeur optimale de x^* par $f_{opt}^1 = \tilde{f}_n^1(x^*)+tx^* = \underset{\underline{b} \leq x \leq \bar{b}}{Max}\left\{\tilde{f}_n^1(x)+tx\right\} = Max$ {0-0, 9-2, 9-4, 10-6, 10-8, 10-10, 15-12} = 7. $x^* = 1$ et $\left(f_{opt}^2, f_{opt}^1\right) = (1,7)$.

Phase de remontée : Nous ne considérons dans cette phase que la partie du Tableau 1.1 associée à $x \leq 1$ ($x^* = 1$) (Tableau 1.2). Comme $\left(f_4^2(1) \neq f_3^2(1)\right)$ et $\left(f_4^2(1) = f_4^2(0)+c_4\right)$, la variable y_4^* est fixée à 1. De même, il suit que $y_3^* = 0$, $y_2^* = 0$ et $y_1^* = 0$. Comme $f_1^2(0) = 0$, y_1^* est fixé à 0. A la fin de cette procédure, une solution optimale forte est trouvée, $y^* = (0,0,0,1)$ et $x^* = 1$.

k / x	1		2		3		4	
	$f_1^2(x)$	$\tilde{f}_1^1(x)$	$f_2^2(x)$	$\tilde{f}_2^1(x)$	$f_3^2(x)$	$\tilde{f}_3^1(x)$	$f_4^2(x)$	$\tilde{f}_4^1(x)$
0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	1	9

Tableau 1.2 : Phase de remontée

Nous remarquons que l'obtention d'une solution optimale nécessite la mémorisation des deux tableaux. Ceux-ci peuvent occuper un espace mémoire important. Pour surmonter ce problème, nous avons intégré les règles de dominances ((Martello et Toth (1990), Kellerer, Pferschy et Pisinger (2004)) dans l'algorithme de programmation dynamique (**PD**). Rappelons que ces règles ont été adoptées par Dempe et Richter (2000) pour le SDB. L'intégration de ces règles de dominances dans la programmation dynamique nous conduisent vers un algorithme de programmation dynamique creuse pour le SDB (**PDD**).

1.2.2.4 Programmation dynamique creuse pour le SDB

Un ensemble de sous-problèmes n'aboutissant pas à la solution optimale (voir Martello et Toth (1990)) peut être éliminé de façon à réduire l'espace de recherche et ainsi l'espace mémoire requis pour l'exécution de l'algorithme. Ces règles sont les suivantes :

- Soit k une étape de l'algorithme *PD*, la solution partielle $(\bar{x}, \bar{y})$ domine la solution partielle $(\hat{x}, \hat{y})$ si

$$(f_k^2(\bar{x}) > f_k^2(\hat{x}) \text{ et } \bar{x} \leq \hat{x}) \text{ ou } (f_k^2(\bar{x}) = f_k^2(\hat{x}) \text{ et } \bar{x} \leq \hat{x} \text{ et } f_k^2(\bar{x}) \geq f_k^2(\hat{x})).$$

Ces règles éliminent le sous-problème associé à $\hat{x}$ s'il existe une autre solution $\bar{x}$ qui la domine. Dans ce cas, le profit du suiveur dans la solution $\bar{x}$ est meilleur que celui de $\hat{x}$ avec une capacité du sac plus petite pour $\bar{x}$. Si les profits du suiveur sont égaux, si la capacité du sac de $\bar{x}$ est plus petite que celle de $\hat{x}$ et si le profit du meneur de $\bar{x}$ est plus grand que celui de $\hat{x}$, alors la solution $\bar{x}$ domine la solution $\hat{x}$.

Dans la section suivante, nous présentons les résultats numériques obtenus lors de la validation de notre méthode pour résoudre le SDB.

1.2.3 Résultats numériques

Dans cette section, nous comparons la performance du nouvel algorithme de programmation dynamique (*PD*) proposé à celle de l'algorithme de Dempe et Richter (2000) (*D&R*) et à celui de Bard et Moore (1992) (*B&M*) (voir Section 1.1). Ce dernier est un

algorithme de *Branch-and-Bound* développé pour la résolution des problèmes bi-niveaux avec variables binaires. Son utilisation pour le SDB nécessite une décomposition binaire de x (i.e., $x = \sum_{i=1}^{q} z_i 2^{i-1}$, où $z \in \{0,1\}^q$ avec $q = \lfloor \log(\bar{b})/\log(2) \rfloor + 1$). Les trois algorithmes sont codés en C++ et les sous-problèmes en variables entières de l'algorithme de *B&M* sont résolus à l'aide du logiciel CPLEX (2008). Les tests sont effectués sur un *Pentium IV EE* avec un processeur de capacité 3,4 Ghz et 4 Gb de RAM.

Les instances utilisées pour cette étude expérimentale sont créées par le générateur de Martello, Pisinger et Toth (1999) pour le problème du sac dos classique. Plusieurs types d'instances sont générés selon la relation entre les coefficients du problème. Nous distinguons trois types d'instances : celles avec des coefficients non corrélés (*NC*), des coefficients corrélés (*C*) et des coefficients fortement corrélés (*FC*) (Martello, Pisinger et Toth (1999)). Les données du problème du meneur sont aussi générées aléatoirement. La taille des coefficients est de 100 dans un premier temps puis de 1000. Rappelons que la taille d'un coefficient est la valeur maximale pouvant être prise par ce coefficient.

Le Tableau 1.3 présente les résultats pour des instances qui comprennent de 10 à 250 objets. Chaque ligne donne la moyenne du temps d'exécution (en secondes) pour 10 instances de chaque type. Notons que l'exécution d'un algorithme est stoppée après une heure si la solution du problème n'a pas pu être obtenue.

	Taille des coefficients 100								
	NC			C			FC		
n	*B&M*	*D&R*	*PD*	*B&M*	*D&R*	*PD*	*B&M*	*D&R*	*PD*
10	0.39	< 0.01	< 0.01	0.51	< 0.01	< 0.01	0.16	< 0.01	< 0.01
25	3.29	< 0.01	< 0.01	6.03	0.039	< 0.01	0.94	0.11	< 0.01
50	15.25	0.08	< 0.01	30.00	0.908	< 0.01	39.48	2.01	< 0.01
75	23.00	0.44	0.01	52.76	3.625	0.01	584.62	7.91	0.01
100	63.21	1.65	0.01	140.76	13.343	0.01	3223.70	18.81	0.02
250	191.83	85.24	0.10	375.28	331.339	0.10	-------	394.52	0.11

Tableau 1.3 : Temps d'exécution des algorithmes en secondes pour la taille 100

L'algorithme *B&M* est clairement le moins rapide des trois algorithmes. Ceci est dû à la décomposition binaire de la variable x. En effet, la contrainte de sac à dos $ay \leq x$ est remplacée par la contrainte $-\sum_{i=1}^{q} 2^{i-1} z_i + ay \leq 0$. Ainsi la résolution du SDB comprend un nombre important de variables z_i non nulles puisque leurs coefficients dans la contrainte sont négatifs et sont nuls dans la fonction objectif. Or, un grand nombre de variables du meneur z_i égales à 1 entraîne la création d'un nombre important de nœuds par l'algorithme *B&M*, et

donc un nombre important d'appels à un algorithme exponentiel pour résoudre un programme linéaire avec variables binaires.

L'algorithme *PD* est le plus performant parmi les trois algorithmes pour les trois types d'instances. L'algorithme *D&R* reste le deuxième meilleur algorithme par rapport au temps d'exécution. En outre, plus la corrélation entre les coefficients s'intensifie, plus la différence entre les algorithmes augmente, en faveur de l'algorithme *PD*.

De façon générale, le degré de corrélation influence logiquement le temps d'exécution pour les trois algorithmes. Néanmoins cette augmentation reste très faible pour l'algorithme *PD* pour ces tailles d'instances. Pour l'algorithme *D&R*, cette augmentation provient du nombre de solutions non dominées qui sont de même qualité et qui ne sont donc pas éliminées. Si les coefficients sont fortement corrélés, l'algorithme *D&R* génère presque toutes les solutions réalisables du problème.

	Taille des coefficients 1000								
	NC			C			FC		
n	B&M	D&R	PD	B&M	D&R	PD	B&M	D&R	PD
10	1.94	<0.01	< 0.01	8.16	< 0.01	< 0.01	1.33	< 0.01	< 0.01
25	41.67	< 0.01	0.01	93.31	0.07	< 0.01	38.58	1.60	< 0.01
50	220.67	0.08	0.03	669.77	1.97	0.03	-----	90.81	0.04
75	------	0.90	0.09	------	9.69	0.09	-----	828.89	0.09
100	------	2.44	0.16	------	75.68	0.15	-----	3537.70	0.16

Tableau 1.4 : Temps d'exécution des algorithmes en secondes pour la taille 1000

Le Tableau 1.4 illustre une augmentation du temps d'exécution pour tous les algorithmes lorsque la taille des coefficients augmente (1000 au lieu de 100). Ceci provient de la grande valeur de $\bar{b}$ qui est calculée en fonction de la taille des coefficients par le générateur. L'algorithme *PD* donne des résultats relativement moins rapides à cause de sa complexité en $O(n\bar{b})$. L'algorithme *D&R* devient plus long car l'augmentation de $\bar{b}$ augmente le nombre des solutions réalisables non dominées. L'algorithme *B&M* donne de moins bons résultats car cette valeur $\bar{b}$ augmente encore le nombre de variables du meneur.

Ces expériences numériques montrent que notre nouvel algorithme est beaucoup plus rapide que les deux autres. Lorsque la taille des instances augmente, les algorithmes *D&R* et *B&M* demandent plus d'une heure d'exécution pour s'achever. Cependant, la programmation dynamique est une méthode coûteuse en termes d'espace mémoire, ce qui est un obstacle pour résoudre des instances de plus grande taille.

Le Tableau 1.5 présente les résultats obtenus en appliquant la programmation dynamique creuse avec les règles de dominances présentées dans la Section 1.2.2.4. Nous reportons les

temps d'exécution de l'algorithme *PD* et de la programmation dynamique avec dominance (*PDD*) pour des instances des trois types avec un nombre de variables au second niveau allant de 100 à 500.

	Taille des coefficients est : 1000						Taille des coefficients est : 3000					
	NC		C		FC		NC		C		FC	
n	*PD*	*PDD*	*PD*	*PDD*	*PD*	*PDD*	*PD*	*PDD*	*PD*	*PDD*	*PD*	*PDD*
100	0.16	0.01	0.16	0.04	0.18	0.23	0.51	0.01	0.50	0.04	0.55	0.66
200	0.72	0.07	0.68	0.50	0.77	1.25	2.02	0.08	2.23	0.31	2.31	3.41
300	1.54	0.24	1.62	1.29	1.71	2.95	4.77	0.27	4.60	1.14	5.00	7.95
400	2.80	0.58	2.79	2.66	3.10	5.42	9.03	0.60	8.44	2.51	9.44	15.71
500	4.53	1.12	4.33	4.52	4.70	8.68	12.91	1.38	12.25	5.15	-----	25.24

Tableau 1.5 : Temps d'exécution de *PD* et *PDD* en secondes

Nous remarquons que le temps d'exécution de l'algorithme *PDD* tend à être plus important lorsque les règles de dominances sont moins efficaces (i.e., pour les instances FC). Ceci provient du coût additionnel engendré pour la détection des éléments dominés. Par contre, le temps d'exécution de *PDD* est plus petit que celui de la programmation dynamique avec les tableaux *PD* pour les instances faiblement corrélées. L'augmentation du taux de corrélation favorise donc l'algorithme *PD* quand n augmente. Cependant, celui-ci sature plus rapidement la mémoire.

En termes d'espace mémoire, une étude comparative entre l'algorithme *PD* et l'algorithme *PDD* est présentée dans la Figure 1.5. Nous avons considéré les trois types d'instances *NC*, *C* et *FC* et pour chacune d'elles trois tailles de problème, n = 100, 150 et 200. La taille maximum des coefficients est 100. Nous rappelons que les deux algorithmes sont pseudo-polynomiaux en termes d'espace mémoire. Dans la Figure 1.5, nous présentons l'espace utilisé par les deux algorithmes, avec celui de l'algorithme *PD* à gauche et celui de *PDD* à droite. Sur l'axe des abscisses, nous présentons les résultats pour les trois types de corrélation, *NC*, *C* et *FC* pour n = 100. Nous procédons de la même manière pour les instances de taille n = 150 et n = 200. Chaque valeur correspond à la moyenne observée sur 10 instances.

La Figure 1.5 met en évidence l'impact positif de la dominance sur l'utilisation de la mémoire. L'espace consommé par l'algorithme *PD* pour une taille fixée est stable, quelle que soit la corrélation de l'instance. Par ailleurs, l'espace mémoire utilisé par l'algorithme *PDD* augmente avec le degré de corrélation des instances. Cette augmentation est justifiée par l'affaiblissement de la dominance face à la corrélation des coefficients car les sous-problèmes traités deviennent de qualité similaire.

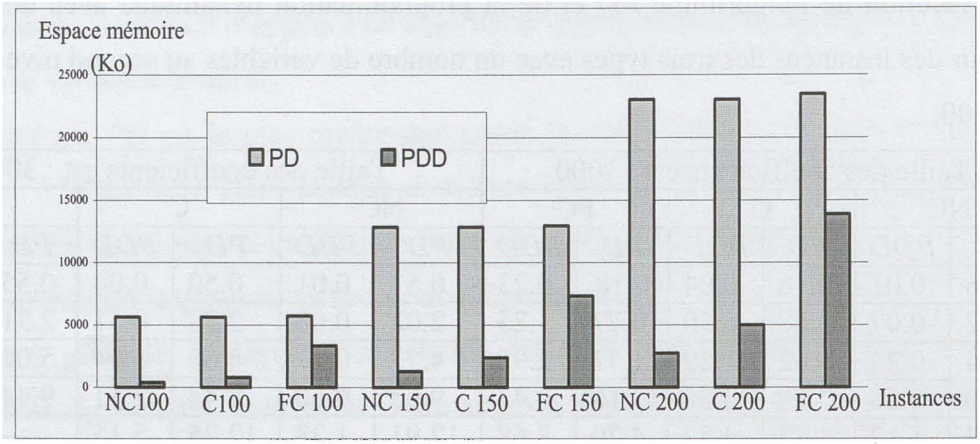


Figure 1.5 : Espace mémoire pour *PD* et *PDD*

Nous terminons par l'étude de l'impact de la valeur du coût t associée à la capacité du sac dans la fonction objectif du meneur sur la solution optimale, et plus exactement sur le nombre d'objets sélectionnés. Les instances considérées dans cette étude sont de type C, où nous utilisons la valeur de 100 comme taille pour les coefficients et $n = 100$. Les valeurs prises par t sont les entiers dans {-1, -2, -3, -4, -5}. Nous présentons les résultats dans la Figure 1.6. Pour chaque valeur de t, nous représentons à gauche le rapport du nombre d'objets utilisés dans la solution sur le nombre d'objets disponibles n et à droite le rapport de la capacité occupée dans le sac sur la capacité totale. Chaque valeur représente une moyenne sur 10 instances.

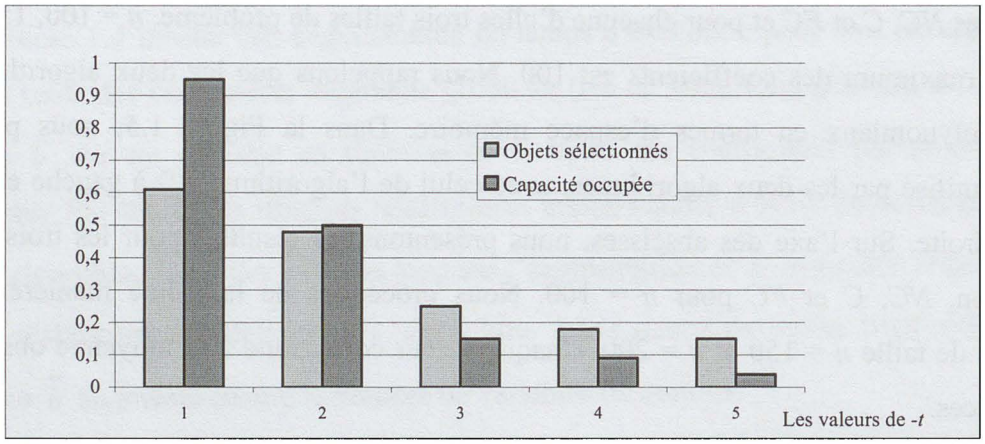


Figure 1.6 : Influence de t sur le nombre d'objets et la capacité

Nous remarquons que le nombre d'objets sélectionnés décroît avec la valeur de t. Plus précisément, lorsque la valeur absolue de t augmente, le nombre d'objets sélectionnés devient faible (de même pour la capacité occupée). Dans ce cas, la fonction objectif du meneur $f^1(x, y)$ converge vers sa borne inférieure, la valeur 0 (voir Proposition 1).

1.3 Reformulation du problème du sac à dos unidimensionnel bi-niveaux

Dans cette section, nous proposons une nouvelle méthode de résolution exacte (Mansi, Brotcorne et Hanafi (2008), Hanafi, Mansi et Brotcorne (2008)) pour le problème du sac à dos unidimensionnel bi-niveaux (**SDUB**). Pour des raisons de simplicité d'écriture, nous limitons sa présentation au cas optimiste. Le SDUB est un problème bi-niveaux caractérisé par des variables du meneur et du suiveur entières, des contraintes au premier niveau et une seule contrainte de sac à dos au second niveau. Le SDUB peut donc être formulé comme suit :

$$\text{(SDUB)} \begin{cases} \underset{x,y}{Max}\ f^1(x,y) = d^1x + d^2y \\ \quad s.c \\ \quad\quad B^1x + B^2y \le b^1 \\ \quad\quad x \in N^{n^1} \\ \quad\quad \underset{y}{Max}\ f^2(y) = cy \\ \quad\quad\quad s.c. \\ \quad\quad\quad\quad a^1x + a^2y \le b \\ \quad\quad\quad\quad y \in N^{n^2} \end{cases}$$

où a^1, a^2 et c sont des vecteurs d'entiers strictement positifs de dimension n^1, n^2 et n^2 respectivement.

La méthode de résolution proposée est composée de deux phases : la première, basée sur la programmation dynamique dense (avec tableau), permet de déterminer toutes les réactions possibles du suiveur. Celles-ci sont utilisées dans la seconde phase pour formuler le SDUB sous forme d'un programme linéaire à variables entières pouvant être résolu par un algorithme de type séparations et évaluations par exemple.

1.3.1 Phase 1 : Programmation Dynamique pour le problème du suiveur

Dans cette phase, nous considérons uniquement le problème du suiveur où nous ignorons les ressources a^1x consommées par le meneur. Plus précisément, nous considérons le problème de sac à dos (**SD**) en nombres entiers suivant :

$$\text{(SD)} \begin{cases} \underset{y}{Max}\ f^2(y) = cy \\ s.c. \\ \quad a^2y \le b \\ \quad y \in N^{n^2} \end{cases}$$

La programmation dynamique, et plus particulièrement la phase de descente, est appliquée pour obtenir l'ensemble des valeurs optimales du problème associé à des valeurs de capacité appartenant à l'intervalle [0, b]. Soient k et β des entiers avec $1 \le k \le n^2$ et $0 \le \beta \le b$, où k est l'itération de la programmation dynamique et β la ressource disponible. Le sous-problème du suiveur $f_k^2(\beta)$ associé aux k premières variables $y_1,...,y_k$ et à la capacité β est défini par

$$f_k^2(\beta) = \max\left\{\sum_{j=1}^{k} c_j y_j : \sum_{j=1}^{k} a_j^2 y_j \le \beta, y \in N^k\right\}.$$

Soit u_j une borne supérieure sur la variable du suiveur y_j. Notons qu'une valeur possible est $u_j = \lfloor b/a_j^2 \rfloor$ car les coefficients du vecteur a^2 sont strictement positifs. Les relations de récurrences pour calculer $f_k^2(\beta)$ sont décrites ci-après. L'initialisation (k = 1) est donnée par :

$$f_1^2(\beta) = \min\{hc_1, u_1 c_1\} \text{ pour } \beta \in [ha_1^2, (h+1)a_1^2[.$$

Pour $k = 2, \ldots, n^2$, $f_k^2(\beta)$ est calculée à partir de la formule suivante :

$$f_k^2(\beta) = \max\left\{f_{k-1}^2(\beta - ha_k^2) + hc_k : h = 0...u_j, ha_k^2 \le \beta\right\}.$$

Soit $\Omega = \left\{(\hat{c}_i, \hat{a}_i^2) : i = 1..p\right\}$ l'ensemble des p couples $(\hat{c}_i, \hat{a}_i^2)$, où $\hat{c}_i$ (respectivement $\hat{a}_i^2$) représente les profits (resp. les ressources consommées) des solutions optimales des sous-problèmes du suiveur dont les valeurs optimales sont données par $f_{n^2}^2(\beta)$ pour $\beta = 0,...,b$. Plus explicitement, $\hat{c}_i = cy^i, \hat{a}_i^2 = a^2 y^i$ avec $cy^i = f_{n^2}^2(\beta)$ pour $\beta = 0,...,b$ et p est égal au nombre de solutions.

1.3.2 Phase 2 : Reformulation du SDUB

L'objectif de la seconde phase est de reformuler le problème SDUB sous forme d'un programme linéaire en nombres entiers en se basant sur l'ensemble Ω. Plus précisément, nous établissons un lien entre les actions du meneur avec les réactions compatibles du suiveur sans résoudre le problème du suiveur à nouveau. Le lemme suivant permet de caractériser l'ensemble des réactions du suiveur pour x fixé

$$\mathrm{R}(x) = Arg\max\left\{f^2(y) : a^2 y \le b - a^1 x,\ y \in N^{n^2}\right\}.$$

Lemme 1 : Soit x une variable du meneur fixée, nous avons $y \in \mathrm{R}(x)$ si et seulement si les deux conditions suivantes sont vérifiées :

i) $f_{n^2}^2(b-a^1x)=cy$ et

ii) $a^2y \le b-a^1x$.

Preuve : La preuve suit de la définition de la fonction objectif du suiveur :

$$f_{n^2}^2(b-a^1x)=\max\left\{\sum_{j=1}^{n^2}c_jy_j : \sum_{j=1}^{n^2}a_j^2y_j \le b-a^1x, y\in N^{n^2}\right\}=\max\left\{cy : a^2y \le b-a^1x, y\in N^{n^2}\right\}. \square$$

Avant de redéfinir l'ensemble R(x), nous proposons une procédure pour déterminer les intervalles de sensibilité des réactions y du suiveur respectant l'action du meneur. Il s'agit de déterminer l'intervalle d'appartenance de la capacité résiduelle $b-a^1x$ tel que la solution y reste optimale pour le problème de sac à dos.

Nous notons $[s^1,s^2[$, $[s^2,s^3[$,..., $[s^p,s^{p+1}[$ les intervalles de sensibilité des capacités résiduelles avec $s^1 = 0$ et $s^{p+1} = b+1$. Autrement dit, si $b-a^1x\in[s^l,s^{l+1}[$ alors d'après le Lemme 1 la réaction y du suiveur doit vérifier les conditions : $cy=f_{n^2}^2(s^l)$ et $a^2y\in\left[s^l,b-a^1x\right]$. Les bornes de chaque intervalle sont calculées plus explicitement comme suit :

$$\begin{cases} s^1=0, \\ s^l=\min\left\{b-a^1x : f_{n^2}^2(b-a^1x)>f_{n^2}^2(s^{l-1})\right\} \quad \text{pour } l=2,...,p=\max\left\{l : f_{n^2}^2(b)>f_{n^2}^2\left(s^{l-1}\right)\right\}. \\ s^{p+1}=b+1 \end{cases}$$

Notons que p représente aussi le nombre d'intervalles de sensibilité. Par construction de la suite s^l, nous avons

$$f_{n^2}^2(b-a^1x)=f_{n^2}^2(s^l) \text{ pour tout } b-a^1x\in\left[s^l,s^{l+1}\right[.$$

Avant de donner la reformulation du SDUB, nous présentons ci-dessous quelques résultats théoriques la justifiant.

Proposition 6 : Soit (x, y) une solution du problème SDUB, alors (x, y) $\in$ R si et seulement si :

i) $B^1x+B^2y\le b^1$ et

ii) si $b-a^1x\in[s^l,s^{l+1}[$ alors $cy=f_{n^2}^2(s^l)$ et $a^2y\in\left[s^l,b-a^1x\right]$.

Preuve : Montrons tout d'abord que toute solution (x, y) dans la région induite R vérifie les deux conditions i) et ii). La condition i) est justifiée par la définition de R. D'après le Lemme 1, nous avons $y \in R(x)$ est équivalent à $f_{n^2}^2(b-a^1x)=cy$ et $a^2y \le b-a^1x$. Ainsi d'après la définition des intervalles de sensibilité, nous avons $cy=f_{n^2}^2(s^l)$ et $a^2y \in [s^l, b-a^1x]$ si $b-a^1x \in [s^l, s^{l+1}[$. Ce qui montre que la condition ii) est vérifiée.

Réciproquement, d'après la condition ii) nous avons $cy=f_{n_2}^2(s^l)$ si $b-a^1x \in [s^l, s^{l+1}[$, implique que $f_{n_2}^2(b-a^1x)=cy=f_{n_2}^2(s^l)$ puisque $f_{n_2}^2(\beta)$ est constant sur l'intervalle, pour tout $\beta \in [s^l, s^{l+1}[$. De plus, $a^2\ y \in [s^l, b-a^1x]$ d'où $a^1x+a^2y \le b$. Donc, nous avons $f_{n_2}^2(b-a^1x)=cy$ et $a^2y \le b-a^1x$, ce qui est équivalent à $y \in R(x)$. D'après la condition i), la solution (x, y) appartient à la région induite R. □

Le lemme suivant permet de valider le choix de l'intervalle de sensibilité compatible avec la réaction du suiveur en respectant l'action du meneur.

Lemme 2 : Soit (x, y) une solution du SDUB et s^l les bornes des intervalles de sensibilité. Nous avons :

$$\left.\begin{array}{l} a^2y \le b-a^1x < s^{l+1} \\ \text{et} \\ cy=f_{n_2}^2(s^l) \end{array}\right\} \Rightarrow s^l \le b-a^1x$$

Preuve : Supposons par l'absurde que $s^l > b-a^1x$. Par construction de la suite s^l, il suit que $f_{n^2}^2(b-a^1x) < f_{n^2}^2(s^l)$. Or, par hypothèse, $a^2y \le b-a^1x$, ce qui implique que y est une solution réalisable du sous-problème dont la valeur optimale est $f_{n^2}^2(b-a^1x)$. Ainsi, $cy \le f_{n^2}^2(b-a^1x)$. Par hypothèse, $cy=f_{n^2}^2(s^l)$, et donc $cy \le f_{n^2}^2(b-a^1x) < f_{n^2}^2(s^l) = cy$. D'où la contradiction avec l'hypothèse $s^l > b-a^1x$. □

Le théorème suivant permet de faire le lien entre les éléments précédents et une reformulation linéaire entière du problème SDUB.

Théorème 2 : Soit (x, y) une solution du problème SDUB, alors $(x, y) \in R$ si et seulement si :

$$\begin{cases} B^1x + B^2y \le b^1 & (5-a) \\ a^1x + a^2y \le b & (5-b) \\ a^1x + \sum_{l=1}^{p} s^{l+1}z^l \ge b+1 & (5-c) \\ cy = \sum_{l=1}^{p} f_{n^2}^2(s^l)z^l & (5-d) \\ \sum_{l=1}^{p} z^l = 1 & (5-e) \\ x \in N^{n^1}, y \in N^{n^2}, z \in \{0,1\}^p \end{cases}$$

Preuve : Montrons tout d'abord la condition nécessaire. D'après la Proposition 6, si $(x, y) \in R$ alors les deux conditions ci-dessous sont vérifiées

i) $B^1x + B^2y \le b^1$ et

ii) si $b - a^1x \in [s^l, s^{l+1}[$ alors $cy = f_{n^2}^2(s^l)$ et $a^2y \in [s^l, b - a^1x]$.

Il est trivial que les contraintes (5-a) et (5-b) sont respectées à partir de i) et ii). Pour tout x, il existe un seul intervalle de sensibilité tel que $b - a^1x \in [s^l, s^{l+1}[$. Ce qui ramène à introduire les variables binaires z^l où

$$z^l = \begin{cases} 1 & \text{si } b - a^1x \in [s^l, s^{l+1}[\\ 0 & \text{sinon} \end{cases} \qquad \text{avec} \quad \sum_{l=1}^{p} z^l = 1.$$

Ainsi, il suit que (5-e) est satisfaite et que d'après ii) la contrainte (5-d) est satisfaite. Comme $b - a^1x \in [s^l, s^{l+1}[$ nous avons $s^l \le b - a^1x < s^{l+1}$. En multipliant les inégalités par z^l, nous obtenons $z^l s^l \le z^l(b - a^1x) < z^l s^{l+1}$. En sommant ces contraintes, nous obtenons les inégalités :

$$\sum_{l=1}^{p} z^l s^l \le \sum_{l=1}^{p} z^l(b - a^1x) < \sum_{l=1}^{p} z^l s^{l+1}.$$

En les développant et sachant que $\sum_{l=1}^{p} z^l = 1$, nous obtenons

$$\sum_{l=1}^{p} z^l s^l \le (b - a^1x) < \sum_{l=1}^{p} z^l s^{l+1} \qquad (5\text{-f})$$

L'inégalité gauche de (5-f) est satisfaite d'après (5-b). Puisque les données sont entières, l'inégalité droite de (5-f) peut être réécrite sous la forme de (5-c).

Montrons maintenant la condition suffisante. La contrainte de choix (5-e) assure l'existence d'un indice l tel que $z^l = 1$ et $z^{l'}=0$ pour $l' \neq l$. Cela implique que les contraintes (5-b), (5-c) et (5-d) s'exprime par :

$$\begin{cases} a^1x + a^2y \leq b & (5-\text{b}) \\ a^1x + s^{l+1} \geq b+1 & (5-\text{c}') \\ cy = f^2_{n^2}(s^l) & (5-\text{d}') \end{cases}$$

Puisque les données sont entières, la contrainte (5-c') peut s'écrire $a^1x + s^{l+1} > b$ et les trois contraintes précédentes se réduisent à

$$\left\{ \begin{matrix} a^2y \leq b - a^1x < s^{l+1} \\ cy = f^2_{n^2}(s^l) \end{matrix} \right\}$$

D'après le Lemme 2, nous avons $s^l \leq b - a^1x$, ce qui implique que $b - a^1x \in [s^l, s^{l+1}[$. D'après la condition ii) de la Proposition 6, nous avons donc $(x, y) \in$ R. □

Par définition de la région induite R, et en appliquant le Théorème 2, le problème SDUB peut être formulé comme le programme linéaire en nombres entiers suivant :

$$\text{(IP(SDUB))} \begin{cases} Max \quad g(x,y,z) = d^1x + d^2y \\ s.c. \\ \quad B^1x + B^2y \leq b^1 & (5-\text{a}) \\ \quad a^1x + a^2y \leq b & (5-\text{b}) \\ \quad a^1x + \sum_{l=1}^{p} s^{l+1} z^l \geq b+1 & (5-\text{c}) \\ \quad cy = \sum_{l=1}^{p} f^2_{n^2}(s^l) z^l & (5-\text{d}) \\ \quad \sum_{l=1}^{p} z^l = 1 & (5-\text{e}) \\ \quad x \in N^{n^1}, y \in N^{n^2}, z \in \{0,1\}^p \end{cases}$$

Pour une décision x du meneur, l'intervalle de sensibilité considéré est celui auquel appartient $b - a^1x$. La réaction y doit alors vérifier les contraintes du meneur (5-a) et du suiveur (5-b). Les contraintes (5-c) et (5-d) expriment les conditions du Lemme 2, qui permettent d'associer l'intervalle choisi avec la valeur optimale du suiveur. La contrainte de choix (5-e) exprime que pour chaque décision du meneur un seul intervalle est pris en compte.

L'exemple suivant illustre les étapes de la reformulation d'un SDUB en un problème avec variables entières.

1.3.3 Illustration

Soit le problème

$$(\text{SDUB}_1)\begin{cases} Max\ f^1(x,y) = -x_1 + x_2 + 2y_1 + 5y_2 + y_3 + 10y_4 \\ \quad s.c. \\ \quad\quad 2x_1 - 3x_2 + y_1 + 5y_2 + 3y_3 + 2y_4 \le 5 \\ \quad\quad x_1, x_2 \ge 0, \quad x_1, x_2 \text{ entiers} \\ \quad\quad Max\ f^2(y) = 5y_1 + y_2 + 3y_3 + 2y_4 \\ \quad\quad\quad s.c \\ \quad\quad\quad\quad x_1 + x_2 + 2y_1 + 4y_2 + y_3 + 3y_4 \le 8 \\ \quad\quad\quad\quad y_i \in \{0,1,2\} \quad \text{pour } i = 1...4 \end{cases}$$

La phase 1 consiste à appliquer la programmation dynamique au problème de sac à dos en nombres entiers :

$$(\text{SD}_1)\begin{cases} Max\ f^2(y) = 5y_1 + y_2 + 3y_3 + 2y_4 \\ \quad s.c \\ \quad\quad 2y_1 + 4y_2 + y_3 + 3y_4 \le 8 \\ \quad\quad y_i \in \{0,1,2\} \quad \text{pour } i = 1...4 \end{cases}$$

La première colonne du Tableau 1.6 donne la variation de β de 0 à b = 8, les cellules des quatre colonnes suivantes donnent les valeurs de $f_k^2(\beta)$ pour k = 1,..., 4 et β = 0,..., 8. Les deux dernières colonnes donnent les bornes des intervalles de sensibilité s^l et $f_{n_2}^2(s^l)$. Notons que dans cet exemple le nombre d'intervalles de sensibilité est égal à 7 :

$[s^l, s^{l+1}[= \{l\}$ pour l =1,..., 3 ;

$[s^4, s^5[= \{4,5\}$;

$[s^l, s^{l+1}[= \{l+1\}$ pour l = 5,..., 7 avec $s^8 = 9 = b+1$.

β	y_1	y_2	y_3	y_4	s^l	$f_{n_2}^2(s^l)$
0	0	0	0	0	0	0
1	0	0	3	3	1	3
2	5	5	6	6	2	6
3	5	5	9	9	3	9
4	10	10	12	12	4	12
5	10	10	15	12		
6	10	10	18	18	6	18
7	10	10	21	21	7	21
8	10	11	24	24	8	24

Tableau 1.6 : Phase de programmation dynamique

Ensuite, en se basant sur le Théorème 2 et le Tableau 1.6, le SDUB peut être écrit sous la forme du programme linéaire en nombres entiers suivant :

$$\text{IP(SDUB}_1\text{)} \begin{cases} Max\ g(x,y,z) = -x_1 + x_2 + 2y_1 + 5y_2 + y_3 + 10y_4 \\ s.c. \\ \quad 2x_1 - 3x_2 + y_1 + 5y_2 + 3y_3 + 2y_4 \le 5 & (6\text{-a}) \\ \quad x_1 + x_2 + 2y_1 + 4y_2 + y_3 + 3y_4 \le 8 & (6\text{-b}) \\ \quad x_1 + x_2 + z_1 + 2z_2 + 3z_3 + 4z_4 + 6z_5 + 7z_6 + 8z_7 + 9z_8 \ge 9 & (6\text{-c}) \\ \quad 5y_1 + y_2 + 3y_3 + 2y_4 - z_2 - 2z_3 - 3z_4 - 4z_5 - 6z_6 - 7z_7 - 8z_8 = 0 & (6\text{-d}) \\ \quad z_1 + z_2 + z_3 + z_4 + z_5 + z_6 + z_7 + z_8 = 1 & (6\text{-e}) \\ \quad x_1, x_2 \text{ entiers}, y \in \{0,1,2\}^4, z \in \{0,1\}^8 \end{cases}$$

Les contraintes (6-a) et (6-b) sont les contraintes du meneur et du suiveur associées respectivement aux contraintes (5-a) et (5-b) de la reformulation IP(SDUB$_1$). Les contraintes (6-c), (6-d) et (6-e) assurent que la réaction du suiveur y est optimale pour une action x du meneur.

Avant d'étendre les résultats obtenus dans cette section à un problème de sac à dos multidimensionnel bi-niveaux, qui est une généralisation du problème SDUB avec plusieurs contraintes de sac à dos au second niveau, nous présentons un ensemble de résultats numériques concernant le SDUB.

1.3.4 Résultats numériques

Dans cette section, nous présentons d'abord une étude comparative entre l'algorithme proposé appelé *PD_SDUB* et celui de Moore et Bard (1990) (*M&B*). Ensuite, nous étudions le comportement de *PD_SDUB* sur des instances de grandes tailles.

La programmation des algorithmes a été faite en langage C++ et utilise le solveur CPLEX. Les tests ont été effectués sur un ordinateur à base d'un processeur *Xeon* 5160 de capacité 3 GHz avec 16 Gb de RAM.

Les instances sont générées aléatoirement en faisant varier le nombre de variables (n^1, n^2) et le nombre des contraintes du meneur. Comme dans la Section 1.2.3, il est possible de générer les données du problème du suiveur à l'aide d'un générateur pour le problème du sac à dos selon les trois degrés de corrélation (non corrélé (NC), corrélé (C) et fortement corrélé (FC)). Les données du problème du meneur sont aussi générées aléatoirement. La taille des coefficients du problème est de 1000. Notons que toutes les variables du problème sont

bornées supérieurement par la valeur 10. Les valeurs reportées dans les Tableaux 1.7 et 1.8 représentent la moyenne sur 10 instances de mêmes caractéristiques.

Le comportement des algorithmes *PD_SDUB* et *M&B* a été comparé sur des instances non corrélées (NC). Dans le Tableau 1.7, nous présentons les résultats obtenus pour les tests associés à $m^1 \in \{5,10,15\}$, $n^1 \in \{5,6,7,8,9,10\}$ et $n^2 \in \{10,15,20,25,30\}$.

m^1	n^2	n^1	*PD_SDUB*	*M&B*	m^1	n^2	n^1	*PD_SDUB*	*M&B*	m^1	n^2	n^1	*PD_SDUB*	*M&B*
5	10	5	0.01	0.21	10	10	5	0.01	0.18	15	10	5	0.01	0.40
		6	0.01	2.77			6	0.01	0.04			6	0.01	2.34
		7	0.01	0.01			7	0.01	3.39			7	0.01	2.53
		8	0.01	6.18			8	0.01	4.39			8	0.02	8.90
		9	0.02	40.01			9	0.00	0.01			9	0.01	6.53
		10	0.01	0.01			10	0.00	0.49			10	0.01	0.03
	15	5	0.01	0.11		15	5	0.01	18.36		15	5	0.01	12.69
		6	0.01	14.43			6	0.01	10.84			6	0.01	3.87
		7	0.01	40.32			7	0.00	0.01			7	0.01	6.15
		8	0.02	105.41			8	0.01	24.97			8	0.01	4.91
		9	0.01	27.80			9	0.02	18.40			9	0.01	48.15
		10	0.05	26.68			10	0.00	0.41			10	0.02	21.04
	20	5	0.01	2.81		20	5	0.01	4.32		20	5	0.01	0.12
		6	0.01	6.08			6	0.02	33.63			6	0.04	167.42
		7	0.06	821.46			7	0.01	0.01			7	0.01	6.67
		8	0.02	10.22			8	0.11	639.46			8	0.12	920.60
		9	0.01	9.49			9	0.01	1.23			9	0.01	64.56
		10	0.01	0.08			10	0.03	2663.35			10	0.24	1671.82
	25	5	0.01	7.49		25	5	0.01	0.01		25	5	0.01	24.89
		6	0.01	26.33			6	0.03	250.34			6	0.02	386.34
		7	0.02	4.62			7	0.08	2096.18			7	0.15	336.00
		8	0.08	728.79			8	0.11	64.36			8	0.01	25.10
		9	0.03	18.44			9	0.02	201.19			9	0.11	1435.86
		10	0.04	834.47			10	0.06	10233.96			10	0.13	3373.12
	30	5	0.07	3374.58		30	5	0.01	0.15		30	5	0.03	70.50
		6	0.12	9152.25			6	0.02	6.11			6	0.02	33.15
		7	0.02	10.11			7	0.03	14.06			7	0.07	1630.47
		8	0.14	14874.49			8	0.14	1585.35			8	0.14	21658.78
		9	0.01	0.08			9	0.11	12652.31			9	0.02	415.01
		10	0.01	0.70			10	0.06	163.91			10	0.02	247.63

Tableau 1.7 : Comparaison entre les algorithmes *PD_SDUB* et *M&B*

Le temps d'exécution important pour l'algorithme *M&B* provient du fait que les problèmes CBLP à résoudre à chaque nœud sont des problèmes NP-Difficiles. De plus, la borne supérieure fournie à chaque nœud n'est pas forte. En conséquence, le nombre de nœuds parcourus par l'algorithme de *Branch-and-Bound* augmente très rapidement avec la taille des instances. L'algorithme *PD_SDUB* est très efficace pour résoudre toutes les instances du Tableau 1.7. L'application de la programmation dynamique sur le problème de second niveau comprenant 30 variables au maximum est instantanée. De plus, cette taille de problème génère un nombre limité de variables de choix, ce qui permet de résoudre la reformulation rapidement. Notons que notre algorithme ne résout qu'un seul programme linéaire mixte là où l'algorithme *M&B* en résout un à chaque nœud.

La deuxième partie des expériences réalisées vise à tester l'algorithme sur des instances plus difficiles et de plus grandes tailles en considérant les trois types de corrélation (*NC*, *C* et *FC*). La valeur de b (la capacité du sac dans la contrainte du second niveau de SDUB) est définie par la formule suivante : $\alpha\left(\sum_{i=1}^{n^1} a_i^1 + \sum_{j=1}^{n^2} a_j^2\right)$, où $\alpha = 0.25$, 0.5 ou 0.75.

L'objectif est d'abord d'étudier l'impact du problème du suiveur (la corrélation des instances, le nombre de variables du suiveur et le coefficient de resserrement α). Dans le Tableau 1.8, le nombre de variables du meneur est fixé à 50 et le nombre de contraintes du premier niveau à 5, le nombre de variables du suiveur n^2 varie parmi {100, 200, 300, 400, 500}. La colonne p du Tableau 1.8 donne le nombre de variables de choix (z_i) dans la reformulation, la colonne *CPU_PD* contient le temps d'exécution de la première phase de notre algorithme (basée sur la programmation dynamique), et *CPU_IP* est le temps d'exécution de la deuxième phase (résolution de la reformulation du problème). Nous reportons dans la colonne *opt* le nombre de fois (sur 10) où une solution optimale est obtenue avant la limite de temps fixée à une heure. La colonne *Gap* donne en pourcentage la moyenne des écarts entre les bornes fournies par CPLEX sur les 10 instances.

n^2	α	CPU-PD			p			CPU-IP								
		NC	*C*	*FC*	*NC*	*C*	*FC*	*NC*	*opt*	*Gap*	*C*	*opt*	*Gap*	*FC*	*opt*	*Gap*
100	0.25	0.08	0.09	0.08	1809	3376	13942	68.13	10	<0.01	579.47	9	<0.01	1621.55	7	<0.01
100	0.50	0.19	0.20	0.18	3270	8037	29524	375.83	10	<0.01	2588.85	3	<0.01	2364.43	5	<0.01
100	0.75	0.29	0.30	0.29	4720	17930	44849	25.28	10	<0.01	2421.50	4	<0.01	3292.96	2	<0.01
200	0.25	0.32	0.34	0.34	5072	14136	27815	31.42	10	<0.01	1695.98	7	<0.01	2680.79	3	<0.01
200	0.50	0.69	0.70	0.70	9785	23374	56930	99.54	10	<0.01	2687.36	3	<0.01	3226.96	3	<0.01
200	0.75	1.06	1.09	1.06	15070	38760	85031	970.73	8	0.29	3089.81	3	0.16	3585.53	2	0.01
300	0.25	0.73	0.74	0.74	10251	18668	40909	1014.02	8	0.13	2460.34	5	0.01	3264.98	2	0.01
300	0.50	1.50	1.50	1.51	18674	45959	82444	858.37	10	<0.01	3061.25	2	<0.01	3300.86	4	0.07
300	0.75	2.28	2.27	2.31	26975	65583	123950	1093.53	8	0.26	3374.10	1	0.17	3267.27	3	0.01
400	0.25	1.30	1.31	1.30	16398	30670	53625	1683.34	7	0.31	2969.62	4	0.17	3509.37	2	0.01
400	0.50	2.64	2.64	2.66	31679	62792	107538	1687.08	7	0.18	3692.89	0	0.17	3338.82	3	0.01
400	0.75	4.06	4.05	3.99	42590	95841	161218	908.05	10	<0.01	2388.21	5	0.01	3049.60	3	<0.01
500	0.25	1.98	1.97	1.99	25816	40841	65264	2039.72	7	0.06	3154.97	2	0.34	3331.29	1	<0.01
500	0.50	4.12	4.07	4.00	43590	94759	132427	2696.57	4	0.04	3457.63	2	0.33	3169.08	3	<0.01
500	0.75	6.21	6.17	5.95	64903	132365	197620	2564.93	9	<0.01	3022.18	2	0.33	3267.46	3	<0.01

Tableau 1.8 : Effets de la taille du problème du suiveur

Nous remarquons d'abord que l'augmentation du nombre de variables du suiveur n^2 entraîne une augmentation du temps d'exécution de la première phase, ainsi que la taille de la reformulation IP(SDUB) et le temps d'exécution nécessaire pour la résoudre. Ainsi par exemple dans le cas où $\alpha = 0.25$, le temps d'exécution de la programmation dynamique varie de 0.08 secondes pour $n^2 = 100$ à 1.98 secondes pour n^2=500 (coefficients *NC*). Le nombre de

variables de choix dans la reformulation augmente sensiblement (de 1809 à 25816), car le nombre de solutions non dominées de la programmation dynamique augmente avec le nombre de combinaisons possibles des solutions. Le temps de résolution de la reformulation passe alors de 68 à 2039 secondes. Ce comportement est aussi valide pour les instances corrélées et fortement corrélées, et pour toute valeur de α (voir Tableau 1.8).

Les figures suivantes illustrent le comportement de l'algorithme PD_SDUB relativement à la valeur de α. La Figure 1.7-a (resp. Figure 1.7-b) donne le nombre moyen de variables de choix (resp. le temps de résolution de la reformulation) pour chaque valeur de n^2.

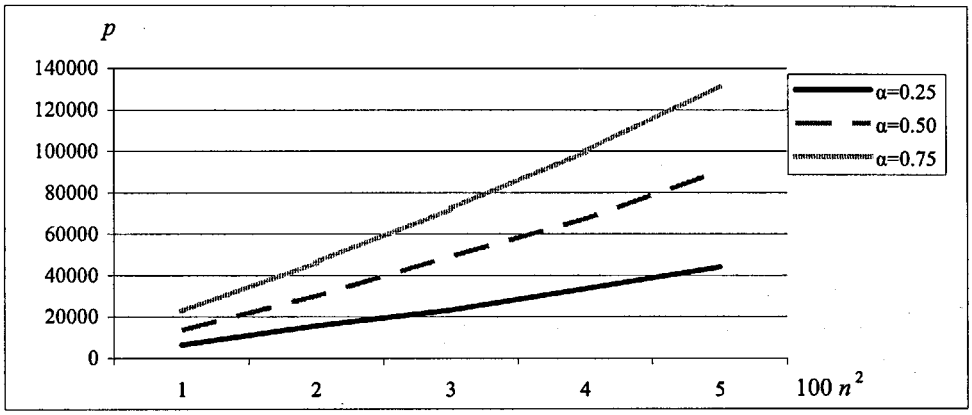


Figure 1.7-a : Impact de α sur p

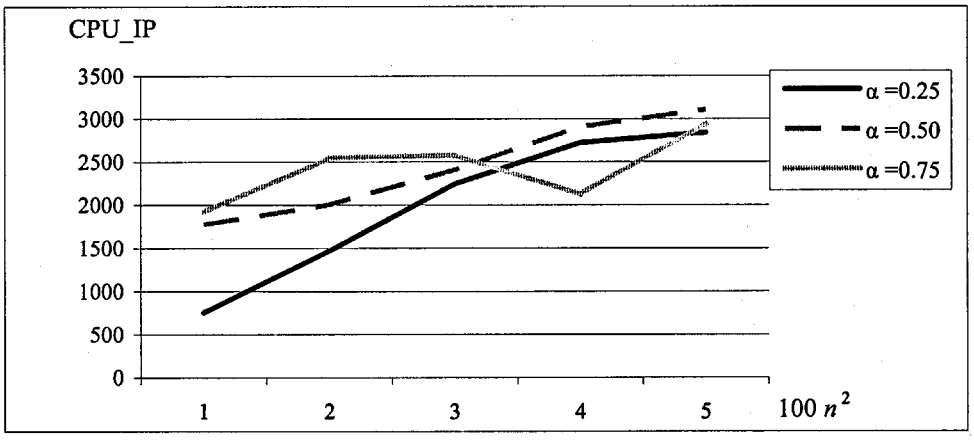


Figure 1.7-b : Impact de α sur le CPU_IP

La Figure 1.7-a montre clairement que l'augmentation de la valeur de α génère plus de variables de choix à cause de l'augmentation de la capacité du sac. Cela complique en général la résolution de la reformulation comme le montre la Figure 1.7-b. Dans le Tableau 1.8, nous ne pouvons pas toujours trouver le lien entre le nombre de variables de choix et le temps de résolution du modèle, car ce problème dépend aussi des données du premier niveau.

Les figures suivantes illustrent le comportement de la méthode relativement au degré de corrélation. La première (resp. la deuxième) figure donne le nombre moyen de variables de choix (resp. le temps d'exécution de la reformulation) pour chaque valeur de n^2.

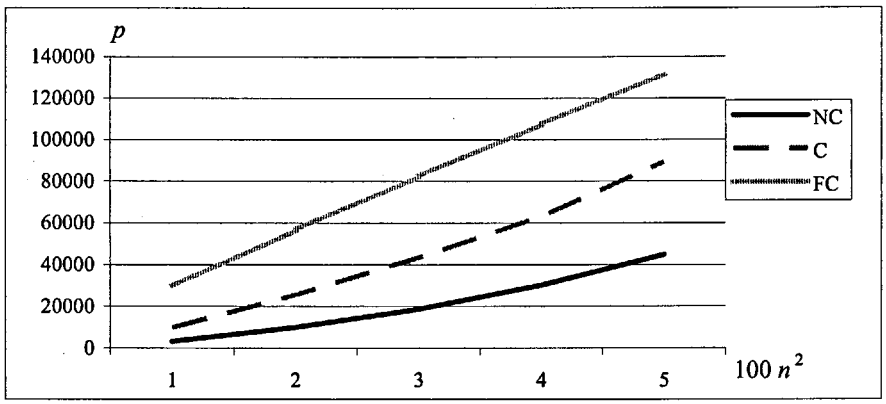


Figure 1.8-a : Impact de la corrélation sur p

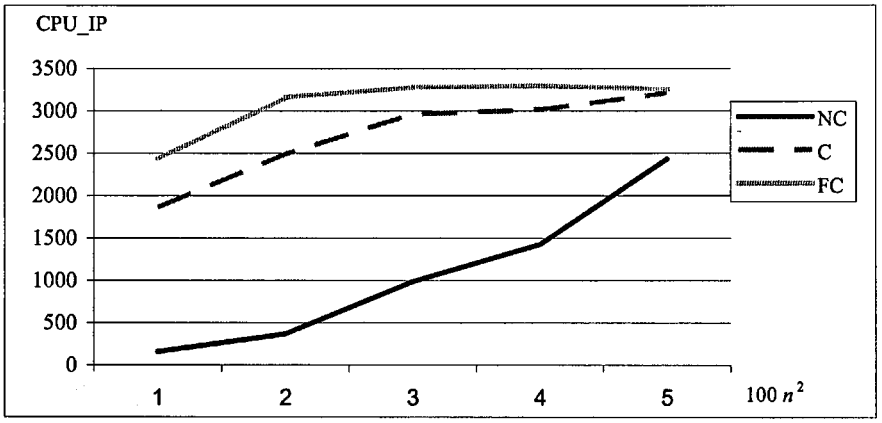


Figure 1.8-b : Impact de la corrélation sur CPU_IP

Le degré de corrélation des coefficients du problème du second niveau influence le nombre de variables de choix. Plus les instances sont corrélées et plus le nombre de solutions non dominées devient important, ce qui génère plus de variables de choix (Figure 1.8-a). Celui-ci complique davantage la résolution de la reformulation comme le montre la Figure 1.8-b. Le temps de résolution de la reformulation est donc toujours influencé par l'augmentation du nombre de variables de choix.

Nous examinons aussi l'impact du nombre de contraintes de premier niveau sur l'efficacité de l'algorithme *PD_SDUB*. Dans le Tableau 1.9, nous présentons les résultats obtenus lorsque le nombre de variables du meneur et du suiveur sont fixés alors que le nombre de contraintes du meneur varie. La colonne *opt* donne le nombre de fois où l'algorithme *PD_SDUB* trouve une solution optimale sur un ensemble de 30 instances.

m^1	p	CPU_IP	opt	Gap(%)	m^1	p	CPU_IP	opt	Gap(%)
5	3266	157.03	29	< 0.01	5	5234	1261.78	22	0.58
10	3270	617.04	25	0.31	10	5396	1511.94	18	1.50
15	3433	313.47	28	0.01	15	5178	886.00	24	0.39
20	3402	311.36	28	0.07	20	5102	435.93	27	< 0.01
25	3333	7.22	30	< 0.01	25	5126	523.57	27	0.01
30	3323	10.17	30	< 0.01	30	4829	816.77	25	0.15
35	3276	100.22	30	< 0.01	35	5014	861.00	23	0.45
40	3560	5.34	30	< 0.01	40	5634	969.47	23	0.45
45	3199	5.49	30	< 0.01	45	5355	864.47	23	0.03
50	3246	1.88	30	< 0.01	50	4871	988.56	22	0.28
$n^1 = 50, n^2 = 100$					$n^1 = 100, n^2 = 100$				
5	7566	1565.00	19	2.09	5	10843	2185.56	14	1.06
10	7481	1744.72	16	3.09	10	10726	2712.86	9	3.96
15	8261	1807.29	18	0.73	15	10470	2516.06	10	2.91
20	7357	1702.47	16	1.10	20	10919	2278.35	14	1.16
25	7950	952.56	24	0.34	25	10099	2125.08	12	1.38
30	7583	1069.11	22	0.93	30	11262	2038.10	14	1.12
35	7607	1013.98	22	0.36	35	11029	1080.86	22	0.47
40	8036	1512.62	18	0.64	40	10243	2214.58	13	0.36
45	8074	1309.99	20	0.79	45	10054	1251.00	21	0.67
50	8084	1447.81	21	0.25	50	11146	1220.42	22	0.71
$n^1 = 150, n^2 = 100$					$n^1 = 200, n^2 = 100$				
5	12462	2622.16	10	3.73	5	13669	2303.69	11	5.38
10	12857	3227.88	5	2.63	10	13545	2284.39	11	3.74
15	13937	2322.97	11	3.64	15	14207	1955.83	14	2.39
20	13114	2405.78	10	2.90	20	14227	2009.27	14	1.63
25	12829	1361.46	20	1.49	25	13903	1655.01	18	1.36
30	12340	1565.24	18	0.80	30	13889	1793.79	15	1.22
35	12443	1194.77	21	0.89	35	13898	1455.30	18	0.80
40	13090	1575.55	18	0.81	40	13828	1213.75	21	0.34
45	12440	1237.35	20	0.65	45	13884	1019.96	22	0.27
50	12905	1351.09	20	0.36	50	13826	673.60	25	0.21
$n^1 = 250, n^2 = 100$					$n^1 = 300, n^2 = 100$				

Tableau 1.9 : Impact des contraintes du meneur

Nous remarquons dans le Tableau 1.9 que l'Algorithme *PD_SDUB* rencontre des difficultés pour identifier une solution optimale en une heure d'exécution dans certains cas où le nombre de contraintes du meneur est faible. La prise en compte de contraintes au premier niveau complique en général la résolution du problème bi-niveaux. Or dans notre cas,

l'existence des contraintes du premier niveau peut réduire l'espace de solutions réalisables, ce qui permet alors à CPLEX de trouver une solution optimale dans un temps raisonnable. Le Gap reporté dans les deux Tableaux 1.8 et 1.9 montre en effet que les solutions trouvées sont généralement de bonne qualité. Cela s'explique par le fait qu'il est parfois rapide pour un algorithme de *Branch-and-Bound* d'obtenir une solution optimale d'un sac à dos multidimensionnel mais très difficile de prouver l'optimalité de cette solution (voir par exemple Boussier (2008)).

Afin d'illustrer le comportement de l'algorithme *PD_SDUB*, nous donnons dans la Figure 1.9 l'évolution des la bornes supérieures et inférieures lors de la résolution d'une instance du SDUB générée avec les paramètres suivants : $n^1 = 50$, $n^2 = 100$, $m^1 = 10$, $\alpha = 0{,}75$ et avec des coefficients fortement corrélés.

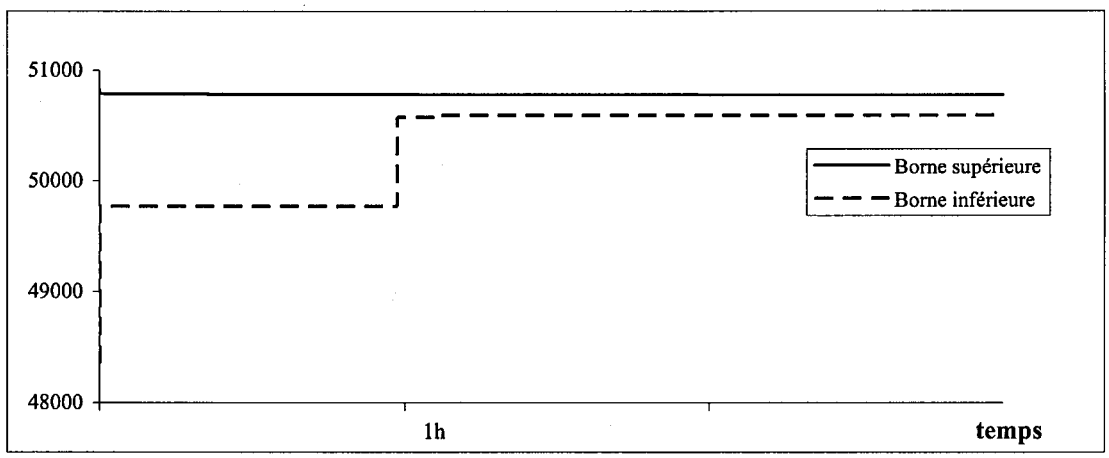


Figure 1.9 : Evolution de la borne inférieure de la reformulation

Nous remarquons clairement que, pour cette instance, la borne inférieure trouvée est de bonne qualité, avec un *Gap* de 0,36%. Cette borne a été trouvée en moins d'une heure d'exécution. Sa qualité se justifie par les explications précédentes. En conséquence, la qualité de la solution dépend fortement de l'instance, et notamment du problème du meneur.

Dans le Tableau 1.10, nous comparons la qualité de la borne supérieure pour le SDUB fournie par notre algorithme à celle considérée dans la littérature (Moore et Bard (1990)). Cette borne est calculée par la résolution de la relaxation du problème bi-niveaux (HPP) (voir Section 1.1), où la fonction objectif du suiveur et les contraintes d'intégrité sur les variables entières du problème sont ignorées. La borne supérieure fournie par l'algorithme *PD_SDUB* est la relaxation en continu de la reformulation IP(SDUB). Les instances considérées dans cette partie sont générées aléatoirement avec des coefficients *NC*. L'amélioration obtenue par l'algorithme *PD_SDUB* pour la borne supérieure (*ub*) du problème SDUB est exprimée en pourcentage, et est calculée par la formule :

$$v(\text{HPP(SDUB)}) - v(\text{H(IP(SDUB))}) \,/\, v(\text{HPP(SDUB)}) - f^{1*}(\text{SDUB}),$$

où v(P) indique la valeur optimale du problème P, H(IP(SDUB)) est la relaxation en continu de la reformulation proposée, f^{1*}(SDUB) est la valeur optimale du SDUB, et HPP(SDUB) est la relaxation du SDUB. Nous donnons dans la ligne CPU_ub le temps d'exécution nécessaire pour trouver la borne supérieure via notre reformulation. Notons que le temps d'exécution pour obtenir v(HPP(SDUB)) est négligeable. Les valeurs dans le Tableau 1.10 présentent la moyenne sur 10 instances de mêmes caractéristiques.

	n^1	50				100			
	m^1	5		10		5		10	
$n^2 = 100$	α	0.25	0.50	0.25	0.50	0.25	0.50	0.25	0.50
	ub	91%	96%	92%	90%	95%	98%	93%	98%
	CPU_ub	0.96	3.87	0.50	2.29	2.64	14.05	3.06	13.80

Tableau 1.10 : Qualité de la borne supérieure de l'algorithme de *PD_SDUB*

Les bornes supérieures fournies par notre reformulation surpassent significativement celle produites par *M&B*. Cette différence se justifie théoriquement par la prise en compte des coupes qui respectent la fonction objectif du suiveur, alors que cette fonction est complètement ignorée dans la relaxation HPP. L'obtention de meilleures bornes supérieures pour le SDUB n'est pas coûteuse en termes du temps d'exécution comme le montre le Tableau 1.10 (le temps d'exécution reporté est celui de la programmation dynamique et de la résolution de la relaxation en continu de la reformulation).

1.4 Heuristique pour le sac à dos multidimensionnel bi-niveaux

Le problème du sac à dos multidimensionnel bi-niveaux en nombres entiers (**SDMB**) est un problème d'optimisation où l'ensemble des solutions réalisables est déterminé par l'ensemble des solutions optimales d'un problème de sac à dos multidimensionnel paramétrique. Le SDMB peut être formulé comme suit :

$$\text{(SDMB)} \begin{cases} \underset{x,y}{Max}\ f^1(x,y) = d^1x + d^2y \\ \quad s.c. \\ \quad\quad B^1x + B^2y \le b^1 \\ \quad\quad x \in N^{n^1} \\ \quad\quad \underset{y}{Max}\ f^2(y) = cy \\ \quad\quad\quad s.c. \\ \quad\quad\quad A^1x + A^2y \le b^2 \\ \quad\quad\quad y \in N^{n^2} \end{cases}$$

où le vecteur d^1 (respectivement c, d^2) est de dimension n^1 (resp. n^2). Le vecteur b^1 (respectivement b^2) est de dimension m^1 (resp. m^2). Les coefficients des matrices B^1 $(m^1 \times n^1)$ et B^2 $(m^1 \times n^2)$ sont des entiers. Nous supposons que les coefficients des matrices $A^1(m^2 \times n^1)$ et $A^2(m^2 \times n^2)$ et les coefficients du second membre b^2 sont des entiers positifs. Le vecteur x (resp. y) de dimension n^1 (resp. n^2) représente les variables de décision du meneur (resp. du suiveur). La fonction objectif du meneur (resp. du suiveur) est notée $f^1(x,y)$ (resp. $f^2(y)$). A notre connaissance, le SDMB n'a pas été étudié dans la littérature, sauf lorsque l'ensemble des contraintes du meneur est vide.

L'approche proposée dans la section précédente peut être généralisée au SDMB en adaptant la première phase de la programmation dynamique au problème du sac à dos multidimensionnel. Malheureusement, cette approche ne permet pas de résoudre des instances de taille significative car elle nécessite un effort calculatoire (mémoire et temps) important. C'est pourquoi nous proposons une heuristique basée sur la relaxation surrogate qui nous permet de définir des problèmes de sac à dos unidimensionnels bi-niveaux résolus par l'approche précédente.

1.4.1 Relaxation surrogate

La relaxation surrogate a été introduite pour la première fois par Glover (1965). Elle consiste à agréger un ensemble de contraintes en une seule. Cette relaxation a fait l'objet de nombreuses études afin de proposer des méthodes de calcul des multiplicateurs optimaux du dual surrogate. Elle a souvent été utilisée pour résoudre le problème du sac à dos multidimensionnel. Nous pouvons ainsi citer par exemple les travaux de Greenberg et Pierskela (1970), Glover (1975), Dyer (1980), Gavish et Pirkul (1985), Fréville et Plateau (1986) et plus récemment Osorio, Glover et Hammer (2002), Fréville et Hanafi (2005) et Hanafi et Glover (2007) (voir la Section 2.3.2 de la Partie 2 pour plus de détails).

Avant de présenter l'heuristique, nous proposons une relaxation spécifique du SDMB basée sur la relaxation surrogate. Plus explicitement, les contraintes du sac à dos du suiveur sont déplacées au premier niveau et remplacées par une contrainte surrogate au second niveau.

La relaxation du SDMB associée au multiplicateur $\mu \in \Re_+^{m^2}$ est alors définie par :

$$(\text{SDMB}(\mu))\begin{cases} \underset{x,y}{Max}\ f^1(x,y)=d^1x+d^2y \\ \quad s.c. \\ \quad B^1x+B^2y\le b^1 & (7\text{-a}) \\ \quad A^1x+A^2y\le b^2 & (7\text{-b}) \\ \quad x\in N^{n^1} \\ \quad \underset{y}{Max}\ f^2(y)=cy \\ \quad\quad s.c. \\ \quad\quad \sum_{i=1}^{m^2}\mu_i\left(A_i^1x+A_i^2y\right)\le\sum_{i=1}^{m^2}\mu_i b_i^2 & (7\text{-c}) \\ \quad\quad y\in N^{n^2} \end{cases}$$

Proposition 7 : Toute solution (x, y) réalisable pour la relaxation SDMB(μ) est aussi réalisable pour le primal SDMB.

Preuve : Soit (x', y') une solution réalisable pour la relaxation SDMB(μ), alors y' est une solution optimale du problème de sac à dos suivant :

$$(\text{SD}(\mu, x'))\begin{cases} \underset{y}{Max}\ f^2(y)=cy \\ \quad s.c. \\ \quad \sum_{i=1}^{m^2}\mu_i\left(A_i^1x'+A_i^2y\right)\le\sum_{i=1}^{m^2}\mu_i b_i^2 \\ \quad y\in N^{n^2} \end{cases}$$

Et puisque la solution (x', y') satisfait les contraintes (7-b), alors y' est forcément une solution optimale du problème de sac à dos multidimensionnel suivant :

$$(\text{SDM}(x'))\begin{cases} \underset{y}{Max}\ f^2(y)=cy \\ \quad s.c. \\ \quad A^1x'+A^2y\le b^2 \\ \quad y\in N^{n^2} \end{cases}$$

Ce qui implique trivialement que (x', y') est une solution réalisable pour le problème primal SDMB. □

L'heuristique que nous proposons, décrite dans la section suivante, consiste à résoudre les relaxations SDMB(μ) en faisant varier μ. Le problème SDMB(μ) peut quant à lui être reformulé selon l'approche décrite dans la Section 1.3 pour le SDUB.

1.4.2 Formulation approchée du SDMB

D'après la Proposition 7, une solution de chaque relaxation SDMB(μ) définit une solution réalisable pour le problème SDMB. Notre but consiste à sélectionner la meilleure solution parmi l'ensemble des solutions réalisables engendrées par les résolutions des relaxations SDMB(μ). Par la suite, nous ne considérons que les relaxations SDMB(μ), avec $\mu_i = 1$ et $\mu_j = 0, j = 1...m^2$ et $j \neq i$, définies par $SDMB_i$ suivant :

$$(SDMB_i)\begin{cases} \underset{x,y}{Max}\ f^1(x,y) = d^1x + d^2y \\ \quad s.c. \\ \quad\quad B^1x + B^2y \leq b^1 \\ \quad\quad A^1x + A^2y \leq b^2 \\ \quad\quad x \in N^{n^1} \\ \quad\quad \underset{y}{Max}\ f^2(y) = cy \\ \quad\quad\quad s.c. \\ \quad\quad\quad A_i^1x + A_i^2y \leq b_i^2 \\ \quad\quad\quad y \in N^{n^2} \end{cases}$$

Une valeur approchée de v(SDMB) est alors définie par Max = $\{v(SDMB_i), i = 1,\ldots, m^2\}$. Une conséquence de la Section 1.3 est que chaque relaxation $SDMB_i$ peut être reformulée sous la forme d'un programme linéaire en nombres entiers :

$$(IP(SDMB_i))\begin{cases} Max \quad g(x,y,z) = d^1x + d^2y \\ \quad s.c. \\ \quad\quad B^1x + B^2y \leq b^1 \\ \quad\quad A^1x + A^2y \leq b^2 \\ \quad\quad A_i^1x + \sum_{l=1}^{p_i} s_i^{l+1} z^l \geq b_i^2 + 1 \\ \quad\quad cy = \sum_{l=1}^{p_i} f_{n^2}^2(s_i^l) z^l \\ \quad\quad \sum_{l=1}^{p_i} z^l = 1 \\ \quad\quad x \in N^{n^1}, y \in N^{n^2}, z \in \{0,1\}^{p_i} \end{cases}$$

où s_i^l sont les bornes des intervalles de sensibilité et p_i leur nombre associé à la $i^{ème}$ contrainte du suiveur.

Nous proposons un seul modèle approché pour le SDMB dont la résolution est équivalente à celle de $\max\{v(\text{IP(SDMB}_i)) : i = 1, \ldots, m^2\}$. Pour modéliser le fait qu'il n'existe qu'une seule contrainte du suiveur correspondant à une solution optimale du modèle IP(SDMB), nous ajoutons la contrainte de choix $\sum_{i=1}^{m^2} \mu_i = 1$, avec $\mu \in \{0,1\}^{m^2}$. Ainsi la variable μ_i indique si la $i^{ème}$ contrainte du suiveur est conservée dans la reformulation. Nous donnons ici la formulation approchée du problème SDMB :

$$
(IP(SDMB))\begin{cases}
Max \quad f^1(x,y,z) = d^1x + d^2y & (8-\text{a}) \\
s.c. & \\
\quad A^1x + B^1y \le b^1 & (8-\text{b}) \\
\quad A^2x + B^2y \le b^2 & (8-\text{c}) \\
\quad \left(A_i^2 \times \mu_i\right)x + \sum_{l=1}^{p_i} s_i^{l+1} z_i^l \ge \left(b_i^2 \times \mu_i\right) + \mu_i \quad \forall i = 1,\ldots,m^2 & (8-\text{d}) \\
\quad cy = \sum_{i=1}^{m^2} \sum_{l=1}^{p_i} f_{n^2}^2(s_i^{l+1}) z_i^l & (8-\text{e}) \\
\quad \sum_{l=1}^{p_i} z_i^l = \mu_i \qquad \forall i = 1,\ldots,m^2 & (8-\text{f}) \\
\quad \sum_{i=1}^{m^2} \mu_i = 1 & (8-\text{g}) \\
\quad x \in N^{n^1}, y \in N^{n^2}, \mu_i \in \{0,1\}^{m^2}, z_i \in \{0,1\}^{p_l}, i = 1,\ldots,m^2 &
\end{cases}
$$

Dans cette formulation, si une variable $\mu_i = 0$, alors la contrainte (8-d) devient $\sum_{l=1}^{p_i} s_i^{l+1} z_i^l \ge 0$, ce qui est toujours vrai. Dans ce cas, la contrainte (8-f) s'écrit $\sum_{l=1}^{p_i} z_i^l = 0$, c'est-à-dire que toutes les variables de choix associées à la $i^{ème}$ contrainte du suiveur et toute la partie associée à ces variables dans la contrainte (8-e) sont nulles. Dans le cas où μ_i n'est pas nul, ces trois contraintes ne seront évidemment pas relâchées et aucune partie associée ne sera annulée.

Cette formulation approchée assure l'obtention d'une solution réalisable pour le problème d'origine SDMB d'après la Proposition 7. Elle peut également nous donner une information sur l'impact des contraintes par rapport à la qualité des solutions associées. Par exemple, la résolution de la relaxation en continu du problème IP(SDMB) donne une solution avec différentes valeurs pour les variables μ_i. Plus la valeur d'une variable μ_i est grande, plus la

contrainte associée est importante par rapport aux autres contraintes. Notons que la reformulation demande en pratique un temps important (i.e, le temps de la programmation dynamique), et qu'elle génère un modèle de grande taille, ce qui complique la résolution du IP(SDMB).

1.5 Conclusions et perspectives

Nous avons proposé dans cette partie un nouvel algorithme exact pour résoudre le problème du sac à dos bi-niveaux (SDB). Le SDB est un cas particulier des problèmes de programmation bi-niveaux linéaires en nombres mixtes dans lequel le meneur contrôle une variable continue (la capacité du sac à dos) et le suiveur résout un problème de sac à dos paramétrique. Nous avons montré que la résolution du SDB est équivalente à la résolution d'une série de problèmes de sac à dos. L'algorithme proposé est basé sur la programmation dynamique, où les règles de récurrences prennent en compte simultanément la fonction objectif du meneur et celle du suiveur. Les expériences numériques montrent d'une part que l'algorithme proposé surpasse de manière significative les algorithmes présents dans la littérature, et d'autre part qu'il est plus robuste aux modifications des instances, en particulier le degré de corrélation.

Nous avons également proposé un nouvel algorithme pour résoudre le problème du sac à dos bi-niveaux avec plusieurs contraintes au premier niveau (SDUB). Le SDUB est un cas particulier des problèmes bi-niveaux linéaires en nombres entiers dans lequel le meneur contrôle cette fois un ensemble de variables en nombres entiers, alors que le suiveur résout toujours un problème de sac à dos paramétrique. L'approche proposée est composée de deux phases : la première est basée sur la programmation dynamique dans laquelle les règles de récurrences ne considèrent que le problème du suiveur. Dans la deuxième phase, nous reformulons le problème SDUB sous forme d'un programme linéaire en nombres entiers. Les expériences numériques ont prouvé que l'algorithme proposé est plus puissant que les algorithmes existants dans la littérature. Il nous permet de résoudre des problèmes de taille moyenne dans un temps raisonnable.

En perspective de cette thèse nous comptons étendre l'étude du problème de sac à dos multidimensionnel bi-niveaux. Une heuristique basée sur une relaxation surrogate particulière des contraintes du suiveur a été proposée dans la Section 1.4. De nombreuses questions restent ouvertes pour le développement de schémas de mise à jour des multiplicateurs. De

façon plus générale, d'autres perspectives peuvent être envisagées comme par exemple l'utilisation de la programmation dynamique de manière approximative afin de garder seulement un sous-ensemble des réactions du suiveur les plus prometteuses.

Partie 2 : Sac à dos multidimensionnel à choix multiples

Le travail dans cette partie se concentre sur le problème du sac à dos multidimensionnel à choix multiples (MMKP) qui, comme son nom l'indique, est une variante du problème du sac à dos multidimensionnel. Comme nous le verrons dans la suite, l'ajout de contraintes de choix rend ce problème encore plus complexe. Devant la difficulté de trouver une solution optimale, ou même juste réalisable du MMKP, nous nous sommes concentrés sur des approches hybrides (Hanafi, Mansi et Wilbaut (2009)). Les techniques proposées combinent les relaxations et les recherches locales en exploitant les informations duales et l'ajout de pseudo-coupes pour réduire l'espace de recherche. De plus, nous avons intégré des techniques de fixation de variables associées aux relaxations proposées.

Dans la première section nous présentons le modèle mathématique pour le MMKP et un ensemble d'applications existantes. La deuxième section s'intéresse à la connexion du MMKP avec d'autres variantes de sac à dos et à l'étude de la littérature. Dans la troisième section, nous nous intéressons à différentes relaxations possibles du MMKP, avant de présenter dans la quatrième section une méthode de recherche locale. Nous abordons dans la cinquième section notre approche hybride basée sur l'utilisation de relaxations. Dans la dernière section, nous étudions la qualité de nos heuristiques en les comparant aux meilleures approches recensées dans la littérature sur un ensemble de 33 instances existantes. Nous montrons que nos algorithmes convergent rapidement vers des solutions élites.

2.1 Formulation et Applications du MMKP

2.1.1 Formulation 0-1 MIP du MMKP et notations

Le problème du sac à dos multidimensionnel à choix multiples (**MMKP** pour *Multi-choice Multidimensional Knapsack Problem*) est une variante du problème du sac à dos (Martello et

Toth (1990), Kellerer, Pferschy et Pisinger (2004)). Le MMKP est défini par un ensemble d'objets G, divisé en n groupes (ou classes) disjoints $G = G_1 \cup \ldots \cup G_n$, avec $G_i \cap G_{i'} = \varnothing$ pour $i' \neq i \in \{1,\ldots, n\}$ où chaque groupe G_i a $n_i = |G_i|$ objets. À chaque objet ($j \in G_i$) du groupe G_i est associé un profit c_{ij} et un ensemble de poids $A_{ij} = (a_{ij}^1, a_{ij}^2, \ldots a_{ij}^k, \ldots, a_{ij}^m)$ relatifs à m contraintes de sac à dos. Le vecteur $b = (b^1, b^2, \ldots, b^k, \ldots, b^m)$ représente l'ensemble des capacités des sacs. Plus formellement, le MMKP peut être défini comme suit :

$$\text{(MMKP)} \begin{cases} \max \sum_{i=1}^{n} \sum_{j \in G_i} c_{ij} x_{ij} & \\ s.c. & \\ \quad \sum_{i=1}^{n} \sum_{j \in G_i} a_{ij}^k x_{ij} \leq b^k & \forall k = 1,\ldots,m \quad (1) \\ \quad \sum_{j \in G_i} x_{ij} = 1 & \forall i = 1,\ldots,n \quad (2) \\ \quad x_{ij} \in \{0,1\} & \forall i = 1,\ldots,n, j \in G_i \quad (3) \end{cases}$$

L'objectif consiste à choisir un objet et un seul dans chaque classe de façon à maximiser le profit total sans violer les contraintes de sac à dos. Les contraintes de sac à dos (1) assurent que la capacité de chaque sac k n'est pas excédée tandis que les contraintes de choix (2) assurent la sélection d'un et un seul objet dans chaque classe de G. Les variables du problème x_{ij} sont des variables binaires qui indiquent le choix de l'objet si $x_{ij} = 1$ ($x_{ij} = 0$ dans le cas contraire) pour $j \in G_i$ et $i = 1,\ldots, n$. Le problème MMKP est un problème NP-Difficile dont la difficulté de résolution à l'optimum augmente exponentiellement avec la taille du problème (Khan, Li, Manning et Akbar (2002)). Avant de décrire quelques applications existantes du MMKP, nous introduisons ici quelques notations utilisées dans la suite de cette partie. La *relaxation en continu* du MMKP, notée LP(MMKP), considère la même fonction objectif que le MMKP et toutes les contraintes (de sac à dos et de choix), sauf les contraintes d'intégrité. Elle est présentée par le modèle suivant :

$$\text{LP(MMKP)} \begin{cases} \max \sum_{i=1}^{n} \sum_{j \in G_i} c_{ij} x_{ij} & \\ s.c. & \\ \quad \sum_{i=1}^{n} \sum_{j \in G_i} a_{ij}^k x_{ij} \leq b^k & \forall k = 1,\ldots,m \\ \quad \sum_{j \in G_i} x_{ij} = 1 & \forall i = 1,\ldots,n \\ \quad x_{ij} \in [0,1] & \forall i = 1,\ldots,n, j \in G_i \end{cases}$$

De manière générale, soit P un problème d'optimisation en variables binaires portant sur un ensemble N de variables. Nous notons par $v(P)$ la valeur optimale du problème P et par (P | Q) le problème P auquel est ajouté un ensemble de contraintes Q. Soit un vecteur $y \in [0,1]^{|N|}$, nous définissons les ensembles : $J^0(y) = \{j \in N, y_j = 0\}$, $J^1(y) = \{j \in N, y_j = 1\}$, $J^*(y) = \{j \in N, y_j \in]0, 1[\}$ et $J(y) = \{j \in N, y_j \in \{0, 1\}\} = J^0(y) \cup J^1(y)$. Nous appelons problème réduit associé au vecteur y le problème suivant : $P(y) = \{P \mid x_j = y_j, j \in J(y)\}$. De plus, le vecteur e est le vecteur avec toutes ses composantes égales à un avec la dimension appropriée, $e = (1, ..., 1)$. Soient x et y deux vecteurs de $\Re^{|N|}$, l'inégalité $x \leq y$ signifie que $x_i \leq y_i$ pour $i = 1, ..., |N|$. La distance de *Hamming*, entre deux vecteurs binaires x et y, est le nombre de composantes de x qui diffèrent de celles de y. Formellement, la distance de *Hamming* entre x et y est définie par :

$$\delta(x,y) = \sum_{i=1}^{|N|} |x_i - y_i| .$$

2.1.2 Applications du MMKP

Le MMKP est une variante intéressante de la famille du sac à dos. Nous présentons ainsi dans la Section 2.2 un ensemble de problèmes d'optimisation directement connectés au MMKP. Plusieurs applications pratiques du MMKP peuvent être recensées dans la littérature. Nous présentons ici quelques exemples.

Le problème MMKP peut ainsi modéliser des problèmes pour l'affectation et l'allocation de ressources (Gavish et Pirkul (1982)), la gestion de budgets (Pisinger (2001)), la fiabilité de systèmes (Tillman, Hwang et Kuo (1980)), les systèmes multimédias (Chen, Khan, Li et Manning (1999), Khan, Li et Manning (1997)), la gestion de réseaux de télécommunications (Watson (2001), Hifi, Michrafy et Sbihi (2004)).

Plusieurs problèmes d'emplois du temps ont aussi été modélisés en faisant apparaître des problèmes de type MMKP. Ainsi dans le milieu hospitalier, la gestion du planning des infirmières a été traitée sous cette forme par Aickelin et Dowsland (2000) ou plus récemment par Goodman, Dowsland et Thompson (2007). Pour réaliser ce type de planning, il faut respecter plusieurs conditions : (i) assurer une couverture minimum, (ii) prendre en compte un ensemble de préférences et de demandes, (iii) proposer des plannings cohérents. Le problème MMKP intervient dans la seconde phase pour affecter les infirmières aux différents créneaux. Il s'agit de trouver pour chaque infirmière la suite de créneaux qu'elle va occuper sur une semaine (7 jours et 7 nuits). Ce problème peut être modélisé sous la forme suivante :

$$\text{(MMKP-1)} \begin{cases} \min z = \sum_{i=1}^{n} \sum_{j \in F(i)} c_{ij} x_{ij} & \\ s.c. & \\ \sum_{i \in G(r)} \sum_{j \in F(i)} q_{ir} a_{jk} x_{ij} \geq b_{kr} & \forall r,k \quad (4) \\ \sum_{j \in F(i)} x_{ij} = 1 & \forall i \\ x_{ij} = \{0,1\} & \forall i,j \end{cases}$$

avec $x_{ij} = 1$ si l'infirmière i est affectée à la suite de créneaux j, sinon $x_{ij} = 0$. Le coefficient c_{ij} est une pénalité associée à l'affectation de l'infirmière i à la suite j basée sur les préférences des infirmières, l'historique de l'hôpital et l'attractivité de cette suite de créneaux. L'ensemble $F(i)$ représente les suites de créneaux réalisables pour l'infirmière i. La contrainte (4) oblige à avoir un nombre suffisant d'infirmières de chaque grade r chaque jour k, avec $a_{jk} = 1$ si la suite j couvre le jour de travail k, $G(r)$ l'ensemble des infirmières de grade r, $b_{k,r}$ le nombre minimum d'infirmières de grade r nécessaires le jour k, et $q_{ir} = 1$ si l'infirmière i est de grade r, 0 sinon. Toujours dans le domaine de la planification, Detienne, Péridy, Pinson et Rivreau (2009) ont aussi utilisé le MMKP pour modéliser un problème d'emploi du temps dans un contexte industriel.

Dans le contexte de la chaîne logistique, Basnet et Wilson (2005) ont proposé un modèle mathématique sous forme d'un problème de sac à dos minimisant le nombre d'entrepôts à utiliser pour stocker un ensemble de biens. Un sous-problème est formulé sous la forme d'un MMKP pour prendre en compte les trois contraintes suivantes : (i) la considération de la capacité de l'espace de stockage, (ii) la compatibilité des biens, et le fait que (iii) chaque bien i doit être stocké dans un seul entrepôt j. Soit n (resp. m) le nombre d'objets (resp. entrepôts). A chaque entrepôt j est associé une variable binaire z_j qui prend la valeur 1 si l'entrepôt j est utilisé, 0 sinon. La variable binaire x_{ij} indique le stockage du bien i dans l'entrepôt j (si $x_{ij} = 1$). Le modèle MMKP-2 ci-dessous donne une formulation de ce problème.

L'objectif de ce modèle est de minimiser le nombre d'entrepôts utilisés. La contrainte (5) assure le respect de la capacité de stockage des entrepôts, sachant que b_j est la capacité de l'entrepôt j et a_i est le volume de l'objet i. La contrainte (6) assure le respect des contraintes d'incompatibilité au niveau du stockage des biens dans un même entrepôt : deux objets incompatibles ne doivent pas être stockés dans le même entrepôt. $\overline{C}$ désigne l'ensemble des couples d'objets incompatibles entre eux dans la contrainte (6). La dernière contrainte indique que chaque bien doit être stocké dans un et un seul entrepôt.

$$
\text{(MMKP-2)} \begin{cases} \min \sum_{j=1}^{m} z_j & & & \\ s.c. & & & \\ \sum_{i=1}^{n} a_i x_{ij} \le b_j z_j & j=1,...,m & & (5) \\ x_{ij} + x_{kj} \le 1 & (i,k) \in \overline{C} & j=1,...,m & (6) \\ \sum_{j=1}^{m} x_{ij} = 1 & i=1,...,n & & \\ x \in \{0,1\}^{n \times m}, z \in \{0,1\}^{m} & & & \end{cases}
$$

D'autres exemples d'applications du MMKP moins courantes peuvent aussi être recensées, par exemple dans Chocolaad (1998), Raidl (1999), Romaine (1999) Li, Feng (2001), Lardeux, Knippel et Geffard (2003), ou encore Li, Curry et Boyd (2004).

Dans la section suivante, nous présentons un ensemble de problèmes proches du MMKP, puis quelques propriétés et techniques de fixation, avant de réaliser une étude de la littérature pour ce problème.

2.2 Variantes et Méthodes pour le MMKP

2.2.1 Connexion avec des variantes du sac à dos

Le MMKP peut être considéré comme étant une généralisation d'un ensemble de variantes de problèmes de sac à dos. Quelques problèmes cités dans cette section peuvent être vus comme une réduction ou une généralisation du problème MMKP : le problème du *sac à dos* (**KP**), le problème du *sac à dos multidimensionnel* (**MKP**), le problème du *sac à dos multidimensionnel avec des bornes généralisées* (**GUBMKP**) et le problème du *sac à dos à choix multiples* (**MCKP**). Nous disons qu'un problème P représente une réduction d'un problème P' si le problème P est un sous-problème de P'. La généralisation représente la relation symétrique de la réduction. Notons que si P est une réduction de P' alors P' est une généralisation de P. La Figure 2.1 ci-dessous donne un résumé relationnel du problème MMKP avec les problèmes cités précédemment.

La considération de la fonction objectif du MMKP, des contraintes de choix et d'une seule contrainte de sac à dos génèrent un problème de type MCKP (Sinha et Zoltners (1979)). Le MMKP est donc une généralisation du problème MCKP. De la même façon, si nous considérons la fonction objectif du MMKP et seulement les contraintes de sac à dos, il en

résulte un problème de type MKP (Fréville et Hanafi (2005)). Le problème MMKP se réduit donc à un problème MKP.

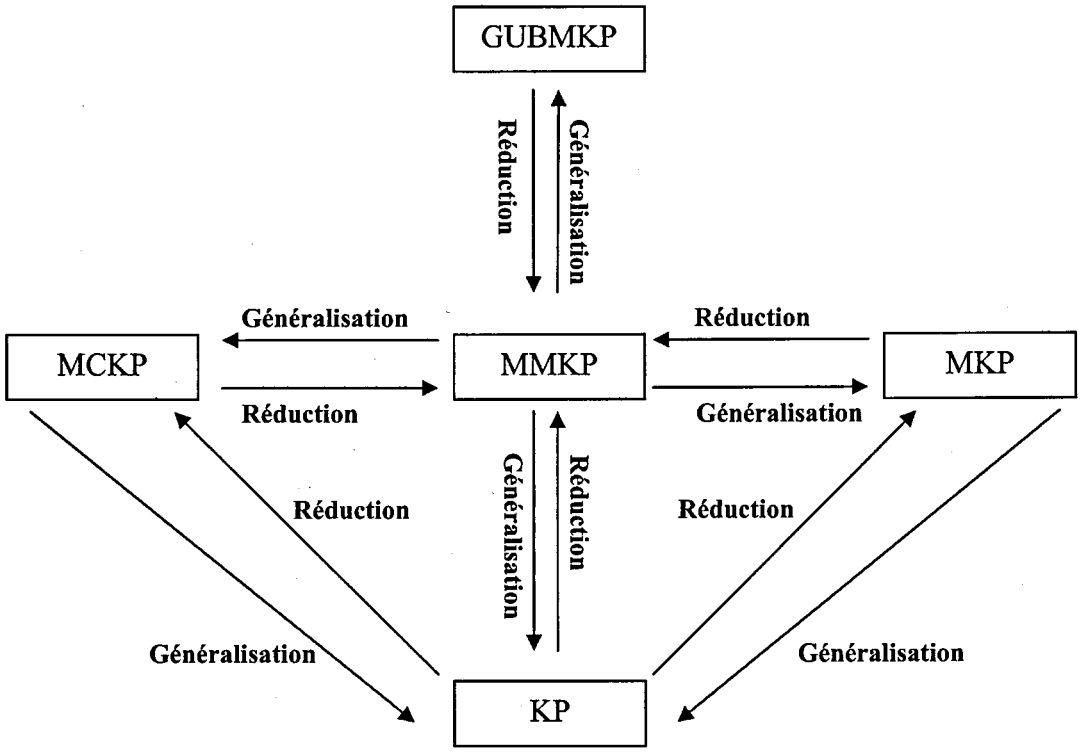


Figure 2.1 : Connexions entre le MMKP et d'autres variantes du sac à dos

Un problème de type MMKP peut aussi être obtenu à partir d'un MKP en considérant n groupes, un pour chacun des n objets du MKP. Un groupe est composé d'un objet du MKP et d'un et un seul objet fictif de coût et consommations nuls permettant d'assurer la sélection d'un seul objet par groupe. La même démarche peut être appliquée pour obtenir les relations entre un problème KP (Martello et Toth (1990)) et les problèmes MCKP et MMKP, puisque le MKP est une généralisation du KP dans lequel nous avons plusieurs contraintes de sac à dos. Finalement il existe un autre problème proche du MMKP qui s'appelle le problème du sac à dos multidimensionnel avec des bornes généralisées (GUBMKP) (Li et Curry (2005)). Pour celui-ci, nous pouvons sélectionner au plus un objet dans chaque groupe. Le GUBMKP est donc un sous-problème du MMKP. Plus formellement, si nous remplaçons les contraintes de choix (2) du MMKP par les inégalités $\sum_{j \in G_i} x_{ij} \leq 1$ et si nous considérons la fonction objectif du MMKP, nous obtenons le GUBMKP. Cela montre que le problème MMKP est une généralisation du problème GUBMKP.

2.2.2 Réduction et Fixation

Cette section est consacrée à la présentation de quelques propriétés classiques du MMKP et à quelques techniques de fixation de variables qui peuvent être appliquées pour réduire la taille du problème traité.

Quelques propriétés peuvent être utilisées dans une phase de prétraitement pour éliminer des variables et/ou des contraintes du MMKP. Nous pouvons ainsi fixer toute variable x_{ij} à la valeur 0 s'il existe $k \in \{1, \ldots, m\}$ pour lequel $a_{ij}^k > b^k$, c'est-à-dire si le choix de l'objet j va forcément violer au moins une contrainte de sac à dos. De plus, nous pouvons remarquer que s'il existe $k \in \{1, \ldots, m\}$ pour lequel $\sum_{i=1}^{n} \min\{a_{ij}^k : j \in G_i\} > b^k$ alors le problème MMKP n'admet pas de solution réalisable. Il est également possible d'éliminer la contrainte de sac à dos k si $\sum_{i=1}^{n} \max\{a_{ij}^k : j \in G_i\} \leq b^k$. Dans ce cas la contrainte k est satisfaite quelle que soit la solution x respectant les contraintes de choix (2). Nous pouvons noter qu'en général ces propriétés sont inefficaces pour les instances difficiles de la littérature. Finalement, il est aussi habituel d'éviter le cas trivial correspondant à sélectionner le meilleur objet de chaque classe en s'assurant d'avoir au moins une valeur de $k \in \{1, \ldots, m\}$ telle que $\sum_{i=1}^{n} a_{i\hat{j}}^k > b^k$ avec $\hat{j} = \arg\max\{c_{ij} : j \in G_i\}$.

D'autres propriétés du MMKP ont été mises en évidence. Ainsi, à partir du principe de simple dominance et de dominance convexe énoncé par Armstrong, Kung, Sinha et Zoltners (1983) pour le problème du sac à dos avec contrainte de choix, nous pouvons dire que l'objet j domine simplement l'objet l dans la classe i si les conditions suivantes sont satisfaites : $a_{ij}^k \leq a_{il}^k$ et $c_{ij} \geq c_{il}$, $k = 1, \ldots, m$. Notons que si la variable associée à l'objet l est dominée simplement par la variable associée à l'objet j dans la même classe i alors la variable x_{il} peut être fixée à 0. De plus, nous disons que l'objet h de la classe i est LP-dominé par les objets j et l (dominances convexes) si

$$a_{ij} < a_{ih} < a_{il},\ c_{ij} < c_{ih} < c_{il} \text{ et } (c_{il} - c_{ih}) / (a_{il} - a_{ih}) \geq (c_{ih} - c_{ij}) / (a_{ih} - a_{ij}).$$

Dans ce cas, $x_{ih} = 0$ dans une solution optimale de la relaxation en continu du MMKP.

Les techniques de fixation de variables, qui consistent à fixer des variables à leurs valeurs optimales via des traitements peu coûteux en temps de calcul, sont utiles pour résoudre efficacement les problèmes d'optimisation. Plusieurs chercheurs (voir par exemple Saunders

et Schinzinger (1970), Balas et Martin (1980), Fayard et Plateau (1981), Gavish et Pirkul (1985), Fréville et Plateau (1986), Osorio, Glover et Hammer (2002), Trick (2003), ou encore Vimont, Boussier et Vasquez (2008)) ont présenté des méthodes de réduction pour le MKP dont certaines sont valides pour un problème 0-1 MIP, et en particulier le MMKP.

Soit un problème linéaire en variables binaires : (P) : $\max\{cx : Ax \leq b, x \in \{0,1\}^n\}$.

Proposition 1 : Soit $\underline{v}(p)$ une borne inférieure du problème P et $\alpha \in \{0, 1\}$, s'il existe $j \in \{1, \ldots, n\}$ tel que $\bar{v}(P \mid x_j = \alpha) \leq \underline{v}(P)$, alors x_j doit être fixée à $1-\alpha$, avec $\bar{v}(P)$ une borne supérieure du problème P.

Preuve : Il est facile de constater que $v(P) = \max\{v(P| x_j = \alpha), v(P| x_j = 1-\alpha)\}$, donc si $\bar{v}(P| x_j = \alpha) \leq \underline{v}(P)$ alors forcément nous avons $v(P) = v(P| x_j = 1-\alpha)$, donc $x^*_j = 1-\alpha$ où x^* est une solution optimale de P. □

La borne supérieure $\bar{v}(P| x_j = \alpha)$ est souvent obtenue en résolvant une relaxation du problème P (voir Section 2.3). La relaxation la plus utilisée dans la littérature est la relaxation en continu. La relaxation en continu du problème P peut s'écrire sous sa forme standard comme suit :

$$(LP) : \max\{cx : Ax + y = b, e \geq x \geq 0, y \geq 0\},$$

où le vecteur y de dimension m correspond aux variables d'écarts. La relaxation en continu LP peut s'écrire dans sa base optimale de la façon suivante :

$$\begin{cases} \max & BS + \sum_{j \in J^0(\bar{x})} \hat{c}_j x_j - \sum_{j \in J^1(\bar{x})} \hat{c}_j (1 - x_j) + \hat{d}y \\ s.c. & \\ & \hat{A}x + \hat{t}y = \hat{b} \\ & x \in [0,1]^n, y \geq 0 \end{cases}$$

où BS est la valeur optimale de LP, $\bar{x}$ est une solution de base optimale de la relaxation, $(\hat{c}, \hat{d})$ le vecteur des coûts réduits correspondant aux variables (x, y) pour la base optimale. La notation ^ indique ici la valeur du coefficient associé dans la base optimale. En supposant que toutes les données sont entières et que nous disposons d'une borne inférieure BI sur la valeur optimale de P, nous avons :

$$BS + \sum_{j \in J^0(\bar{x})} \hat{c}_j x_j - \sum_{j \in J^1(\bar{x})} \hat{c}_j (1 - x_j) + \hat{d}y \geq BI.$$

Puisque les coûts réduits associés à une base optimale sont négatifs (i.e. $\hat{d} \leq 0$) et que les variables d'écarts sont positives, la quantité $\hat{d}y$ est négative. De plus, sachant que $\hat{c}_j < 0$ pour $j \in J^0(\bar{x})$ et $\hat{c}_j > 0$ pour $j \in J^1(\bar{x})$, nous pouvons utiliser la coupe des coûts réduits suivante (Vimont, Boussier et Vasquez (2008)) :

$$\sum_{j \in J^0(\bar{x})} |\hat{c}_j| x_j + \sum_{j \in J^1(\bar{x})} |\hat{c}_j| (1 - x_j) \leq BS - BI \quad (coupe\text{-}1).$$

Proposition 2 : Soit $\bar{x}$ une solution optimale de la relaxation en continu LP, $\hat{c}$ le vecteur des coûts réduits associés, alors :

Si $\bar{x}_j$ est une variable hors base et $|\hat{c}_j| > c\bar{x} - BI$, alors $x_j^* = \bar{x}_j$ où x^* est une solution optimale de P.

Preuve : La preuve provient directement de (*coupe-1*).

Si $j \in J^0(\bar{x})$ et $|\hat{c}_j| > c\bar{x} - BI$ alors $x_j^* = 0 = \bar{x}_j$.

Si $j \in J^1(\bar{x})$ et $|\hat{c}_j| > c\bar{x} - BI$ alors $(1 - x_j^*) = 0$ donc $x_j^* = 1 = \bar{x}_j$. □

Notons que la fixation d'une variable du problème MMKP à 1 implique la fixation de toutes les variables de sa classe pour respecter les contraintes de choix.

En raison de la difficulté du problème MMKP, la plupart des approches proposées pour résoudre ce problème sont basées sur des heuristiques. Dans la section suivante, nous présentons quelques approches efficaces proposées pour résoudre le problème MCKP et le problème MMKP.

2.2.3 Méthodes existantes pour le MCKP et le MMKP

Pour le problème MCKP, il est possible de trouver quelques références décrivant des méthodes exactes. Sinha et Zoltners (1979) ont ainsi proposé un algorithme de type *Branch-and-Bound* pour le MCKP. La relaxation en continu à chaque nœud guide le branchement sur les variables. L'algorithme est testé sur des instances générées aléatoirement. Les résultats numériques montrent que le temps d'exécution augmente plus rapidement avec le nombre de classes qu'avec le nombre de variables par classe. Armstrong, Kung, Sinha et Zoltners (1983) ont proposé des améliorations de cet algorithme en réduisant l'espace mémoire et le temps d'exécution pour les instances de grande taille.

Dyer, Riha et Walker (1995) ont proposé un algorithme hybride combinant programmation dynamique et *Branch-and-Bound* pour résoudre ce problème. Ils appliquent itérativement un algorithme de programmation dynamique sur un problème réduit contenant les k premières contraintes de choix, et appliquent différentes règles d'élimination par dominance et non-faisabilité pour limiter le nombre de solutions partielles. Dans une seconde phase, ils considèrent chaque solution partielle et appliquent un test d'élimination par borne pour ne conserver que la meilleure solution et converger vers une solution optimale du problème. Les résultats numériques, réalisés sur un ensemble d'instances aléatoires, montrent l'intérêt de l'hybridation par rapport à l'utilisation du *Branch-and-Bound* seul.

Pisinger (1995) a proposé un algorithme de partitionnement pour trouver une solution optimale de la relaxation en continu du MCKP. Il a montré comment celui-ci peut être incorporé au sein d'un algorithme de programmation dynamique de sorte qu'un nombre minimal de classes soit énuméré, trié et réduit. A partir de la première solution réalisable obtenue, Pisinger utilise la programmation dynamique pour résoudre le MCKP en ajoutant de nouvelles classes au problème noyau (*core problem* selon la terminologie anglaise) (Balas et Zemel (1980)). Son algorithme a obtenu des résultats références pour des grandes instances comportant jusqu'à 10 000 variables.

Moser, Jokanovic et Shiratori (1997) furent parmi les premiers à s'intéresser à la résolution du MMKP à l'aide d'un algorithme basé sur la relaxation Lagrangienne. Partant d'une solution non réalisable, l'algorithme choisit itérativement l'objet à permuter qui entraîne la plus petite augmentation du multiplicateur et qui rapproche le plus la solution du domaine réalisable. Le processus est répété jusqu'à ce qu'un objet de chaque classe soit choisi ou qu'aucun remplacement ne soit possible. Une phase d'amélioration par échange est ensuite appliquée sur chaque classe. Cette heuristique a été améliorée par Akbar, Ergun et Punnen (2001). Le temps d'exécution de l'algorithme en fait un candidat intéressant pour des applications en temps réel, notamment pour les systèmes multimédias adaptatifs. Egalement dans le contexte multimédia, Khan, Li, Manning et Akbar (2002) ont proposé une heuristique notée (HEU). Ils ont utilisé les concepts proposés par Toyoda (1975) pour le MKP et basés sur l'agrégation des contraintes de ressources pour le choix des objets à placer dans le sac. Ils améliorent la solution en utilisant une procédure d'échange d'objets. HEU a été testée et comparée avec un algorithme de *Branch-and-Bound* utilisant la programmation linéaire et l'algorithme de Moser, Jokanovic et Shiratori (1997). Les résultats obtenus montrent que HEU surpasse les autres algorithmes à la fois au niveau du temps d'exécution et de la qualité des solutions trouvées.

Parra-Hernandez et Dimopoulos (2002) ont proposé une extension de l'approche de Pirkul (1987) pour le MKP afin de résoudre le MMKP. Ils relâchent les contraintes de choix multiples dans une première phase, ce qui ramène le MMKP à un GUBMKP. Ils utilisent ensuite une relaxation Lagrangienne et trient les variables selon leur efficacité, au sens classique des problèmes de sac à dos. Ils construisent ainsi une solution initiale non nécessairement réalisable, et la modifient ensuite à l'aide d'une recherche locale. Cet algorithme trouve de meilleures solutions que celles obtenues par Akbar, Ergun et Punnen (2001), avec cependant un temps de calcul plus important.

Hifi, Michrafy et Sbihi (2004) ont proposé plusieurs heuristiques constructives pour résoudre le MMKP. Ils ont aussi proposé une méthode de recherche locale guidée qui a permis l'obtention de meilleurs résultats que Khan, Li, Manning et Akbar (2002). Hifi, Michrafy et Sbihi (2006) ont également proposé une recherche locale réactive qui a permis une amélioration des résultats obtenus par la méthode précédente. Nous reviendrons plus en détail sur ces deux approches dans la Section 2.4 qui est consacrée aux recherches locales pour le MMKP.

Akbar, Rahman, Kaykobad, Manning et Shoja (2006) ont transformé le MMKP en agrégeant les contraintes multidimensionnelles de sac à dos en une seule contrainte en employant un vecteur de pénalité. Ils présentent une heuristique qui construit les *enveloppes convexes* pour réduire l'espace de recherche et trouver une solution approximative du MMKP. Ces enveloppes convexes sont des polygones qui représentent des petits sous-ensembles convexes de l'ensemble des solutions réalisables. Les résultats obtenus sont intéressants pour des instances non corrélées par rapport aux algorithmes de Moser, Jokanovic et Shiratori (1997), et de Akbar, Ergun et Punnen (2001).

Finalement, Cherfi et Hifi (2008) ont proposé une des heuristiques les plus efficaces pour résoudre le problème MMKP. Cette heuristique est basée sur les techniques de la génération de colonnes pour définir un processus de recherche locale efficace. L'idée originale de l'approche est d'utiliser une procédure de type *Branch-and-Bound* sur un problème de petite taille et de résoudre quelques nœuds par un processus hybridant génération de colonnes, heuristique d'arrondi et recherche locale. La méthode est testée numériquement sur un ensemble d'instances publiées dans la littérature et comparée aux résultats atteints par CPLEX et d'autres heuristiques récentes. Pour ces instances, la méthode proposée améliore 21 bornes inférieures.

Précédemment, nous avons présenté dans un ordre chronologique un ensemble d'heuristiques pour résoudre les problèmes MMKP et MCKP. Le problème MMKP peut aussi

être résolu exactement avec un algorithme de *Branch-and-Bound classique* utilisé pour résoudre les problèmes linéaires mixtes. Cependant ce type d'algorithme éprouve rapidement de grandes difficultés pour obtenir une solution optimale du problème et démontrer l'optimalité de cette solution (voir Section 2.5). Dans la littérature, nous trouvons d'autres algorithmes exacts pour le problème MMKP qui exploitent la structure spéciale de ce problème. Dans l'algorithme de Sbihi (2007), la borne supérieure considérée pour le branchement est calculée à partir d'une relaxation appelée auxiliaire et basée sur le principe de la relaxation surrogate des contraintes de sac à dos (voir Section 2.3 pour plus de détails sur cette relaxation). Le branchement dans l'arbre se fait en respectant la stratégie *meilleure d'abord.* Cet algorithme est capable de résoudre des instances assez grandes comprenant jusqu'à 50 classes, 15 objets par classes (soit 750 variables) et 10 contraintes de sac à dos. Dans l'algorithme de Razzazi et Ghasemi (2008) la valeur optimale de la relaxation continue de la relaxation surrogate est considérée comme étant une borne supérieure du problème MMKP. La stratégie de branchement considérée est en *profondeur d'abord* dans le but de minimiser l'espace mémoire consommé. D'après l'étude numérique présentée, cet algorithme domine l'algorithme de Sbihi (2007) sur un ensemble d'instances non corrélées et corrélées et comportant jusqu'à 1000 objets et 5 contraintes de sac à dos.

Une étude récente et détaillée de la littérature du MMKP peut être trouvée dans Hiremath (2008).

2.3 Relaxations du MMKP

La résolution de problèmes d'optimisation difficiles ne permet pas toujours d'obtenir une solution optimale. Une des motivations dans ce cas réside dans l'encadrement de la valeur optimale. Un bon encadrement de la valeur optimale fournit un ensemble d'informations très utiles pour réduire le problème en fixant des variables ou restreindre l'espace de recherche. L'encadrement de la valeur optimale peut être amélioré par l'ajout d'un ensemble de contraintes valides au problème et consiste en une borne supérieure et inférieure de la valeur optimale. Plus la différence entre la borne supérieure et la borne inférieure est faible, plus les informations sur une solution optimale sont précises. Dans cette section, nous allons présenter un ensemble de techniques dont chacune d'elles permet d'obtenir une borne supérieure plus ou moins fine du MMKP.

La relaxation en continu présentée dans la Section 2.1.1 est la technique la plus appliquée dans la littérature pour l'obtention de bornes supérieures pour des problèmes linéaires. Elle est

considérée comme étant un problème polynomial. Sa résolution est souvent faite par l'algorithme du simplexe. Ce dernier est un algorithme exponentiel qui a montré, en pratique, une efficacité lors de la résolution des problèmes linéaires continus (Spielman et Teng (2001)). La résolution de la relaxation en continu fournit aussi des informations importantes sur les variables de branchement dans l'arbre de *Branch-and-Bound* pour la résolution des problèmes en nombres entiers mixtes.

2.3.1 Relaxation et Décomposition Lagrangiennes

La *relaxation Lagrangienne* est une technique souvent utilisée en recherche opérationnelle (Gondran et Minoux (1995)). Elle est employée pour calculer une borne supérieure sur la valeur optimale du problème étudié (dans le cas d'un problème de maximisation). Elle consiste à déplacer une partie des contraintes, souvent les plus difficiles du problème, dans la fonction objectif en considérant une pénalité pour chaque contrainte. La pénalisation de ces contraintes se fait à l'aide des multiplicateurs de Lagrange. Soit par exemple le problème d'optimisation :

$$(Z)\ \max\{f(x) |\ g_1(x) \le b_1, g_2(x) \le b_2\}.$$

Si nous supposons ici que les contraintes difficiles du problème sont $g_1(x) \le b_1$, alors la relaxation Lagrangienne associée est :

$$L(\lambda)\ \max\{L(x,\lambda) |\ g_2(x) \le b_2\},$$

avec $L(x, \lambda) = f(x) - \lambda(b_1 - g_1(x))$ appelée fonction Lagrangienne du problème d'origine et associée au multiplicateur λ de dimension m (si nous supposons que m est la dimension du vecteur b_1). Le dual Lagrangien associé est :

$$(L)\ \min\{v(L(\lambda)) |\ \lambda \in \Re_+^m\}.$$

Lors de la recherche d'une borne supérieure du problème d'origine en appliquant la relaxation Lagrangienne, il nous faut trouver un multiplicateur λ^* optimal du dual Lagrangien L. Pour trouver λ^*, nous pouvons utiliser la méthode du sous-gradient (Gondran et Minoux (1995)). Cette méthode itérative consiste à mettre à jour à chaque itération les multiplicateurs de Lagrange. Cette mise à jour se fait selon l'équation suivante : $\lambda^{k+1} = \lambda^k + t^k \gamma(\lambda^k)$, où λ^k indique le vecteur des multiplicateurs à la $k^{ème}$ itération. Le vecteur $\gamma(\lambda^k)$ est un sous-gradient de la fonction $v(L(\lambda))$ au point λ^k. Notons que si les pas de déplacement t^k sont choisis

convenablement (par exemple, $\sum_k t^k \to +\infty$ et $t^k \to 0$ quand $k \to +\infty$) alors le processus converge vers un multiplicateur optimal.

Plusieurs relaxations Lagrangiennes peuvent être considérées selon l'ensemble des contraintes déplacées dans la fonction objectif. Nous en présentons trois dans la suite pour le MMKP.

La première relaxation Lagrangienne pour le MMKP consiste à relaxer les contraintes de sac à dos. Cette relaxation Lagrangienne $L^1(\lambda)$ associée au multiplicateur $\lambda \in \Re_+^m$ est présentée par le modèle mathématique suivant :

$$L^1(\lambda)\begin{cases} \max \sum_{i=1}^{n} \sum_{j\in G_i} c_{ij}x_{ij} - \sum_{k=1}^{m}\lambda_k\left(\sum_{i=1}^{n}\sum_{j\in G_i} a_{ij}^k x_{ij} - b^k\right) \\ s.c. \\ \qquad \sum_{j\in G_i} x_{ij} = 1 \qquad \forall i = 1,...,n \\ \qquad x_{ij} \in \{0,1\} \qquad \forall i = 1,...,n, j \in G_i \end{cases}$$

En développant la fonction objectif de $L^1(\lambda)$, nous obtenons :

$$L^1(\lambda)\begin{cases} \max \sum_{i=1}^{n} \sum_{j\in G_i}\left(c_{ij} - \sum_{k=1}^{m}\lambda_k a_{ij}^k\right)x_{ij} + \sum_{k=1}^{m}\lambda_k b^k \\ s.c. \\ \qquad \sum_{j\in G_i} x_{ij} = 1 \qquad \forall i = 1,...,n \\ \qquad x_{ij} \in \{0,1\} \qquad \forall i = 1,...,n, j \in G_i \end{cases}$$

Une solution optimale de ce problème est immédiate. Il suffit de choisir dans chaque classe i l'objet auquel est associé le meilleur profit. Plus formellement, $x_{i\hat{j}} = 1$ $\forall i = 1,\ldots, n$ avec $\hat{j} = \arg\max\left\{c_{ij} - \sum_{k=1}^{m}\lambda_k a_{ij}^k : j \in G_i\right\}$. La valeur optimale de cette relaxation Lagrangienne peut donc être déterminée explicitement par :

$$v\left(L^1(\lambda)\right) = \sum_{k=1}^{m}\lambda_k b^k + \sum_{i=1}^{n}\max\left\{c_{ij} - \sum_{k=1}^{m}\lambda_k a_{ij}^k : j \in G_i\right\}.$$

Cette relaxation est très utilisée dans la littérature car le problème Lagrangien associé est facile à résoudre.

Lorsque nous relaxons les contraintes de choix à la place des contraintes de sac à dos, le problème qui en résulte est plus difficile à résoudre. La relaxation Lagrangienne $L^2(\lambda)$

associée au multiplicateur $\lambda \in \Re^n$ et relative aux contraintes de choix est présentée par le modèle suivant :

$$L^2(\lambda)\begin{cases} \max \sum_{i=1}^{n} \sum_{j\in G_i} c_{ij}x_{ij} - \sum_{k=1}^{m} \lambda_k \left(1 - \sum_{j\in G_i} x_{ij}\right) \\ s.c. \\ \quad \sum_{i=1}^{n} \sum_{j\in G_i} a_{ij}^k x_{ij} \le b^k \qquad \forall k = 1,...,m \\ \quad x_{ij} \in \{0,1\} \qquad \forall i = 1,...,n, j \in G_i \end{cases}$$

La plus grande difficulté du problème $L^2(\lambda)$ se justifie par le fait qu'il se présente sous la forme d'un MKP.

En décomposant les contraintes de choix en deux inégalités, c'est-à-dire en remplaçant la contrainte de choix $\sum_{j\in G_i} x_{ij} = 1$ par $\sum_{j\in G_i} x_{ij} \le 1$ et $\sum_{j\in G_i} x_{ij} \ge 1$, la relaxation Lagrangienne $L^3(\lambda)$ associée au multiplicateur $\lambda \in \Re_+^n$ et à la contrainte $\sum_{j\in G_i} x_{ij} \ge 1$ est présentée par le modèle mathématique suivant :

$$L^3(\lambda)\begin{cases} \max \sum_{i=1}^{n} \sum_{j\in G_i} c_{ij}x_{ij} + \sum_{k=1}^{m} \lambda_k \left(1 - \sum_{j\in G_i} x_{ij}\right) \\ s.c. \\ \quad \sum_{i=1}^{n} \sum_{j\in G_i} a_{ij}^k x_{ij} \le b^k \qquad \forall k = 1,...,m \\ \quad \sum_{j\in G_i} x_{ij} \le 1 \qquad \forall i = 1,...,n \\ \quad x_{ij} \in \{0,1\} \qquad \forall i = 1,...,n, j \in G_i \end{cases}$$

Notons que cette relaxation $L^3(\lambda)$ est un problème GUBMKP. Elle représente la relaxation la plus difficile à résoudre par rapport aux deux précédentes $L^1(\lambda)$ et $L^2(\lambda)$.

Les duaux Lagrangiens associés aux trois relaxations Lagrangiennes $L^k(\lambda)$ pour k = 1, 2, 3 sont définis par :

$$(L^k) \min\{ v(L^k(\lambda)) : \lambda \in \Lambda^k\} \text{ où } \Lambda^1 = \Re_+^m, \Lambda^2 = \Re^n, \Lambda^3 = \Re_+^n.$$

Nous avons présenté précédemment trois relaxations Lagrangiennes du problème MMKP dans un ordre croissant de difficulté. La qualité des bornes fournies par chaque relaxation est explicitée dans la proposition suivante.

Proposition 3 : $v(\text{MMKP}) \le v(L^3) \le v(L^2) \le v(L^1) = v(LP(\text{MMKP}))$.

Preuve : Il est bien connu qu'un dual Lagrangien fournit une borne supérieure et que cette dernière est inférieure à la borne générée par la relaxation en continu, i.e.

$$v(\text{MMKP}) \leq v(L^k) \leq v(LP(\text{MMKP})), \text{ pour } k = 1, 2, 3.$$

De plus, d'après Geoffrion (1974), si la relaxation possède la propriété d'intégrité, alors le dual associé fournit la même borne que la relaxation en continu. Donc, puisque la relaxation $L^1(\lambda)$ possède la propriété d'intégrité, nous avons :

$$v(L^1) = v(LP(\text{MMKP})).$$

Il nous reste à démontrer que $v(L^3(\lambda)) \leq v(L^2(\lambda))$. Pour démontrer cette inégalité, il suffit de constater que le problème $L^2(\lambda)$ représente une relaxation du problème $L^3(\lambda)$. En effet, en relaxant la contrainte $\sum_{j \in G_i} x_{ij} \leq 1$ du problème $L^3(\lambda)$, nous obtenons :

$$L'^3(\lambda, \lambda') \begin{cases} \max \sum_{i=1}^{n} \sum_{j \in G_i} c_{ij} x_{ij} + \sum_{k=1}^{m} \lambda_k \left(1 - \sum_{j \in G_i} x_{ij}\right) - \sum_{k=1}^{m} \lambda'_k \left(1 - \sum_{j \in G_i} x_{ij}\right) \\ s.c. \\ \quad \sum_{i=1}^{n} \sum_{j \in G_i} a_{ij}^k x_{ij} \leq b^k \qquad \forall k = 1,\ldots,m \\ \quad x_{ij} \in \{0,1\} \qquad \forall i = 1,\ldots,n, j \in G_i \end{cases}$$

avec λ et $\lambda' \in \Re_+^n$. La fonction objectif du problème $L'^3(\lambda, \lambda')$ peut aussi être écrite de la manière suivante : $\sum_{i=1}^{n} \sum_{j \in G_i} c_{ij} x_{ij} + \sum_{k=1}^{m} (\lambda_k - \lambda'_k) \left(1 - \sum_{j \in G_i} x_{ij}\right)$. En remplaçant $\lambda - \lambda'$ par λ'' avec $\lambda'' \in \Re^n$, nous obtenons la relaxation $L'^3(\lambda'')$ qui est équivalente à $L^2(\lambda)$. Donc le problème $L^2(\lambda)$ représente une relaxation du problème $L^3(\lambda)$. □

Une autre technique liée à la relaxation Lagrangienne est la décomposition Lagrangienne. Elle consiste à transformer le problème initial, généralement en dupliquant les variables, afin d'appliquer la relaxation Lagrangienne dans le but de minimiser la difficulté du problème (voir Guignard et Kim (1987) par exemple).

2.3.2 Relaxations Surrogate et Composite

Une contrainte surrogate (Glover (1965)) est une inégalité déduite par les contraintes du problème traité. Elle est conçue pour saisir des informations utiles qui ne peuvent pas être directement obtenues des contraintes du problème. La relaxation surrogate associée au multiplicateur $\mu \in \Re_+^m$, obtenue en agrégeant les contraintes de sac à dos en une seule

contrainte surrogate, et employée pour fournir une borne supérieure pour le MMKP est présentée par le modèle mathématique suivant :

$$S(\mu)\begin{cases}\max \sum_{i=1}^{n} \sum_{j\in G_i} c_{ij}x_{ij} \\ s.c. \\ \quad \sum_{k=1}^{m}\mu_k\left(\sum_{i=1}^{n}\sum_{j\in G_i} a_{ij}^k x_{ij}\right) \le \sum_{k=1}^{m}\mu_k b^k \\ \quad \sum_{j\in G_i} x_{ij} = 1 \qquad \forall i = 1,\ldots,n \\ \quad x_{ij} \in \{0,1\} \qquad \forall i = 1,\ldots,n, j \in G_i\end{cases}$$

Notons que la relaxation $S(\mu)$ se présente sous la forme d'un problème MCKP. Le dual surrogate associé est :

$$\text{(S)}\ \min\{v(S(\mu))|\ \mu \in \Re_+^m\}.$$

La relaxation surrogate associée au cas où $\mu_k = \mu$ pour $k = 1,\ldots, m$, nommée relaxation auxiliaire, a été utilisée par Sbihi (2007). La relaxation surrogate a souvent été utilisée pour résoudre des problèmes de sac à dos (voir par exemple Fréville et Plateau (1993), Fréville et Hanafi (2005)).

Notons qu'un autre type de relaxation, appelée relaxation composite, peut aussi être trouvée dans la littérature (Greenberg et Pierskalla (1970)). Cette relaxation combine à la fois la relaxation Lagrangienne et la relaxation surrogate. Un exemple de relaxation composite C(λ,μ) pour le MMKP associée aux multiplicateurs $\lambda \in \Re_+^n$ et $\mu \in \Re_+^m$ est donné ci-après.

$$\text{C}(\lambda,\mu)\begin{cases}\max \sum_{i=1}^{n} \sum_{j\in G_i} c_{ij}x_{ij} + \sum_{k=1}^{m}\lambda_k\left(1 - \sum_{j\in G_i} x_{ij}\right) \\ s.c. \\ \quad \sum_{k=1}^{m}\mu_k\left(\sum_{i=1}^{n}\sum_{j\in G_i} a_{ij}^k x_{ij}\right) \le \sum_{k=1}^{m}\mu_k b^k \\ \quad \sum_{j\in G_i} x_{ij} \le 1 \qquad \forall i = 1,\ldots,n \\ \quad x_{ij} \in \{0,1\} \qquad \forall i = 1,\ldots,n, j \in G_i\end{cases}$$

Le dual composite (C) associé est défini par : $(C)\ \min_{\lambda\ge 0,\mu\ge 0} v(\text{C}(\lambda,\mu))$.

2.3.3 Relaxations en Nombres Entiers Mixtes et Semi-Continue

La relaxation en continu LP(MMKP) relâche totalement les contraintes d'intégrité. Dans cette section, nous considérons la relaxation en nombres entiers mixtes relative au problème MMKP et à un sous-ensemble $J \subseteq G$ introduite par Glover (2006) et Hanafi et Wilbaut (2006). Cette relaxation est définie comme suit :

$$(\text{MIP (MMKP, } J)) \begin{cases} \max \sum_{i=1}^{n} \sum_{j \in G_i} c_{ij} x_{ij} \\ s.c. \\ \sum_{i=1}^{n} \sum_{j \in G_i} a_{ij}^k x_{ij} \le b^k & \forall k = 1,...,m \\ \sum_{j \in G_i} x_{ij} = 1 & \forall i = 1,...,n \\ x_{ij} \in \{0,1\} & \forall i = 1,...,n, j \in G_i, (i,j) \in J \\ x_{ij} \in [0,1] & \forall i = 1,...,n, j \in G_i, (i,j) \in G - J \end{cases}$$

Cette relaxation consiste à forcer un sous-ensemble J des variables à être binaire dans une solution optimale. La taille de ce sous-ensemble doit en pratique être petite par rapport à $|G|$, et les autres variables sont continues. La justification de cette relaxation provient de la proposition évidente suivante :

Proposition 4 : Soient P une instance du MMKP, J et J' deux sous-ensembles de G avec $J' \subset J \subset G$, nous avons:

$$v(\text{MIP(P,}G)) = v(\text{P}) \le v(\text{MIP(P,}J)) \le v(\text{MIP(P,}J')) \le v(\text{MIP(P,}\varnothing)) = v(\text{LP(P)}).$$

Comme nous l'avons dit précédemment la relaxation en continu est souvent la relaxation la moins coûteuse pour fournir une borne supérieure. Cependant pour des instances difficiles, cette borne n'est pas de bonne qualité. La relaxation en nombres entiers mixtes, qui relaxe les contraintes d'intégrité sur seulement un sous-ensemble de variables, donne en général une meilleure borne mais elle peut être coûteuse si le nombre de variables entières est grand.

Nous proposons une autre relaxation, dite semi-continue (**RSC**) qui relaxe aussi partiellement les contraintes d'intégrité dans le but d'obliger les variables à obtenir des valeurs *proches* de 0 ou de 1. Autrement dit, l'objectif est de trouver dans la solution de la relaxation des variables x_{ij} qui prennent des valeurs soit dans l'intervalle [0, α_{ij}], soit dans l'intervalle [1- α_{ij}, 1], avec α_{ij} appartenant à [0, 1/2], pour $i = 1,..., n$ et $j = 1,..., n_i$. Nous allons

présenter dans la Proposition 5 la contrainte que nous générons pour une variable x_{ij} afin d'assurer son appartenance à un des deux intervalles $[0, \alpha_{ij}]$ ou $[1-\alpha_{ij}, 1]$.

Proposition 5: Soient $x \in [0,1]$ et $\alpha \in [0, 1/2]$.

$$x \in [0, \alpha] \cup [1-\alpha, 1] \Leftrightarrow 0 \leq x - (1-\alpha) y \leq \alpha, y \in \{0,1\}.$$

Preuve : La preuve est triviale, il suffit de constater que la variable binaire $y = 0$ si $x \in [0, \alpha]$ et $y = 1$ si $x \in [1-\alpha, 1]$. □

La Proposition 5 nous permet de relaxer les contraintes d'intégrité des variables binaires du problème en les remplaçant par les contraintes : $0 \leq x_{ij} - (1 - \alpha_{ij}) y_{ij} \leq \alpha_{ij}$ avec $y_{ij} \in \{0,1\}$ et $\alpha_{ij} \in [0, 1/2]$ pour $i = 1,..., n$ et $j = 1,..., n_i$. La relaxation semi-continue RSC associée au multiplicateur $\alpha \in [0, \frac{1}{2}]^{n'}$ avec $n' = \sum_{i=1}^{n} n_i$, est définie comme suit :

$$\text{RSC}(\alpha) \begin{cases} \max \sum_{i=1}^{n} \sum_{j \in G_i} c_{ij} x_{ij} \\ s.c. \\ \quad \sum_{i=1}^{n} \sum_{j \in G_i} a_{ij}^k x_{ij} \leq b^k \qquad \forall k = 1,...,m \\ \quad \sum_{j \in G_i} x_{ij} = 1 \qquad \forall i = 1,...,n \\ \quad 0 \leq x_{ij} - (1-\alpha_{ij}) y_{ij} \leq \alpha_{ij} \quad \forall i = 1,...,n, j \in G_i \quad (7) \\ \quad y_{ij} \in \{0,1\} \qquad \forall i = 1,...,n, \ j \in G_i \end{cases}$$

La valeur optimale de la relaxation semi-continue RSC(α) fournit une borne supérieure du problème MMKP. La qualité de cette borne dépend fortement des valeurs associées aux paramètres α_{ij}. Dans la Proposition 6, nous présentons le lien entre la borne fournie par la relaxation semi-continue RSC(α) et celle de la relaxation en nombres entiers mixtes.

Proposition 6:

i) Pour tout $\alpha \in [0, \frac{1}{2}]^{n'}$, nous avons

$$v(\text{RSC}(0)) = v(\text{MMKP}) \leq v(\text{RSC}(\alpha)) \leq v(\text{RSC}(e/2)) = v(\text{LP}(\text{MMKP})).$$

ii) Si $0 \leq \alpha \leq \beta \leq e/2$ alors $v(\text{RSC}(\alpha)) \leq v(\text{RSC}(\beta))$.

iii) Soit $J \subseteq G$, nous avons $v(\text{MIP}(\text{MMKP}, J)) = v(\text{RSC}(\alpha))$, où $\alpha_{ij} = 0$ pour $(i, j) \in J$ et $\alpha_{i'j'} = 1/2$ pour $(i', j') \notin J$, et inversement : si $\alpha_{ij} \in \{0, \frac{1}{2}\}$ pour (i, j), alors $v(\text{RSC}(\alpha))$ correspond à une relaxation MIP du MMKP.

Preuve :

i) D'après la Proposition 5, la contrainte (7) exprime que $x_{ij} \in [0, \alpha_{ij}] \cup [1- \alpha_{ij}, 1]$. Donc si $\alpha = 0$ (i.e. $\alpha_{ij} = 0$) la contrainte (7) est équivalente à $x_{ij} \in \{0, 1\}$. Par conséquent si tous les paramètres α_{ij} sont nuls alors toutes les variables x_{ij} de RSC(α) sont binaires et donc le problème RSC(α) est équivalent au problème initial MMKP (i.e. v(RSC(0)) = v(MMKP)).

De la même manière, si $\alpha = e/2$ (i.e. $\alpha_{ij} = ½$) alors la contrainte (7) devient $x_{ij} \in [0, ½] \cup [1- ½, 1]$, ce qui est équivalent à $x_{ij} \in [0, 1]$. Par conséquent, nous avons v(RSC($e/2$)) = v(LP(MMKP)). Finalement, pour tout $\alpha_{ij} \in [0, ½]$, nous avons $\{0, 1\} \subseteq [0, \alpha_{ij}] \cup [1- \alpha_{ij}, 1] \subseteq [0, 1]$. Ce qui implique que RSC(α) est une relaxation du MMKP et que LP(MMKP) est une relaxation de RSC(α). Donc nous avons :

$$v(\text{MMKP}) \leq v(\text{RSC}(\alpha)) \leq v(\text{LP(MMKP)}).$$

ii) Il est trivial que si $0 \leq \alpha_{ij} \leq \beta_{ij} \leq ½$ nous avons l'inclusion suivante : $[0, \alpha_{ij}] \cup [1- \alpha_{ij}, 1] \subseteq [0, \beta_{ij}] \cup [1- \beta_{ij}, 1]$. Ce qui implique que le problème RSC(β) est une relaxation du problème RSC(α). Donc v(RSC(α)) $\leq v$(RSC(β)) si $\alpha \leq \beta$.

iii) Cette propriété est immédiate d'après les propriétés précédentes puisque l'on a $x_{ij} \in \{0, 1\}$ si $\alpha_{ij} = 0$ et $x_{ij} \in [0, 1]$ si $\alpha_{ij} = ½$. □

Ainsi plus les valeurs des paramètres α_{ij} sont petites et plus le problème RSC(α) converge vers le problème initial MMKP. Dès lors, le but est de trouver une valeur pour α_{ij} de façon à éviter la difficulté de l'instance et de fournir une "bonne" borne supérieure en cherchant des valeurs pour les variables x_{ij} très proches de 1 ou 0. Les paramètres α_{ij} nous permettent d'éviter la sensibilité et la difficulté des instances quand la décision binaire devient très difficile. Pour réduire le nombre de paramètres de la relaxation semi-continue RSC(α), nous supposerons dans la suite que $\alpha_{ij} \in \{0, \alpha_0, ½\}$ pour tout (i, j), avec $\alpha_0 \in]0, ½[$.

2.4 Recherche Locale pour le MMKP

Une recherche locale (**RL**) est un processus qui emploie un ensemble de déplacements (mouvements) pour transformer une solution en une autre en exploitant une fonction d'évaluation pour mesurer l'attractivité de ces déplacements. Une RL commence à partir d'une solution initiale (réalisable ou non) et se déplace d'une solution à une autre dans le voisinage courant selon quelques règles définies, jusqu'à obtenir une solution finale. A chaque itération, une nouvelle solution x' est choisie dans le voisinage de la solution courante

x. Le voisinage $V(x)$ d'une solution x est un sous-ensemble de solutions voisines de x. Les voisins de x sont les solutions qui peuvent être obtenues à partir de x par un déplacement élémentaire. Les algorithmes de recherche locale en optimisation combinatoire se caractérisent généralement par plusieurs composants : i) définition de la solution, ii) définition du voisinage d'une solution ainsi que des règles définissant les mouvements possibles avec les fonctions d'évaluation d'un voisin, iii) liste candidate.

La façon la plus commune de choisir une nouvelle solution x' dans le voisinage de x est de tirer au hasard (une des) la meilleure(s) ; c'est-à-dire, une solution $x' \in V(x)$ avec $cx' \geq cy$, $\forall\, y \in V(x)$. x' devient alors la nouvelle solution courante si elle améliore x, c'est-à-dire si $cx' > cx$. Sinon, la recherche est arrêtée. Cette stratégie est habituellement appelée une stratégie de *descente* ou de *plus forte descente*. Une description sous-forme de pseudo-code est donnée dans l'Algorithme 2.1.

Algorithme 2.1 : Algorithme de Descente

Sélectionner une solution initiale x; **Stop** = Faux ;
tant que Non Stop **faire**
    Sélectionner une meilleure solution x' dans le voisinage $V(x)$ (i.e. $cx' > cx$) ;
    si x' existe **alors** $x = x'$, **sinon Stop** = Vrai ; **fin si**
fin tant que

Un des inconvénients majeurs de la recherche locale est le risque de bloquer la recherche dans un optimum dit local. Celui-ci correspond à un point qui ne peut être amélioré par des mouvements simples mais qui ne correspond pas nécessairement à un optimum global. Ainsi l'utilisation d'une recherche locale seule est souvent limitée. Plusieurs stratégies ont été proposées pour permettre au processus de recherche de sortir d'un optimum local. Elles consistent la plupart du temps à autoriser un mouvement dégradant selon une décision probabiliste ou l'historique de la recherche. Les métaheuristiques guident la recherche locale à explorer un espace de recherche au-delà d'un optimum local tel que le Recuit Simulé (Siarry et Dreyfus (1989)) et la Recherche Tabou (Hansen (1986), Glover (1989), Glover (1990), Glover et Laguna (1997)). Ces métaheuristiques sont des stratégies de recherche locale explicitement conçues pour éviter une telle situation.

La structure particulière du problème MMKP, caractérisée par les contraintes de choix, a été exploitée dans différentes techniques de recherche locale dans la littérature. Nous présentons ici quelques procédures existantes basées sur la recherche locale, avant de

présenter notre procédure qui se concentre sur la recherche de la meilleure solution dans le voisinage d'une solution courante non nécessairement réalisable.

Parmi les approches existantes pour le MMKP, nous en avons retenu deux récentes et performantes. La recherche locale guidée (GLS pour *Guided Local Search*), est l'une des approches récentes considérée comme étant une métaheuristique pour résoudre des problèmes d'optimisation combinatoire. Elle a été introduite pour la première fois par Voudouris et Tsang (1995) pour résoudre des problèmes de satisfaction de contraintes. Elle peut être vue comme une variante de la recherche Tabou puisqu'elle emploie la notion de mémoire pour contrôler le processus de recherche comme celle opérée dans la recherche Tabou. Même si l'idée générale de la méthode est semblable à d'autres algorithmes d'optimisation combinatoire, des différences peuvent être mises en évidence. Par exemple, la recherche Tabou se caractérise entre autres par des interdictions de mouvements généralement implémentées sous forme de listes (dites listes Tabou). Mais, contrairement à la GLS, ces interdictions se réfèrent uniquement aux solutions et sous forme de pénalisation de la fonction objectif. De plus, elles peuvent être très restrictives et uniquement relâchées par un critère d'aspiration dans la recherche Tabou. Hifi, Michrafy et Sbihi (2004) ont proposé une GLS pour le MMKP dans laquelle la trajectoire des solutions est guidée par la fonction de coût avec une pondération qui pénalise la visite de solutions proches de celles déjà visitées. Les résultats montrent que cette méthode domine d'autres méthodes efficaces pour ce problème tel que l'algorithme de Khan, Li, Manning et Akbar (2002).

La recherche locale peut aussi être complétée par un processus, dit réactif, qui s'appuie sur l'historique de la recherche pour augmenter son efficacité. Hifi, Michrafy et Sbihi (2006) ont proposé une approche basée sur une recherche locale réactive pour améliorer les résultats obtenus par la méthode précédente. Le schéma général de l'approche peut être résumé en trois points : i) générer une solution initiale, ii) construire un voisinage et appliquer une stratégie de choix du voisin, iii) perturber la recherche et construire la nouvelle solution courante. Cette dernière phase vise à permettre au processus de sortir des optima locaux et à introduire de la diversité dans la recherche. Un composant éliminant la visite de solutions déjà évaluées est également intégré dans la méthode. Les résultats montrent une nette amélioration par rapport aux algorithmes précédents sur un ensemble d'instances existantes.

La définition du voisinage est un élément clé de la recherche locale. Nous pouvons dire qu'une solution x' est voisine d'une solution x si la distance entre les deux solutions ne dépasse pas une borne donnée. Plus formellement pour le MMKP nous définissons le voisinage $V_q(x)$ de la solution x par :

$$V_q(x) = \{ x' \in \{0, 1\}^{n'} : \delta(x, x') = \sum_{i=1}^{n} \sum_{j \in G_i} |x_{ij} - x'_{ij}| \leq q \}, \text{ avec } n' = \sum_{i=1}^{n} |G_i|.$$

La proposition suivante permet d'exploiter les contraintes de choix du problème MMKP pour réduire le voisinage d'une solution courante.

Proposition 7 : Si les deux solutions binaires x et x' vérifient les contraintes de choix (2), alors la distance de *Hamming* $\delta(x, x')$ peut s'écrire comme suit :

$$\delta(x,x') = \sum_{i=1}^{n} \delta_i(x,x'), \text{ avec } \delta_i(x,x') = \left(\sum_{j \in G_i} |x_{ij} - x'_{ij}| \right) \in \{0,1\}.$$

Preuve : Puisque les deux solutions x et x' sont binaires, la distance partielle $\delta_i(x,x') = \sum_{j \in G_i} |x_{ij} - x'_{ij}|$ peut être linéarisée comme suit (Spielberg et Guignard (2000), Wilbaut et Hanafi (2009)) :

$$\delta_i(x,x') = \sum_{j \in G_i} x_{ij}(1 - x'_{ij}) + x'_{ij}(1 - x_{ij}) \qquad \text{(8-a)}.$$

En développant l'équation (8-a), nous obtenons

$$\delta_i(x,x') = \sum_{j \in G_i} x_{ij} + \sum_{j \in G_i} x'_{ij} - 2 \sum_{j \in G_i} x_{ij} x'_{ij} \qquad \text{(8-b)}.$$

Puisque les deux solutions x et x' vérifient les contraintes de choix (2) (i.e. $\sum_{j \in G_i} x_{ij} = \sum_{j \in G_i} x'_{ij} = 1$) alors l'équation (8-b) peut s'écrire

$$\delta_i(x,x') = 2(1 - \sum_{j \in G_i} x_{ij} x'_{ij}) \qquad \text{(8-c)}.$$

Sachant que la distance $\delta_i(x,x')$ est positive et que les vecteurs x et x' sont binaires, nous avons $\sum_{j \in G_i} x_{ij} x'_{ij} \in \{0,1\}$. Par conséquent, nous avons $\delta_i(x,x') \in \{0,2\}$. □

Une conséquence de la Proposition 7 est que si nous explorons les solutions voisines de x qui vérifient les contraintes de choix (2), alors la distance $\delta(x, x')$ a toujours une valeur paire (i.e., q est une valeur paire dans $V_q(x)$).

Nous nous sommes intéressés à une procédure de recherche locale dans le but d'améliorer les solutions fournies par un processus itératif décrit dans les sections suivantes. Cette recherche locale prend en entrée une solution non nécessairement réalisable. En fait, comme nous le verrons dans la suite, il est nécessaire que les contraintes de choix soient respectées,

mais pas forcément les contraintes de sac à dos. Nous avons besoin d'un processus rapide applicable plusieurs fois. Nous introduisons deux principes pour éviter de cycler dans une même zone de l'espace de recherche : (a) la recherche est toujours orientée vers une meilleure solution globale, et (b) nous éliminons définitivement de l'espace de recherche l'ensemble des solutions réalisables visitées à chaque itération.

Comme nous le verrons par la suite, l'heuristique proposée dans cette partie fournit une solution plus ou moins proche des solutions de l'enveloppe convexe d'une relaxation du problème MMKP. Nous cherchons donc une solution plus intéressante dans le voisinage de la solution courante à l'aide d'une recherche locale qui coupe l'espace de recherche et retourne uniquement une nouvelle meilleure borne inférieure (**BI**) pour le problème s'il en trouve une. Le principe de cette procédure est basé sur une permutation simultanée d'objets entre deux classes. Nous avons utilisé le voisinage $V_4(x)$. La distance entre la solution courante x et un de ces voisins est donc définie par $\delta(x,x')\in\{0,2,4\}$, car nous exécutons les mouvements éventuels sur deux classes au maximum sachant que chaque mouvement engendre au maximum une distance de 2. L'exécution de cette permutation doit être faite après avoir vérifié les deux conditions concernant la réalisabilité de la solution potentielle et l'amélioration de la meilleure BI. Nous ne validons pas le mouvement si ces deux conditions ne sont pas garanties.

Dans la Figure 2.2, nous illustrons l'échange de deux objets dans deux classes *i* et *i'* d'une solution.

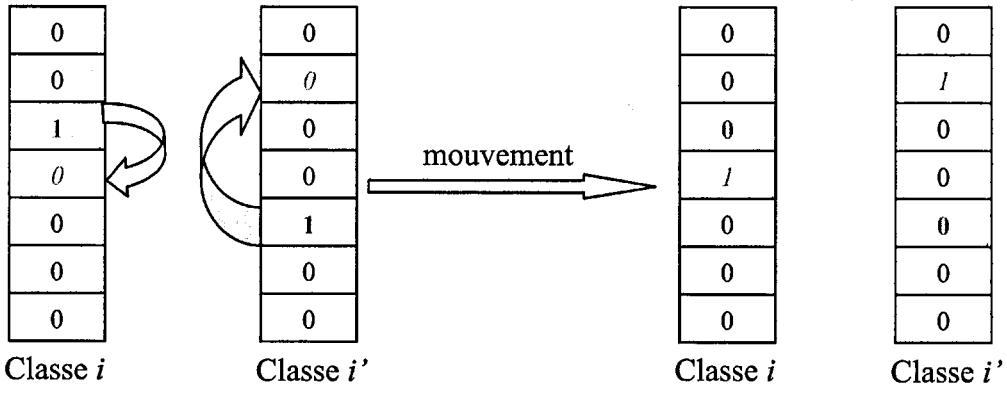


Figure 2.2 : Illustration d'un mouvement dans le voisinage V_4

Chaque tableau représente le vecteur de la solution de la classe correspondante. Les variables sont triées dans chaque classe dans l'ordre décroissant de leur coût dans la fonction objectif. Nous parcourons toutes les paires des objets entre les deux classes *i* et *i'* et nous vérifions si les deux conditions précédentes sont satisfaites. Notons que d'un point de vue

pratique, nous considérons d'abord le test sur l'amélioration de la meilleure borne inférieure car il est moins coûteux et permet d'élaguer rapidement des candidats. Si le résultat est positif nous vérifions la réalisabilité de la solution.

L'Algorithme 2.2 décrit l'évaluation du voisinage V_4 d'une solution courante, notée x. Nous supposons que $j(i)$ (resp. $j(i')$) est l'objet choisi dans la classe i (resp. i') de la solution x, et que j^* et j'^* sont les objets éventuellement choisis respectivement dans la classe i^* et dans la classe i'^*, dans le but de permuter j^* avec $j(i^*)$ et j'^* avec $j(i'^*)$ simultanément sachant que i'^* et i^* sont les deux classes associées à la meilleure permutation. L'évaluation est basée sur le changement de la borne inférieure qui peut être exprimée par $\Delta(j,j') = c_{ij} - c_{ij(i)} + c_{i'j'} - c_{i'j(i')}$.

Algorithme 2.2 : Meilleur voisin dans V_4

Procédure ***Meilleur_Voisin*** (*Solution x*)
début
  $\Delta_{max} = 0$;
  pour $i = 1 \ldots n$ **faire**
    pour $i' = i \ldots n$ **faire**
      $(j^+, j'^+) = \arg\max\{\Delta(j,j') : Ax + A_{ij} - A_{ij(i)} + A_{i'j'} - A_{i'j(i')} \le b, j \in G_i, j' \in G_{i'}\}$
      si $((j^+, j'^+) \neq (j(i), j(i'))$ et $\Delta(j^+, j'^+) > \Delta_{max})$ **alors**
        $i^* = i ; j^* = j^+ ;\ i'^* = i' ; j'^* = j'^+ ; \Delta_{max} = \Delta$;
      fin si
    fin pour
  fin pour
  $j(i^*) = j^* ; j(i'^*) = j'^*$;
  retourner x ;
fin

En termes de complexité, nous pouvons calculer le nombre maximum d'itérations qui peuvent être exécutées pour une recherche locale entre deux classes. Nous supposons que le nombre d'objets dans les classes est identique et est donné par l. Si nous considérons que tous les cas peuvent améliorer la borne inférieure, alors la complexité dans le pire des cas est $l^2 \times m$. Pour l'évaluation du voisinage complet de la solution courante, il est nécessaire d'effectuer la recherche entre chaque couple de classes. La complexité de la procédure de recherche locale dans le pire des cas est donc égale à $n(n-1)l^2m/2$. Même si cette complexité est importante, nous pouvons noter qu'en pratique le temps d'exécution reste raisonnable car de nombreux cas sont élagués par le test sur la borne inférieure. Notons aussi que dans l'Algorithme 2.2 la boucle sur i' commence par la valeur de i de manière à également évaluer le voisinage correspondant à une seule classe.

Une fois la recherche terminée dans le voisinage de la solution x, nous interdisons la recherche dans l'espace $V_4(x)$, en imposant $\delta(x', x) > 4$. En effet, pour que la solution x' ne soit pas voisine de x il faut que la distance entre ces deux solutions soit plus grande que la distance définie par le voisinage. En développant l'inégalité $\delta(x', x) > 4$, nous obtenons la pseudo-coupe suivante :

$$\sum_{j \in J^1(x)} x_j - \sum_{j \in J^0(x)} x_j \leq n - 3 \qquad (coupe\text{-}2).$$

Cette coupe devient valide une fois la procédure de recherche locale terminée.

Dans la partie suivante, nous décrivons nos heuristiques itératives basées sur des relaxations du problème MMKP.

2.5 Heuristiques Itératives basées sur les Relaxations pour le MMKP

La motivation principale de notre travail se base sur l'encadrement de la valeur optimale par des bornes supérieures et inférieures. Pour cela, nous appliquons une méthode itérative qui vise à améliorer ces bornes à chaque itération. Le principe général de cette heuristique (Wilbaut et Hanafi (2009)) est basé sur la résolution à chaque itération d'une relaxation du problème. Elle se compose des étapes suivantes :

- Étape 1 : Résoudre une ou plusieurs relaxations du problème courant et garder la ou les solutions optimales. Mettre à jour la meilleure borne supérieure du problème (si nécessaire).
- Étape 2 : Résoudre un ou plusieurs problèmes réduits induits par la ou les solutions précédentes, et mettre à jour la meilleure borne inférieure du problème (si nécessaire).
- Étape 3 : Si une condition d'arrêt est satisfaite alors renvoyer les deux bornes obtenues pour le problème. Sinon ajouter une ou plusieurs pseudo-coupes induites par la ou les solutions de la ou les relaxations et retourner à l'étape 1.

L'idée d'utiliser des relaxations dans des heuristiques pour résoudre des problèmes difficiles n'est pas nouvelle dans la littérature (Soyster, Lev et Slivka (1978)), notamment pour trouver des bornes supérieures. Dans cette partie de la thèse, nous traitons trois types de relaxation décrites dans la Section 2.3 : la relaxation en continu (LP), la relaxation MIP dans laquelle la relaxation en continu est considérée sur un sous-ensemble de variables, et la relaxation semi-continue des variables binaires dans laquelle nous relaxons les contraintes d'intégrité, mais en obligeant les variables à être proches de 1 ou de 0.

Les heuristiques itératives basées sur les relaxations reposent sur la proposition suivante qui concerne l'ajout de pseudo-coupes dans le problème courant de manière à élaguer une partie de l'espace de recherche.

Proposition 8 :

Soit P un problème 0-1MIP. Soient $\bar{x}$ une solution optimale de la relaxation en continu LP(P) et y une solution optimale du problème réduit $P(\bar{x}, J(\bar{x}))$. Une solution optimale de P est soit la solution y, soit une solution optimale du problème :

$$\left(P \;\middle|\; \left\{ \sum_{j \in J^1(\bar{x})} x_j - \sum_{j \in J^0(\bar{x})} x_j \leq \left| J^1(\bar{x}) \right| - 1 \right\} \right) \qquad (\textit{coupe-3}).$$

La preuve de cette proposition peut être trouvée dans Hanafi et Wilbaut (2009) ou Wilbaut et Hanafi (2009). Notons qu'elle reste valide en remplaçant la relaxation en continu par la relaxation MIP ou la RSC.

Les différentes variantes des heuristiques proposées dans cette partie pour résoudre le MMKP sont basées sur le schéma précédent. Nous commençons par la résolution d'une relaxation qui génère un problème réduit plus facile à résoudre. Nous renforçons la performance de ces heuristiques en appliquant notre procédure de recherche locale (décrite dans la Section 2.4) pour améliorer la solution trouvée à chaque itération et rétablir sa réalisabilité si nécessaire et si possible. Cette procédure exploite la structure du MMKP et a pour objectif de générer des coupes plus fortes à chaque itération. Nous appliquons également les techniques de fixation de variables décrites dans la Section 2.2 et de génération de coupes en exploitant les bornes courantes.

2.5.1 Heuristique basée sur la Programmation Linéaire

L'algorithme décrit dans cette partie utilise la relaxation en continu dans l'étape 1 du schéma général précédent. L'idée a été proposée initialement par Soyster, Lev et Slivka (1978) à la fin des années 1970 pour résoudre exactement les problèmes en variables 0-1. Nous appelons cette heuristique **HIPL** pour Heuristique Itérative basée sur la Programmation Linéaire. Elle élimine de l'espace de recherche les solutions optimales des relaxations produites précédemment en ajoutant des pseudo-coupes à chaque itération générées selon la Proposition 8. L'Algorithme 2.3 décrit l'heuristique HIPL.

Algorithme 2.3 : Heuristique itérative basée sur la programmation linéaire (HIPL)

stop = faux ;
$\bar{v} = \infty$; *// la meilleure borne supérieure*
$\underline{v} = 0$; *// la meilleure borne inférieure*
Q = MMKP ;

tant que Non stop **faire**
Soit x^{LP} une solution optimale de LP(Q) : $\bar{v} = v(\text{LP}(Q))$;
Ajouter à Q la pseudo-coupe (***coupe*-3**) : $Q = Q \left| \sum_{(i,j)\in J^1(x^{LP})} x_{ij} - \sum_{(i,j)\in J^0(x^{LP})} x_{ij} \leq \left| J^1(x^{LP}) \right| - 1 \right.$;
x' = ***Projection***(x^{LP}) ; *// x' satisfait les contraintes de choix*

// Recherche Locale
répéter
$x'' =$ ***Meilleur_Voisin*** (x') ;
Ajouter à Q la pseudo-coupe (***coupe*-2**) : $Q = Q \left| \sum_{(i,j)\in J^1(x')} x_{ij} - \sum_{(i,j)\in J^0(x')} x_{ij} \leq n-3 \right.$;
si $cx'' > \underline{v}$ **alors** $x' = x''$; $\underline{v} = cx''$; **fin si**
jusqu'à $cx'' < \underline{v}$;
Soit x' une solution optimale du problème réduit $Q(x^{LP}, J(x^{LP}))$;

// Recherche Locale
répéter
$x'' =$ ***Meilleur_Voisin*** (x') ;
Ajouter à Q la pseudo-coupe (***coupe*-2**) : $Q = Q \left| \sum_{(i,j)\in J^1(x')} x_{ij} - \sum_{(i,j)\in J^0(x')} x_{ij} \leq n-3 \right.$;
si $cx'' > \underline{v}$ **alors** $x' = x''$; $\underline{v}$ cx'' ; **fin si**
jusqu'à $cx'' < \underline{v}$;
Test de fixation des variables basées sur les coûts réduits et la borne inférieure ;
Ajouter à Q la coupe des coûts réduits (***coupe*-1**) :

$$Q = Q \left| \sum_{j\in J^0(x^{LP})} |\hat{c}_j| x_j + \sum_{j\in J^1(x^{LP})} |\hat{c}_j| (1 - x_j) \leq v(\text{LP}(Q)) - \underline{v} \right. ;$$

si $v(\text{LP}(Q)) - \underline{v} < 1$ **alors stop** = vrai ; **fin si**

fin tant que

L'heuristique HIPL fournit un encadrement de la valeur optimale du problème MMKP. La borne supérieure, fournie par la valeur de LP(Q), est améliorée à chaque itération par l'ajout des pseudo-coupes (*coupe*-1), (*coupe*-2) et (*coupe*-3) au problème courant Q. La borne inférieure correspond à la meilleure solution réalisable trouvée. Elle peut être améliorée dans un premier temps par une recherche locale en partant de la projection de la solution optimale x^{LP} de LP(Q). La procédure de projection répare la solution fractionnaire x^{LP} par rapport aux contraintes d'intégrité et aux contraintes de choix de la façon suivante : nous choisissons dans chaque classe l'objet correspondant à la variable qui a la valeur maximale dans x^{LP}. Dans un deuxième temps, nous résolvons le problème réduit $P(x^{LP}, J(x^{LP}))$, puis nous appliquons une

deuxième fois la procédure de recherche locale sur une solution optimale du problème réduit si elle existe. L'amélioration des bornes des problèmes nous permet d'envisager la fixation de variables selon les propriétés présentées dans la Section 2.2.2. Cette fixation peut permettre une accélération de la résolution des relaxations et des problèmes réduits. Le processus est répété jusqu'à ce qu'une condition d'arrêt soit satisfaite. En pratique, nous pouvons exécuter l'algorithme avec un temps maximum ou l'arrêter quand l'écart entre les bornes supérieure et inférieure est inférieur à une certaine tolérance.

La convergence de HIPL peut être assurée sous certaines conditions (voir Wilbaut et Hanafi (2009)). Cela correspond à la condition d'arrêt de l'Algorithme 2.3. Notons que cette convergence est généralement longue, ce qui justifie la mise en place d'un autre critère d'arrêt. Notons également que la première résolution du problème LP(MMKP) est faite en utilisant l'algorithme primal du simplexe, puis les problèmes LP(Q) sont résolus en utilisant la méthode duale du simplexe pour accélérer la résolution. Nous utilisons le logiciel CPLEX pour résoudre les relaxations et les problèmes réduits. Nous limitons le temps d'exécution de CPLEX pour la résolution des problèmes réduits, ce qui n'est pas gênant si HIPL n'est pas utilisé comme un algorithme exact.

2.5.2 Heuristique basée sur la Relaxation en Nombres Entiers Mixtes

Nous considérons dans cette section la relaxation en nombres entiers mixtes MIP(MMKP). Plusieurs intégrations de cette relaxation dans le schéma général précédent sont possibles. Une première heuristique peut être obtenue à partir de HIPL en remplaçant simplement la relaxation en continu par la relaxation en nombres entiers mixtes. La question qui se pose à nous à ce niveau est de savoir comment définir les critères de choix du sous-ensemble J des variables à contraindre à être entières. Une façon simple est de choisir à chaque itération les variables fractionnaires à l'itération précédente. Cela permet aussi de diversifier la recherche. Cette version s'est montrée peu efficace en pratique, c'est pourquoi nous ne la considérons pas dans la suite. Nous proposons donc une autre variante appelée HIM dans laquelle la sélection de l'ensemble J se fait en exploitant l'information obtenue par la résolution de la relaxation en continu. Ainsi, J est l'ensemble des variables qui ont des valeurs fractionnaires dans une solution optimale de la relaxation LP(MMKP) à chaque itération, i.e., $J = J^*(x^{LP})$, avec x^{LP} une solution optimale de la relaxation en continu du problème courant. Dans la suite, nous notons MIP(Q) la relaxation MIP(Q, $J^*(x^{LP})$), et nous présentons le pseudo-code de HIM dans l'Algorithme 2.4.

Algorithme 2.4 : Heuristique itérative basée sur la relaxation MIP (HIM)

stop = faux ;
$\bar{v} = \infty$; *// la meilleure borne supérieure*
$\underline{v} = 0$; *// la meilleure borne inférieure*
Q = MMKP ;

tant que Non stop **faire**
Soit x^{LP} une solution optimale de LP(Q) ;
Soit x^{MIP} une solution optimale de MIP(Q) ;
$\bar{v}$ = min ($\bar{v}$, v(MIP(Q)), v(LP(Q)) ;
Ajouter à Q la coupe (***coupe*-3**) : $Q = Q \left| \sum_{(i,j) \in J^1(x^{LP})} x_{ij} - \sum_{(i,j) \in J^0(x^{LP})} x_{ij} \leq \left| J^1(x^{LP}) \right| - 1 \right.$;
x' = ***Projection***(x^{MIP}) ; *// x' satisfait les contraintes de choix*

// Recherche Locale
répéter
x'' = ***Meilleur_Voisin*** (x') ;
Ajouter à Q la pseudo-coupe (***coupe*-2**) : $Q = Q \left| \sum_{(i,j) \in J^1(x')} x_{ij} - \sum_{(i,j) \in J^0(x')} x_{ij} \leq n - 3 \right.$;
si $cx'' > \underline{v}$ **alors** $x' = x''$; $\underline{v} = cx''$; **fin si**
jusqu'à $cx'' < \underline{v}$;
Soit x' une solution optimale du problème réduit $Q\left(x^{MIP}, J\left(x^{MIP}\right)\right)$;

// Recherche Locale
répéter
x'' = ***Meilleur_Voisin*** (x') ;
Ajouter à Q la pseudo-coupe (***coupe*-2**) : $Q = Q \left| \sum_{(i,j) \in J^1(x')} x_{ij} - \sum_{(i,j) \in J^0(x')} x_{ij} \leq n - 3 \right.$;
si $cx'' > \underline{v}$ **alors** $x' = x''$; $\underline{v} = cx''$; **fin si**
jusqu'à $cx'' < \underline{v}$;

Test de fixation des variables dans Q et MIP(Q) ;
Ajouter à Q et à MIP(Q) la coupe des coûts réduits associée à MIP(Q) (***coupe*-1**);
si $\bar{v} - \underline{v} < 1$ **alors stop** = vrai ; **fin si** ;
fin tant que

L'application de cet algorithme nécessite la résolution de la relaxation en continu pour définir l'ensemble J, et ainsi définir la relaxation MIP à chaque étape. Nous appliquons la procédure de recherche locale sur la solution trouvée de MIP(Q) et sur la solution du problème réduit afin d'ajouter des coupes sur les deux problèmes Q et MIP(Q). Nous remarquons que nous pouvons également résoudre le problème réduit associé à la relaxation LP(MMKP) à chaque itération pour cumuler les informations obtenues de chaque relaxation. Notons aussi que la fixation est appliquée seulement sur l'ensemble des variables fractionnaires à chaque étape et qu'en pratique, après un certain nombre d'itérations, le

nombre de contraintes du problème Q devient très important, ce qui ralentit la résolution des relaxations mixtes.

2.5.3 Heuristique basée sur la Relaxation Semi-Continue

La relaxation semi-continue, présentée dans la Section 2.3.3, peut aussi être utilisée pour résoudre le problème MMKP. A chaque itération nous résolvons le problème RSC(α), mais sa résolution peut engendrer une solution x où toutes les valeurs des variables dans une classe i sont dans l'intervalle [0, α]. Cela complique alors le choix de l'objet dans la classe i (pour pouvoir appliquer la procédure de recherche locale). Ce cas apparaît quand toutes les valeurs de x_{ij} dans une solution optimale de la relaxation prennent la valeur α, avec $\sum_{j \in G_i} \alpha = 1$. Pour éviter ce cas, il faut borner la valeur de α de façon à avoir $\sum_{j \in G_i} \alpha \leq 1 - \alpha$, c'est-à-dire $l\alpha \leq 1 - \alpha$, ce qui entraîne $\alpha \in [0, 1/l+1]$. De cette façon, il existe forcément une variable dans chaque classe qui prend une valeur dans l'intervalle [1- α, 1]. Le principe de cette variante est décrit dans l'Algorithme 2.5.

Dans cet algorithme, nous résolvons la relaxation RSC(α) pour obtenir la solution (x, y). Nous savons que le vecteur des variables binaires y respecte les contraintes de choix du MMKP et représente aussi la projection de la solution x. Nous appliquons notre recherche locale directement sur ce vecteur. Celle-ci nous permet d'ajouter la pseudo-coupe (*coupe*-2) au problème Q. (*coupe*-2) peut être ajoutée sur les variables continues x de la relaxation RSC(α) ou sur les variables binaires y. Dans notre cas, nous préférons les ajouter sur les variables binaires y pour mieux couper l'espace de recherche. Après avoir appliqué la recherche locale nous résolvons le problème réduit associé à la solution x avant d'appliquer une nouvelle fois la recherche locale sur la solution trouvée. Nous terminons l'itération avec le test de fixation dans lequel nous considérons seulement les variables continues du problème (x).

La résolution du problème RSC(α) donne une information importante sur la borne supérieure, ce qui doit permettre une convergence plus rapide de l'algorithme (à condition d'obtenir les mêmes bornes inférieures). Le temps d'exécution de cet algorithme est évidemment plus long que le temps d'exécution des deux versions précédentes. Cependant, il reste faible car en ajoutant les variables y nous limitons l'espace de relaxation des variables x

et nous ne les forçons pas à être binaires. L'effet de la contrainte $\sum_{(i,j)\in J^1(y')} y_{ij} - \sum_{(i,j)\in J^0(y')} y_{ij} \leq n-3$ (resp. $\sum_{(i,j)\in J^1(x')} y_{ij} - \sum_{(i,j)\in J^0(x')} y_{ij} \leq n-3$) est important sur RSC($\alpha$) car la coupe associée porte sur des variables binaires et non sur des variables continues. Notons que cette heuristique est valide pour les problèmes mixtes avec variables binaires.

Algorithme 2.5 : Heuristique itérative basée sur la relaxation semi-continue (RSC)

stop = faux ;
$\bar{v} = \infty$; *// la meilleure borne supérieure*
$\underline{v} = 0$; *// la meilleure borne inférieure*
Q = RSC(α) ;

tant que Non stop **faire**
  Soit (x, y) une solution optimale de Q : $\bar{v} = v(Q)$;
  $y' = y$; *// y satisfait les contraintes de choix*

  // Recherche Locale
  répéter
    $y'' =$ ***Meilleur_Voisin*** (y') ;
    Ajouter à Q la pseudo-coupe (***coupe-2***) : $Q = Q \mid \sum_{(i,j)\in J^1(y')} y_{ij} - \sum_{(i,j)\in J^0(y')} y_{ij} \leq n-3$;
    si $cy'' > \underline{v}$ **et** y'' est réalisable **alors** $y' = y''$; $\underline{v} = cy''$ **fin si**
  jusqu'à $cy'' < \underline{v}$;

  Soit x' une solution optimale du problème réduit $Q((x,y),J(x,y))$;

  // Recherche Locale
  répéter
    $x'' =$ ***Meilleur_Voisin*** (x') ;
    Ajouter à Q la pseudo-coupe (***coupe-2***) : $Q = Q \mid \sum_{(i,j)\in J^1(x')} y_{ij} - \sum_{(i,j)\in J^0(x')} y_{ij} \leq n-3$;
    si $cx'' > \underline{v}$ **alors** $x' = x''$; $\underline{v} = cx''$ **fin si**
  jusqu'à $cx'' < \underline{v}$;

  Test de fixation des variables dans Q ;
  Ajouter à Q la coupe des coûts réduits (***coupe-1***) ;

  si $v(Q) - \underline{v} < 1$ **alors** stop := vrai ; **fin si**
fin tant que

Les trois heuristiques proposées dans cette partie sont validées dans la section suivante sur un ensemble d'instances publiées dans la littérature.

2.6 Résultats numériques

Nous présentons dans cette section les résultats obtenus en appliquant les trois heuristiques décrites dans la section précédente : HIPL, HIM et RSC. Pour valider nos propositions, nous comparons nos résultats avec ceux obtenus par la meilleure version de l'algorithme proposé par Cherfi et Hifi (2008). Nos algorithmes ont été codés en C++ et ont été exécutés sur un Dell 2.4GHz avec 4 giga-octets de RAM. Pour évaluer nos algorithmes, nous utilisons les instances mentionnées dans Cherfi et Hifi (2008) ou encore dans Hifi, Michrafy et Sbihi (2004) et disponibles sur Internet[1]. Les caractéristiques de ces instances sont récapitulées dans le Tableau 2.1. Nous avons examiné un total de 33 instances issues de deux groupes.

Instance	m	n	n_i	n'	Instance	m	n	n_i	n'
I01	5	5	5	25	Ins01	10	50	10	500
I02	5	5	10	50	Ins02	10	50	10	500
I03	10	10	15	150	Ins03	10	60	10	600
I04	10	10	20	200	Ins04	10	70	10	700
I05	10	10	25	250	Ins05	10	75	10	750
I06	10	10	30	300	Ins06	10	75	10	750
I07	10	10	100	1000	Ins07	10	80	10	800
I08	10	10	150	1500	Ins08	10	80	10	800
I09	10	10	200	2000	Ins09	10	80	10	800
I10	10	10	250	2500	Ins10	10	90	10	900
I11	10	10	300	3000	Ins11	10	90	10	900
I12	10	10	350	3500	Ins12	10	100	10	1000
I13	10	10	400	4000	Ins13	10	100	30	3000
					Ins14	10	150	30	4500
					Ins15	10	180	30	5400
					Ins16	10	200	30	6000
					Ins17	10	250	30	7500
					Ins18	10	280	20	6500
					Ins19	10	300	20	6000
					Ins20	10	350	20	7000

Tableau 2.1 : Instances pour le MMKP

Nous reportons dans le Tableau 2.1 le nombre de contraintes de sac à dos dans la colonne m, le nombre de classes n (resp. le nombre des objets dans chaque classe n_i) et le nombre total

[1] http://www.laria.u-picardie.fr/hifi/OR-Benchmark/

de variables du problème *n'*. Les instances de gauche, notées I01 ... I13 ont été générées et utilisées par Khan, Li, Manning et Akbar (2002). Elles regroupent six petites instances et sept instances plus difficiles. Les instances de droite, notées Ins01 ... Ins20, ont été générées aléatoirement par Hifi, Michrafy et Sbihi (2006) selon la procédure décrite par Khan (1998) et représentent un ensemble d'instances difficiles de moyenne et grande taille.

Nous présentons dans le Tableau 2.2 les meilleures valeurs reportées dans la littérature pour les 33 instances et celles trouvées par le logiciel CPLEX avec une limite d'une heure de temps de calcul. Nous considérons que cette limite est raisonnable pour une comparaison avec nos processus heuristiques pour résoudre le MMKP.

	Best		*Cplex*			*Best*		*Cplex*	
Instance	*v*	*Cpu*	*V*	*Cpu*	Instance	*V*	*Cpu*	*V*	*Cpu*
I01	**173**	< 1	**173 ***	< 1	Ins01	10732	15.65	**10738 ***	2592
I02	**364**	< 1	**364 ***	< 1	Ins02	**13598**	12.78	**13598 ***	626
I03	**1602**	3.48	**1602***	2	Ins03	**10943**	11.60	10939	3600
I04	**3597**	6.42	**3597***	12	Ins04	14440	62.53	**14442**	3600
I05	**3905,7**	< 1	**3905.7***	< 1	Ins05	**17053**	55.92	**17053**	3600
I06	**4799,3**	< 1	**4799.3***	< 1	Ins06	16825	35.91	**16826**	3600
I07	24587	34.09	**24589**	3600	Ins07	16435	122.40	**16440**	3600
I08	36892	170.06	**36896**	3600	Ins08	**17510**	36.33	17502	3600
I09	**49176**	72.18	49163	3600	Ins09	**17760**	29.03	17751	3600
I10	61461	98.61	**61462**	3600	Ins10	**19314**	65.84	19304	3600
I11	73775	103.92	**73776**	3600	Ins11	**19434**	46.25	19433	3600
I12	86078	211.63	**86087**	3600	Ins12	**21731**	126.58	21720	3600
I13	**98431**	334.72	**98431**	3600	Ins13	**21575**	184.95	21573	3600
					Ins14	**32870**	71.16	32869	3600
					Ins15	39157	135.62	**39160**	3600
					Ins16	43361	162.39	**43363**	3600
					Ins17	54349	212.15	**54360**	3600
					Ins18	60460	227.33	**60465**	3600
					Ins19	64923	450.58	**64928**	3600
					Ins20	75611	348.18	**75617**	3600

Tableau 2.2 : Valeurs références pour l'évaluation de nos approches

Nous donnons dans le Tableau 2.2 la meilleure valeur connue pour chaque instance reportée dans Cherfi et Hifi (2008) (resp. obtenue par CPLEX) dans la colonne *Best* (resp. *Cplex*) avec d'un côté la valeur de la borne inférieure dans la colonne *v* et de l'autre côté le temps d'exécution en secondes associé (*Cpu*). Notons que nous reportons les meilleurs résultats pour l'algorithme de Cherfi et Hifi (2008) parmi trois variantes de la méthode, et que

ces résultats ont été obtenus sur un UltraSparc10 (250 MHz et 1Gb de RAM). Dans les Tableaux 2.2 à 2.5, les valeurs écrites en gras signifient que l'algorithme correspondant a fourni la meilleure valeur connue, et * signifie que l'algorithme a prouvé l'optimalité de la solution correspondante.

Le Tableau 2.2 montre la difficulté des instances considérées, en particulier en analysant les résultats obtenus par CPLEX. Après une heure d'exécution, celui-ci n'est pas capable de trouver les solutions optimales pour la plupart des instances (et/ou d'en prouver l'optimalité). De plus, pour une grande partie des instances, CPLEX ne visite pas les meilleures solutions trouvées par l'heuristique de Cherfi et Hifi (2008). CPLEX arrive quand même à obtenir de nouvelles meilleures bornes inférieures et prouver l'optimalité de ses solutions pour 8 instances (I01, I02, I03, I04, I05, I06, Ins01 et Inst02), mais l'amélioration est atténuée par le temps d'exécution nécessaire pour les obtenir. Comme nos méthodes sont des heuristiques et que nous limitons le temps d'exécution, nous nous comparons dans la suite en particulier aux résultats reportés par Cherfi et Hifi (2008).

Nous présentons dans les Tableaux 2.3, 2.4 et 2.5 les résultats obtenus par nos algorithmes pour les 33 instances. Notre objectif est de montrer que les méthodes proposées visitent rapidement de bonnes solutions. Nous allons également vérifier le comportement des algorithmes lorsque nous augmentons le temps d'exécution. Rappelons que les algorithmes proposés convergent en théorie vers une solution optimale, mais qu'ici nous les utilisons uniquement comme heuristiques.

Nous fournissons dans les Tableaux 2.3, 2.4 et 2.5 une trace de l'exécution de HIPL, HIM et RSC respectivement pour chaque instance. Nous reportons les résultats obtenus et le temps d'exécution associé. Pour chaque heuristique, nous donnons les valeurs des bornes inférieures trouvées dans la colonne *v*, le temps d'exécution associé à ces valeurs (en secondes) dans la colonne *Cpu*, le pourcentage de variables fixées dans la colonne *Fix*, et l'écart entre la borne supérieure courante et la borne inférieure courante calculé par (borne supérieure - borne inférieure) / borne supérieure et présenté en pourcentage dans la colonne *Gap*. Nous donnons ces informations pour la première et la dernière valeur trouvée (si nous présentons une seule valeur alors il n'y a pas d'amélioration). Nous reportons aussi les résultats intermédiaires lorsque l'algorithme obtient une amélioration sensible en termes de borne inférieure ou de fixation. Notons que le temps d'exécution total est limité à 10 minutes, de manière à rester dans un cadre heuristique.

Instance	Best	*HIPL*				Instance	Best	*HIPL*			
		v	*Cpu*	*Fix* %	*Gap* %			*v*	*Cpu*	*Fix* %	*Gap* %
I01	**173**	**173***	< 1	36	0	Ins06	16825	16791	1	0	0.39
I02	**364**	**364***	< 1	51	0.43			**16826**	598	0	0.17
I03	**1602**	**1602**	3	10	0.12	Ins07	16435	16382	5	0	0.47
I04	**3597**	**3597**	7	2	0.35			**16440**	289	26	0.11
I05	**3905.7**	**3905.7***	< 1	89	0.01	Ins08	**17510**	17461	6	0	0.41
I06	**4799.3**	**4799.3***	< 1	86	0.04			17503	117	0	0.17
I07	**24587**	24535	1	0	0.30	Ins09	**17760**	17713	1	0	0.38
		24585	60	18	0.09			17751	614	2	0.16
I08	**36892**	36843	1	0	0.17	Ins10	**19314**	19279	5	0	0.30
		36885	394	26	0.05			19306	464	0	0.16
I09	**49176**	49128	1	0	0.13	Ins11	19434	19388	6	0	0.38
		49172	249	19	0.04			**19446**	560	34	0.08
I10	**61461**	61419	1	0	0.11	Ins12	**21731**	21690	2	0	0.31
		61460	188	6	0.04			21727	208	0	0.13
I11	73775	73753	5	0	0.06	Ins13	**21575**	21569	1	22	0.11
		73778	422	24				21574	198	41	0.09
		73784	494	44	0.02	Ins14	**32870**	32862	1	25	0.08
I12	86078	86051	1	0	0.06			32869	106	47	0.06
		86091	542	57	0.01	Ins15	39157	39149	1	22	0.07
I13	**98431**	98394	1	0	0.06			**39160**	311	62	0.04
		98425	11	13	0.02	Ins16	43361	43351	2	14	0.07
Ins01	**10732**	10700	1	0	0.50			**43363**	139	53	0.04
		10732	438	18	0.18	Ins17	54349	**54350**	2	34	0.04
Ins02	**13598**	13561	1	0	0.53			54353	570	48	0.03
		13598	343	26	0.21	Ins18	60460	60447	2	7	0.05
Ins03	**10943**	10877	1	0	0.98			**60462**	293	48	0.03
		10940	307	0	0.38	Ins19	64923	64911	1	6	0.05
Ins04	14440	14381	1	0	0.67			**64924**	37	38	0.03
		14442	106	2	0.24	Ins20	**75611**	75594	3	5	0.04
Ins05	**17053**	16998	2	0	0.47			75606	229	34	0.03
		17044	648	0	0.19						

Tableau 2.3 : Résultats numériques de HIPL

Notre algorithme basé sur la relaxation en continu obtient des résultats assez encourageants. Si nous nous comparons uniquement à l'algorithme de Cherfi et Hifi (2008), le Tableau 2.3 montre que HIPL améliore 11 meilleures bornes inférieures et visite 8 meilleures solutions. Notons que les nouvelles meilleures bornes inférieures sont souvent obtenues bien avant les 10 minutes autorisées. Nous remarquons aussi que l'algorithme HIPL a pu rapidement prouver l'optimalité de la solution sur les 4 petites instances I1, I2, I5 et I6. D'une manière générale, HIPL est plus efficace pour les instances du second groupe, qui sont les plus grandes instances. La résolution rapide de la relaxation en continu utilisée dans HIPL permet d'exécuter un nombre important d'itérations. Cela augmente le nombre de zones prometteuses visitées par l'algorithme, ce qui justifie la qualité des solutions trouvées par cette heuristique. Par rapport aux résultats obtenus pour la fixation, qui dépend directement du saut entre les bornes, nous remarquons que le pourcentage de fixation augmente logiquement

lorsque le saut diminue, sachant que ce dernier diminue lorsque les bornes sont améliorées. Cette analyse reste valable pour les deux autres heuristiques HIM et RSC.

Les résultats obtenus par l'algorithme HIM sont donnés dans le Tableau 2.4.

Instance	Best	*HIM*				Instance	Best	*HIM*			
		v	*Cpu*	*Fix* %	*Gap* %			*v*	*Cpu*	*Fix* %	*Gap* %
I01	**173**	**173***	< 1	4	0	Ins06	**16825**	16797	1	0	0.34
I02	**364**	**364***	< 1	55	0.23			16823	197	2	0.18
I03	**1602**	**1602**	17	1	0	Ins07	**16435**	16390	1	0	0.42
I04	**3597**	**3597**	103	15	0			16421	508	0	0.22
I05	**3905.7**	**3905.7***	< 1	89	0	Ins08	**17510**	17454	1	0	0.45
I06	**4799.3**	**4799.3***	< 1	86	0			17503	222	0	0.16
I07	**24587**	24561	1	0	0.19	Ins09	**17760**	17721	1	0	0.33
		24587	414	25	0.08			17754	160	9	0.14
I08	**36892**	36850	1	0	0.15	Ins10	**19314**	19279	6	0	0.29
		36883	327	16	0.05			19299	96	0	0.19
I09	**49176**	49134	1	0	0.12	Ins11	**19434**	19383	1	0	0.40
		49172	198	19	0.04			19430	426	1	0.16
I10	**61461**	61436	4	0	0.08	Ins12	21731	21665	1	0	0.42
		61461	578	6	0.04			**21738**	171	1	0.08
I11	**73775**	73753	9	0	0.06	Ins13	**21575**	21562	1	7	0.14
		73770	370	10	0.04			21574	391	41	0.08
I12	86078	86036	2	0	0.07	Ins14	**32870**	32862	2	25	0.08
		86091	514	59	0.01			32869	281	49	0.06
I13	98431	98394	1	0	0.06	Ins15	**39157**	39149	2	22	0.07
		98425	20	13	0.02			**39157**	208	52	0.04
Ins01	**10732**	10712	13	0	0.37	Ins16	43361	43349	3	10	0.07
		10718	432	3	0.29			**43362**	438	51	0.04
Ins02	**13598**	13587	1	15	0.31	Ins17	54349	**54350**	3	34	0.04
		13597	83	30	0.06			54356	494	57	0.03
Ins03	**10943**	10904	1	0	0.72	Ins18	60460	60451	4	14	0.05
		10943	341	3	0.34			**60461**	269	45	0.03
Ins04	14440	14410	1	0	0.46	Ins19	**64923**	64916	2	14	0.04
		14442	42	3	0.23			**64923**	74	33	0.03
Ins05	17053	17000	1	0	0.45	Ins20	**75611**	75596	4	7	0.04
		17055	515	8	0.12			75610	255	47	0.02

Tableau 2.4 : Résultats numériques de HIM

L'algorithme basé sur la relaxation MIP obtient 7 nouvelles meilleures bornes et visite 11 meilleures solutions. Rappelons que dans la relaxation MIP nous obligeons les variables fractionnaires de la relaxation en continu à être binaires, ce qui nous ramène à un autre espace de recherche que celui de la relaxation en continu. Ceci justifie le fait de trouver des résultats différents de ceux de HIPL, parfois meilleurs (par exemple pour Ins03 : 10943 pour HIM et 10940 pour HIPL), et parfois moins bons (par exemple pour Ins11 : 19430 pour HIM et 19446 pour HIPL). Ces résultats ne nous permettent pas de tirer de conclusion sur la qualité des zones de recherche visitées par ces algorithmes puisque les résultats varient en fonction des instances. Comme pour l'algorithme HIPL, HIM a pu prouver rapidement l'optimalité de la solution trouvée sur les 4 petites instances I1, I2, I5 et I6.

Le Tableau 2.5 donne les résultats obtenus par la dernière version de notre algorithme, RSC.

Instance	Best	RSC				Instance	Best	RSC			
		v	*Cpu*	*Fix %*	*Gap %*			*v*	*Cpu*	*Fix %*	*Gap %*
I01	**173**	**173***	< 1	48	0	Ins06	**16825**	16808	21	0	0.28
I02	**364**	**364***	< 1	64	0			16820	305	0	0.20
I03	**1602**	**1602**	168	26	0	Ins07	**16435**	16380	21	0	0.48
I04	**3597**	3592	1	14	0			16420	293	0	0.23
I05	**3905.7**	**3905.7***	< 1	89	0	Ins08	**17510**	17485	20	0	0.27
I06	**4799.3**	**4799.3***	< 1	86	0.02			17505	66	6	0.15
I07	**24587**	24584	1	15	0.09	Ins09	**17760**	17740	21	0	0.22
											0.21
I08	**36892**	36849	2	0	0.15	Ins10	**19314**	19270	21	0	0.34
		36867	64	0	0.10			19292	361	0	0.22
I09	**49176**	49143	20	0	0.10	Ins11	**19434**	19412	21	0	0.25
		49173	345	21	0.04			19421	256	0	0.20
I10	**61461**	61447	22	0	0.06	Ins12	**21731**	21703	21	0	0.24
		61455	233	1	0.05			21720	204	0	0.16
I11	**73775**	73760	22	0	0.05	Ins13	**21575**	21568	42	26	0.11
		73771	352	7	0.04			21570	214	62	0.09
I12	**86078**	86063	24	0	0.04	Ins14	**32870**	32865	38	36	0.06
		86076	71	12	0.03			32869	536	51	0.05
I13	**98431**	98397	22	0	0.05	Ins15	**39157**	39144	30	9	0.08
		98420	591	4	0.03			39150	79	9	0.06
Ins01	**10732**	10684	22	0	0.62	Ins16	**43361**	43353	36	21	0.06
		10717	266	2	0.31			43358	199	35	0.04
Ins02	**13598**	13583	12	16	0.33	Ins17	54349	54345	49	0	0.05
		13598	100	41	0.05			**54350**	329	0	0.04
Ins03	**10943**	10922	20	0	0.56	Ins18	**60460**	60448	40	7	0.05
		10933	247	0	0.43			**60460**	379	29	0.03
Ins04	14440	**14445**	9	4	0.22	Ins19	**64923**	64919	25	20	0.04
					0.20			**64923**	126	33	0.03
Ins05	**17053**	17026	20	0	0.30	Ins20	**75611**	75603	48	0	0.03
		17037	319	0	0.23			75608	242	0	0.02

Tableau 2.5 : Résultats numériques de RSC

L'algorithme basé sur la relaxation semi-continue obtient 2 nouvelles meilleures bornes inférieures pour les instances Ins04 et Ins17, et visite 8 meilleures solutions. Ces résultats un peu moins bons peuvent être justifiés en partie par le fait que la résolution de la relaxation semi-continue demande un temps d'exécution plus important, ce qui ne permet pas d'itérer suffisamment pour visiter un large espace de recherche. Par contre cette heuristique fournit des bornes supérieures plus fortes pour le MMKP. Comme pour les deux autres heuristiques HIPL et HIM, RSC a prouvé l'optimalité de la solution trouvée sur les 4 petites instances I1, I2, I5 et I6.

La fixation de variables dépend principalement de trois données : la borne supérieure, la borne inférieure, et l'instance elle-même. Par exemple pour l'instance Ins04, nous remarquons que l'impact des techniques de fixation est très faible devant la difficulté de l'instance. Dans

d'autres instances nous arrivons à fixer plus que la moitié des variables du problème comme pour l'instance Ins15 (avec HILP ou HIM). Par contre, nous remarquons que si les bornes inférieures des trois heuristiques sont identiques alors la fixation fonctionne en général mieux pour RSC que pour HIM, et mieux pour HIM que pour HIPL. Par exemple dans l'instance I02 et pour la même borne inférieure (364), l'heuristique RSC arrive à fixer 64% des variables, HIM 55%, et HIPL 51%. Ceci peut être justifié par la qualité des bornes supérieures fournies par chaque algorithme. Notons que le pourcentage de fixation pour l'heuristique RSC n'augmente pas toujours car nous ne trouvons pas toujours la solution optimale pour la relaxation (par limite de temps) et donc nous ne pouvons pas déclencher la procédure de fixation (voir par exemple l'instance Ins18).

Le Tableau 2.6 synthétise les résultats obtenus par nos trois heuristiques, uniquement en termes de qualité de la borne inférieure. Nous rappelons dans ce tableau pour chaque instance la valeur mentionnée dans Cherfi et Hifi (2008), et les trois valeurs obtenues par nos méthodes, HIPL, HIM et RSC.

Instance	Best	*HIPL*	*HIM*	RSC	Instance	Best	*HIPL*	*HIM*	RSC
I1	**173**	**173***	**173***	**173***	Ins05	17053	17044	**17055**	17037
I2	**364**	**364***	**364***	**364***	Ins06	16825	**16826**	16823	16820
I3	**1602**	**1602**	**1602**	**1602**	Ins07	16435	**16440**	16421	16420
I4	**3597**	**3597**	**3597**	3592	Ins08	**17510**	17503	17503	17505
I5	**3905.7**	**3905.7***	**3905.7***	**3905.7***	Ins09	**17760**	17751	17754	17740
I6	**4799.3**	**4799.3***	**4799.3***	**4799.3***	Ins10	**19314**	19306	19299	19292
I7	**24587**	24585	**24587**	24584	Ins11	19434	**19446**	19430	19421
I8	**36892**	36885	36883	36867	Ins12	21731	21727	**21738**	21720
I9	**49176**	49172	49172	49173	Ins13	**21575**	21574	21574	21570
I10	**61461**	61460	**61461**	61455	Ins14	**32870**	32869	32869	32869
I11	73775	**73784**	73770	73771	Ins15	39157	**39160**	**39157**	39150
I12	86078	**86091**	**86091**	86076	Ins16	43361	**43363**	**43362**	43358
I13	**98431**	98425	98425	98420	Ins17	54349	**54353**	**54356**	**54350**
Ins01	**10732**	**10732**	10718	10717	Ins18	60460	**60462**	**60461**	**60460**
Ins02	**13598**	**13598**	13597	**13598**	Ins19	64923	**64924**	**64923**	**64923**
Ins03	**10943**	10940	**10943**	10933	Ins20	**75611**	75606	75610	75608
Ins04	14440	**14442**	**14442**	**14445**					

Tableau 2.6 : Synthèse des résultats

Le Tableau 2.6 montre que nos algorithmes visitent au total 11 meilleures solutions et améliorent 13 meilleures bornes inférieures pour les 33 instances en comparaison aux résultats de Cherfi et Hifi (2008). Si nous comparons seulement nos résultats à ceux de CPLEX, nos algorithmes visitent 12 meilleures solutions et améliorent 12 bornes inférieures, ce qui prouve que les heuristiques proposées sont efficaces. Finalement, en prenant en considération les meilleures valeurs reportées dans le Tableau 2.2, nos heuristiques visitent 12 meilleures solutions et améliorent 6 bornes inférieures.

De manière générale, le temps d'exécution des algorithmes pour obtenir les meilleures bornes inférieures est un peu plus important que la méthode de Cherfi et Hifi (2008). La raison principale réside dans la résolution des problèmes réduits à chaque itération. De plus, l'ajout des contraintes à chaque itération rend ces problèmes réduits plus grands et donc plus difficiles à résoudre. Même si nos algorithmes ne peuvent pas être utilisés en pratique en tant que méthode exacte, ils peuvent produire des résultats très intéressants dans un temps d'exécution court, et donc un bon compromis entre qualité de la solution et temps d'exécution. Ceci peut être exploité dans le cas des problèmes à temps réels. Généralement, les résultats prouvent que nos méthodes sont plus efficaces pour les grandes instances.

Finalement, nous illustrons dans la Figure 2.3 l'évolution des bornes supérieures des trois heuristiques sur l'instance Ins02.

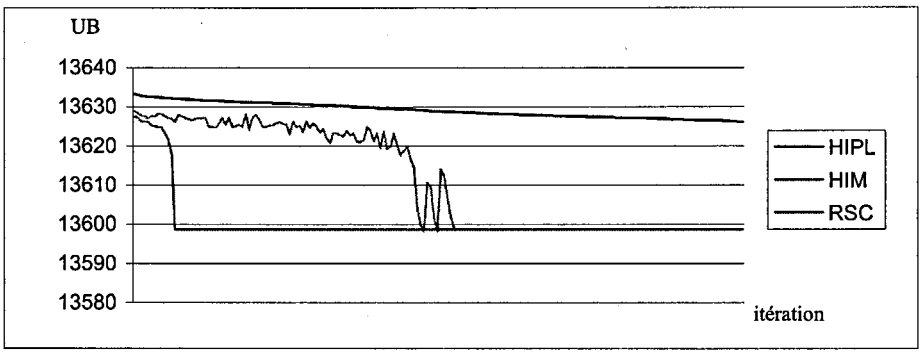


Figure 2.3 : Bornes supérieures pour Ins02

Cet exemple montre que les bornes supérieures fournies par RSC sont les meilleures (pour cette instance), et qu'elles convergent vite vers une bonne valeur (la valeur optimale était 13598). Les bornes supérieures de HIM viennent généralement en deuxième place. Nous remarquons aussi que les bornes fournies par cette heuristique ne se présentent pas sous la forme d'une fonction monotone. Cela peut être justifié par le fait qu'à chaque itération, et après avoir ajouté les pseudo-coupes, nous changeons l'ensemble des variables binaires pour la relaxation MIP(MMKP). Nous ne résolvons donc pas le même problème avec plus de contraintes, ce qui ne garantit pas la décroissance uniforme des bornes supérieures. Ce cas n'arrive pas pour les heuristiques RSC et HIPL, sauf si nous ne trouvons pas la valeur optimale pour une relaxation à une itération donnée.

2.7 Conclusions et perspectives

Dans cette partie, nous nous sommes intéressés au problème du sac à dos multidimensionnel à choix multiples. Après avoir présenté quelques applications et méthodes

existantes pour résoudre ce problème, nous avons décrit notre méthode de résolution basée sur des heuristiques itératives. Ces algorithmes, qui peuvent être considérés théoriquement en tant que méthodes exactes, résolvent une série de sous-problèmes de petite taille produits en exploitant l'information obtenue à partir des relaxations considérées du problème. Nous avons aussi renforcé ces algorithmes par une recherche locale. Les résultats montrent que les heuristiques convergent rapidement vers des solutions élites. En pratique, les méthodes fournissent 13 nouvelles meilleures bornes inférieures et visitent 11 meilleures solutions pour 33 instances difficiles de la littérature. Les expériences ont aussi montré que la version basée sur la relaxation en continu converge plus rapidement vers de bonnes solutions.

La relaxation semi-continue introduite dans cette partie fournit une meilleure borne que la relaxation en continu et la relaxation en nombres entiers mixtes, moyennant un effort calculatoire supplémentaire. Plusieurs pistes de recherche restent à explorer dans le choix des multiplicateurs de cette relaxation de manière à améliorer son comportement. Il serait également intéressant de comparer expérimentalement les autres relaxations citées dans cette partie (relaxation surrogate, Lagrangienne et composite) sur les instances du MMKP et d'autres variantes du sac à dos. D'autres améliorations peuvent être envisagées, par exemple le remplacement de la recherche locale dans les algorithmes itératifs par une métaheuristique. La taille des problèmes réduits et du problème lui-même a tendance à augmenter durant le processus itératif en raison de l'ajout des pseudo-coupes, ce qui peut ralentir le temps d'exécution des algorithmes. Pour remédier à cet inconvénient, nous pouvons aussi intégrer la mémoire adaptative de la recherche Tabou dans le processus de génération des pseudo-coupes. Enfin, les algorithmes itératifs proposés fournissent des bornes inférieures et supérieures de bonne qualité dès les premières itérations. Ces bornes peuvent être exploitées dans un algorithme exact de type évaluations et séparations.

Partie 3 : Gestion de perturbations dans le domaine aérien

Dans cette partie nous nous intéressons à un problème de gestion de perturbations dans le domaine aérien. Ce travail se situe dans le cadre de notre participation à un challenge international organisé par la Société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF[2]). Le sujet proposé par la société Amadeus[3] vise à rétablir une situation normale en cas de perturbations en tenant compte simultanément des avions et des passagers plutôt que selon la hiérarchie classique : avions, équipages, passagers. L'objectif est de mieux prendre en compte le ressenti des passagers en introduisant une mesure de la désutilité des passagers en fonction du retard observé à l'arrivée ou de l'annulation du voyage. Pour cela, nous cherchons à minimiser une somme pondérée des différents coûts associés aux vols de la compagnie. Devant la difficulté du problème, nous proposons une approche heuristique pour le résoudre (Mansi, Wilbaut, Hanafi et Clautiaux (2009), Hanafi, Wilbaut, Mansi et Clautiaux (2009)). Notre méthode est divisée en deux étapes principales. La première essaie de rétablir la réalisabilité dans le voisinage de la situation juste avant les perturbations. La seconde étape tente d'améliorer cette solution en alternant les phases constructives et destructives dans une stratégie d'oscillation. Nous hybridons dans ces deux phases la programmation mathématique avec des heuristiques. Cette approche nous a permis d'être classés seconds parmi 29 équipes participantes.

Cette partie est organisée comme suit : dans la Section 3.1 nous donnons une description du problème puis, dans la Section 3.2, nous présentons une revue de la littérature sur des problèmes connexes. Le problème a été décomposé en deux grands sous-problèmes que nous traitons indépendamment. Le premier, qui est géré dans la première phase de notre approche, vise à obtenir une solution réalisable dans le voisinage du plan de vols initial. Ce sous-

[2] http ://www.roadef.org/ ou http ://challenge.roadef.org/2009/index.fr.htm

[3] http ://www.amadeus.com/amadeus/amadeus.html

problème peut être modélisé en un problème linéaire en nombres mixtes que nous résolvons en appliquant des techniques de relaxation. Cette première phase fait l'objet de la Section 3.3. Le deuxième sous-problème vise à améliorer la solution trouvée dans la première phase. Cela est réalisé avec une heuristique hybride où nous générons et supprimons des vols pour les avions et des itinéraires pour les passagers afin de filtrer les meilleurs couplages entre routes et itinéraires à l'aide de la programmation mathématique. Cette deuxième phase entre dans le contexte des méthodes heuristiques basées sur l'oscillation, et elle traitée en détails dans la Section 3.4. Notre méthode a donné des résultats intéressants, notamment pour des instances de grande taille. Ces résultats sont décrits dans la Section 3.5.

3.1 Description du Problème

Les compagnies aériennes définissent initialement un *programme de vols* qui correspond à un ensemble de vols réalisés sur une période donnée. En cas de perturbations dans ce programme, ces compagnies cherchent à rétablir l'ordre normal des opérations le plus rapidement possible. Il est capital que le retour à une situation normale soit obtenu dans une période bien déterminée appelée *période de recouvrement*, que nous nommerons également dans la suite par *période d'interventions*, car elle représente la période durant laquelle il est possible de modifier le programme de vols.

Les avions, les aéroports, les passagers et les équipages sont les acteurs principaux considérés dans notre problème. Chaque acteur est caractérisé par un ensemble de propriétés et doit respecter un ensemble de contraintes. Dans cette étude nous ne considérons pas les équipages.

3.1.1 La flotte des appareils

L'ensemble des avions constitue la *flotte* d'une compagnie aérienne. Chaque avion se caractérise entre autres par un modèle (par exemple Boeing 747, Airbus A320) et la capacité de chacune de ses trois cabines (first, business, économique). Les avions d'un même modèle ont des spécificités communes telles que le temps de *réengagement* de l'appareil, le *temps de transit*, et la distance maximale franchissable. Le temps de *réengagement* correspond au temps de *débarquement / embarquement* des passagers, au nettoyage de l'appareil et au changement éventuel d'équipage ; le temps de transit correspond au temps de préparation nécessaire à la réalisation du vol suivant. Des sous-ensembles d'appareils ayant des caractéristiques communes sont regroupés sous forme de *famille*. L'intérêt de cette notion de

famille est de pouvoir inter-changer leurs équipages (il est ainsi possible de remplacer un avion prévu initialement pour un vol par un autre avion de la même famille). L'ensemble des vols successifs effectués par un avion est une *rotation.* Chaque vol se définit par ses aéroports d'origine et de destination, sa durée, son type (domestique, continental ou international) et ses dates et heures de départ et d'arrivée. Dans la suite nous utilisons le terme *route* pour une sous-partie de la rotation d'un avion, c'est-à-dire une sous-suite de vols successifs. Notons que des vols additionnels, hors aériens et ne pouvant être modifiés, peuvent exister dans le programme de vols initial et sont opérés par des appareils appropriés. Cette situation se retrouve en pratique dans le cas d'aéroports desservant une même ville (par exemple Charles de Gaulle et Orly pour Paris). Ces vols sont considérés comme des vols fictifs et sont assurés par une flotte de bus, métros ou trains.

La flotte est soumise à un ensemble de *contraintes opérationnelles* relatives aux rotations, aux capacités des cabines, aux maintenances et aux temps minimums de réengagement ou de transit entre deux vols consécutifs. Les contraintes de maintenance pour un appareil ont pour objectif de ne pas excéder un temps maximum de vol entre le début de la période de recouvrement et sa date de maintenance. Ces contraintes fortes ne peuvent être relâchées, en particulier pour des raisons de sécurité. De plus, les opérations de maintenance s'effectuent à un aéroport donné et à une date donnée. L'appareil est indisponible pendant toute la durée correspondante.

3.1.2 Le réseau d'aéroports

Le deuxième acteur du problème est l'ensemble des aéroports couverts par une compagnie. Celui-ci définit le réseau de la compagnie. Dans notre problème, chaque aéroport possède un nombre maximal d'atterrissages et de décollages des avions pouvant être pris en charge dans un intervalle de temps donné. Dans un souci de simplicité, un intervalle correspond en général à une heure pleine, par exemple [7:00,7:59], ou plus généralement [H, H+1[. Nous parlons aussi de *tranche horaire.* Les capacités horaires ne sont pas considérées sur des tranches glissantes. Autrement dit, pour chaque intervalle [H,H+1[, le nombre d'atterrissages et de décollages ne pourra excéder les capacités de cette tranche (hors vols fictifs correspondant au transport non aérien). Notons que dans un souci de simplicité certaines contraintes réelles liées aux aéroports sont ignorées telles que les contraintes de capacité (nombre d'appareils présents à l'aéroport). Finalement, le temps de parcours entre deux aéroports est considéré comme une donnée constante (quel que soit l'appareil utilisé).

3.1.3 Les passagers

Le troisième acteur majeur du problème est l'ensemble des passagers. Les passagers effectuent des *réservations* sur les vols proposés par la compagnie aérienne. Une réservation est définie par un numéro clé (ou identifiant), le nombre de passagers concernés, le prix moyen payé par passager, le descriptif de l'*itinéraire* et le caractère aller/retour. Un itinéraire est une succession de couples (vol, cabine) précisant les origines et destinations de chaque portion de la réservation et le type de cabine associé. Une réservation regroupe ainsi un ensemble de passagers ayant un même itinéraire.

L'ensemble des passagers est soumis au respect d'un temps minimum de correspondance (au moins 30 minutes entre deux vols consécutifs sur un itinéraire dans ce problème). Lors de la ré-accommodation des passagers, les règles suivantes doivent être respectées : annulation du voyage lorsque l'un des aéroports concernés n'est pas accessible, début de l'itinéraire de remplacement postérieur à celui de l'itinéraire initial, ré-accommodation prioritaire des passagers en correspondance ou sur un itinéraire retour sous peine de pénalités très élevées, et retard global maximal de 18 heures pour un vol domestique ou continental et de 36 heures pour un vol international (annulation de l'itinéraire si cette contrainte ne peut pas être respectée).

3.1.4 Perturbations et objectifs

Une des particularités de ce problème est la possibilité de faire face à des perturbations assez diverses. Le programme de vols peut ainsi être soumis à des retards (préparation de l'appareil plus longue qu'en temps normal, attente de passagers ou de membres de l'équipage par exemple) et annulations de vols. Un avion peut aussi être indisponible pour une période déterminée (suite à un problème mécanique par exemple), et le nombre de décollages et d'atterrissages possibles à un aéroport peut être réduit sur une tranche horaire donnée (par exemple lors de conditions météorologiques défavorables ou d'une grève du personnel).

Le nouveau programme de vols résulte de la prise d'un certain nombre de décisions. Celles-ci se résument à l'annulation et aux retardements intentionnels de vols, au changement d'appareil (au sein d'une même famille) et à la création éventuelle de nouveaux vols. Il est également possible de réaffecter les passagers, éventuellement en divisant les réservations et en créant ainsi de nouvelles réservations. Notons que les vols programmés avant la période de recouvrement ne peuvent pas être altérés. De même, les parties des itinéraires des passagers déjà entamées avant le début de cette période ne peuvent pas être modifiées.

L'objectif du problème est de minimiser les coûts induits pour la compagnie et les impacts éventuels dans le nouveau programme de vols. Cet objectif intègre un ensemble de coûts opérationnels, de coûts liés aux retards et annulations des vols et de coûts relatifs à la désutilité des passagers. Dans le cadre de ce travail, les coûts opérationnels sont ramenés aux coûts nécessaires pour la réalisation d'un vol en fonction du type de l'appareil et de la durée du vol. Les coûts de retard et d'annulation des vols sont calculés selon la réglementation européenne. Les coûts relatifs à la désutilité des passagers se calculent essentiellement en fonction du retard global (le cas échéant, en fonction de l'annulation du voyage) et d'une pénalité en cas de déclassement (passage à une cabine inférieure). Le coût d'annulation d'un voyage est nettement supérieur au coût de retard maximal sur ce même voyage, et il est de plus prohibitif dans le cas d'un itinéraire retour ou en correspondance. Finalement, en vue d'un retour à la normale à la fin de la période de recouvrement, nous pénalisons les solutions qui ne respectent pas les contraintes de positionnement des appareils (présence d'un certain nombre d'appareils ayant un modèle et une configuration donnés à chaque aéroport).

3.2 Etude de l'existant

Les problèmes d'optimisation dans le domaine aérien peuvent être décomposés en deux grandes catégories : la planification et la gestion de revenu. La gestion de revenu concerne l'allocation des places et la politique de réservation, et n'est pas considérée dans ce travail. La planification aérienne va de la construction à l'exécution du plan de vols et peut être divisée en plusieurs sous-catégories qui dépendent du moment où le problème intervient. Dans le domaine aérien trois différents niveaux sont généralement mentionnés : le niveau stratégique, le niveau tactique et le niveau opérationnel. Dans notre travail nous nous concentrons en particulier sur ce dernier. Le niveau stratégique vise à construire le programme de vols, c'est-à-dire à décider où et quand les vols s'effectuent. Plusieurs méthodes et modèles sont proposés dans la littérature pour traiter les problèmes de ce niveau pour lesquels les prévisions de la demande de transport sont nécessaires (voir Gopalan et Talluri (1998a) par exemple). Dans notre problème, nous considérons le programme initial comme une entrée. Le but du niveau tactique est de décider quel avion est affecté au vol et quel équipage est affecté à l'avion. Ce niveau regroupe plusieurs problèmes comme l'affectation de la flotte (*fleet assignment problem*), l'affectation de l'équipage (Lettovsky, Johnson et Nemhauser (2000)) (qui se décompose en deux grands problèmes : le *crew pairing* et le *crew assignement problem*). Un grand nombre de travaux peuvent être recensés dans la littérature pour

l'affectation de la flotte (voir par exemple Hane, Barnhart, Johnson, Marsten, Nemhauser et Sigismondi (1995)) et le problème d'affectation des vols aux appareils (*tail assignment problem*, Gabteni et Grönkvist (2008)). Dans ce dernier, quelques perturbations peuvent être prises en compte, comme par exemple les maintenances des avions (Gopalan et Talluri (1998b), Sriram et Haghani (2003)). Dans certains travaux, le problème d'affectation de l'équipage du niveau tactique et celui du routage des avions du niveau stratégique sont considérés simultanément (Weide, Ryana et Ehrgott (2009)). Comme pour le niveau stratégique, nous considérons le niveau tactique comme une entrée du problème.

Le problème traité, nommé "**Gestion de perturbations dans le domaine aérien**" (**GPDA**), est un problème réel très difficile, avec la particularité de vouloir prendre en compte le ressenti passager et plusieurs types de perturbations. La difficulté du problème peut être justifiée facilement par le nombre très important de contraintes et par sa nature non linéaire. Plusieurs variantes du GPDA ont fait l'objet d'une large étude théorique et pratique.

Une des premières approches en recherche opérationnelle pour résoudre un problème de perturbations dans le domaine aérien est due à Teodorovic et Guberinic (1984). Dans cet article, la perturbation considérée est l'indisponibilité des avions. L'approche est basée sur la permutation possible entre les avions et le retardement intentionnel des vols pour éviter d'aggraver la situation par rapport aux passagers. Le modèle proposé est résolu exactement par une méthode de *Branch-and-Bound*. L'approche est étendue dans un second article de Teodorovic et Stojkovic (1990), dans lequel les auteurs permettent l'annulation des vols, le retard des vols et la permutation des avions. Le modèle proposé cette fois est résolu par une heuristique basée sur la programmation dynamique. L'objectif dans un premier temps est de réduire les annulations des vols, puis de réduire le retard total des passagers dans un second temps. Dans un troisième article, Teodorovic et Stojkovic (1995) proposent un processus plus complet pour résoudre le problème de perturbations aériennes, en prenant en compte les contraintes de maintenance des avions ainsi que des contraintes sur les équipages. Le problème d'indisponibilité des avions a également été traité par Jarrah, Yu, Krishnamurthy et Rakhit (1993) qui l'ont présenté sous la forme de deux modèles de flot dans un réseau. Cao et Kanafani (1997a, 1997b) ont étendu cette approche pour intégrer l'annulation des vols et la gestion de plusieurs aéroports. Ils ont présenté un programme quadratique en nombre binaires qui est résolu d'une façon heuristique sans considérer les contraintes de maintenance et les contraintes sur les équipages. Arguello, Bard et Yu (1997) ont présenté une procédure pour générer des routes lors d'une indisponibilité d'avions. L'heuristique permet des annulations,

des retards et la permutation des vols. Les maintenances d'avions et les équipages ne sont cependant pas pris en compte.

Yan et Yang (1996) et Yan et Tu (1997) ont proposé quatre modèles pour traiter d'autres problèmes de perturbations aériennes. Dans le premier modèle, il est possible d'annuler des vols afin de réparer le programme de vols. Le deuxième modèle permet en plus l'utilisation de la capacité résiduelle des avions pour réaffecter les passagers. Le troisième modèle considère les annulations et les retards des vols, et le dernier modèle incorpore toutes les décisions précédemment considérées. La permutation des vols dans la même flotte d'avions est également permise dans tous les modèles. L'objectif est toujours de minimiser le coût de réparation du programme de vols, en respectant les contraintes des passagers. Les deux premiers modèles sont des problèmes de flot purs et sont faciles à résoudre. Les deux autres modèles sont des modèles de flot dans le réseau avec des contraintes supplémentaires. Ils sont résolus à l'aide d'une heuristique basée sur la relaxation Lagrangienne. Dans ces approches, la seule perturbation considérée est à nouveau l'indisponibilité des avions. Thengvall, Bard et Yu (2000) ont étendu les techniques précédentes de manière à pouvoir manipuler des annulations, retards et permutations de vols. Ils ne considèrent cependant pas les contraintes de maintenances des avions et les contraintes des équipages. Yan et Lin (1997) abordent un autre problème critique dans le domaine aérien qui consiste en la fermeture provisoire d'un aéroport. La différence par rapport aux problèmes précédents est qu'ici un seul aéroport est indisponible au lieu d'un ou plusieurs avions. La fermeture d'un aéroport est un exemple de perturbation pouvant entraîner de sérieuses conséquences. Cependant le modèle qui est présenté prend seulement en compte la permutation, le retard et l'annulation de vols, ce qui signifie que tous les vols qui arrivent ou partent de l'aéroport fermé doivent être annulés. Thengvall, Yu et Bard (2001) ont continué le travail entrepris dans Thengvall, Bard et Yu (2000) pour prendre en compte les perturbations associées aux fermetures d'aéroports. Love, Sorensen, Larsen et Clausen (2001) ont développé une recherche locale pour résoudre ce problème. Ils considèrent les annulations, les retards et les permutations entre les avions du même type. Plus récemment, Andersson (2006) a présenté une métaheuristique pour résoudre le problème de perturbations de vols. Il a montré en particulier qu'une méthode de recherche Tabou peut produire de bonnes solutions dans un temps relativement court.

Quelques articles dans la littérature traitent aussi des problèmes proches du GPDA avec plusieurs perturbations. Ainsi par exemple Bratu et Barnhart (2006) considèrent simultanément l'avion, les équipages et les passagers en déterminant les départs des vols pour réduire au minimum les frais d'exploitation des avions, le retard estimé des passagers et les

coûts de perturbations. Clausen, Larsen et Rezanova (2009) ont fourni très récemment une étude bibliographique sur la gestion de perturbations aériennes incluant les avions, les équipages, les passagers et ont présenté plusieurs techniques de résolution et modélisation pour un retour à la normale. Le lecteur peut également se référer à Clarke (1998), Yu (1998) ou Yu et Qi (2004). D'autre part, un grand nombre de problèmes du domaine aérien sont des problèmes d'ordonnancement dont une étude de la littérature peut être trouvée par exemple dans Chrétienne et Picouleau (1995).

3.3 Rétablir la réalisabilité / Générer une solution initiale

Dans une première phase, notre objectif consiste à trouver une solution réalisable du problème. Etant donné l'importance des coûts d'annulation des itinéraires des passagers, nous considérons que cette solution doit maximiser le nombre de passagers transportés et minimiser le retard total (des passagers et des avions). Nous cherchons également à ne pas trop nous éloigner de la situation initiale qui peut être considérée comme une *bonne* solution. L'obtention d'un plan de vols respectant l'ensemble des contraintes s'avère être une tâche non triviale sur certaines instances. Cela est en particulier le cas lorsqu'une combinaison conséquente de perturbations sur les aéroports, la maintenance des appareils, et les retards de vols se produit. Nous résolvons un programme linéaire en nombres entiers mixtes présenté dans la suite pour répondre à ce problème. Nous commençons par introduire un ensemble de notations associées aux données du problème. Dans la suite nous notons vol i le $i^{ème}$ vol dans la séquence des vols programmés de l'avion concerné, et i- (resp. i+) le vol qui précède (resp. suit) le vol i pour un avion donné. De plus, pour deux vols i et j, $i \prec j$ signifie que le vol i est programmé avant le vol j et $i \tilde{\prec} j$ signifie que le vol i est programmé juste avant le vol j.

Données du problème :

- T^- : date de début de la période d'intervention.
- T^+ : date de fin de la période d'intervention.
- K : ensemble des aéroports (k un aéroport donné).
- A : ensemble des avions (a un avion donné).
- A^{maint} : ensemble des avions ayant une maintenance durant la période ($A^{maint} \subseteq A$).
- P : ensemble des groupes de passagers (p un groupe de passagers donné et G_p le cardinal de p).

- $T^-_{i,a}$: date de début du vol i effectué par l'avion a (y compris le retard déjà annoncé).
- $T^+_{i,a}$: date de fin du vol i effectué par l'avion a (y compris le retard déjà annoncé).
- i^m_a : dernier vol programmé avant la maintenance de l'avion a.
- E_a : temps nécessaire de réengagement de l'avion a.
- M_a : date de début de la maintenance de l'avion a lorsque celle-ci est prévue ($+\infty$ sinon). La durée de maintenance de l'avion a est considérée comme étant la durée d'un intervalle d'indisponibilité de l'avion.
- I^-_a : date de début de la période d'indisponibilité de l'avion a si elle existe.
- I^+_a : date de fin de la période d'indisponibilité de l'avion a si elle existe.

Comme mentionné dans la Section 3.1.2, chaque aéroport k est caractérisé par des capacités horaires de décollage et d'atterrissage (sans considérer de fenêtres glissantes dans chaque tranche horaire) que nous représentons par :

- $T^-_{k,h}$: début de la tranche h de l'aéroport k.
- $T^+_{k,h}$: fin de la tranche h de l'aéroport k.
- H_k : ensemble des tranches de l'aéroport k.

Pour chaque tranche h de l'aéroport k, nous notons :

- $C^-_{k,h}$: capacité de décollage de l'aéroport k pendant la tranche h.
- $C^+_{k,h}$: capacité d'atterrissage de l'aéroport k pendant la tranche h.

De plus, nous introduisons les notations suivantes :

- $k^-(i,a)$: aéroport de départ du vol i effectué par l'avion a.
- $k^+(i,a)$: aéroport d'arrivée du vol i effectué par l'avion a.
- $D_{k,k'}$: temps nécessaire pour aller de l'aéroport k à l'aéroport k'.
- V_p : suite des vols (*indice*, *avion*) initialement prévus pour le groupe de passagers p (avec *indice* qui représente le rang du vol dans la rotation de l'*avion*).
- L : constante importante qui peut prendre la durée de toute la période en minutes.

Nous présentons dans la suite la modélisation proposée pour répondre au problème traité dans cette première phase, à savoir trouver une solution réalisable du problème dans le voisinage du plan de vols initial en respectant au mieux les contraintes considérées comme dures du problème. Ainsi les contraintes à respecter concernent les appareils, les aéroports, les

itinéraires, les rotations, la capacité des cabines des appareils, les maintenances, les capacités aéroportuaires, le temps minimum de correspondance et le temps minimum de réengagement ou de transit selon les vols. Nous décomposons l'ensemble des contraintes en trois sous-ensembles : les contraintes sur les avions, sur les aéroports et sur les passagers.

3.3.1 Contraintes sur les avions

Du point de vue des avions, notre objectif est d'assurer la continuité des vols effectués et les contraintes de maintenance.

La Figure 3.1 illustre le parcours initialement prévu d'un avion a sous la forme d'un graphe spatio-temporel dans lequel un sommet est un couple (*aéroport*, *instant*). La notion d'instant correspond ici à la date de départ du vol depuis le sommet associé. Selon ce programme initial, nous savons que l'avion a est à l'aéroport k_1 à l'instant t_1, qu'il doit partir pour l'aéroport k_2 puis pour l'aéroport k_3. Ensuite, il revient à l'aéroport k_2 avant de repartir à l'aéroport k_1 où sera menée une opération de maintenance, et sa rotation se termine par un vol vers l'aéroport k_4.

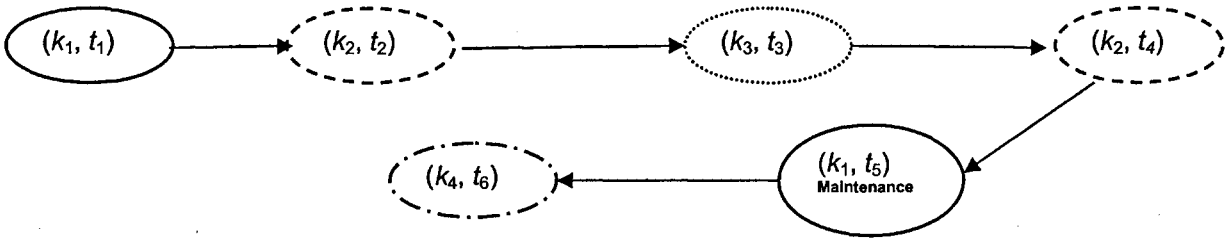


Figure 3.1 : Continuité des vols d'un avion.

Dans le cas d'une perturbation dans le réseau, nous nous permettons de modifier le parcours initial des avions pour assurer la réalisabilité. Pour modifier le parcours d'un avion sans trop perturber le plan de vols initial, nous avons opté pour la suppression de circuits effectués par un avion, ce qui assure la satisfaction des contraintes de continuité des vols de chaque avion. Le traitement effectué pour l'avion a est le même pour tous les autres avions et il se fait simultanément. Pour cela, nous avons introduit la notion de *vol fictif*, qui correspond à l'annulation d'un circuit et laisse l'avion à son aéroport courant. Sur l'exemple de la Figure 3.2, si l'avion a est dans l'aéroport k_1 à l'instant t_1 alors il est possible que cet avion n'effectue aucun vol avant sa maintenance. Dès lors, il reste à l'aéroport k_1 jusqu'à l'instant t_5. Cela se traduit par le vol fictif (en pointillés) de l'état (k_1, t_1) vers l'état (k_1, t_5). De la même manière, cet avion peut effectuer le vol vers l'aéroport k_2 puis rester à cet aéroport jusqu'à l'instant t_4. Ce type de suppression assure donc de conserver la continuité des vols pour tous les avions.

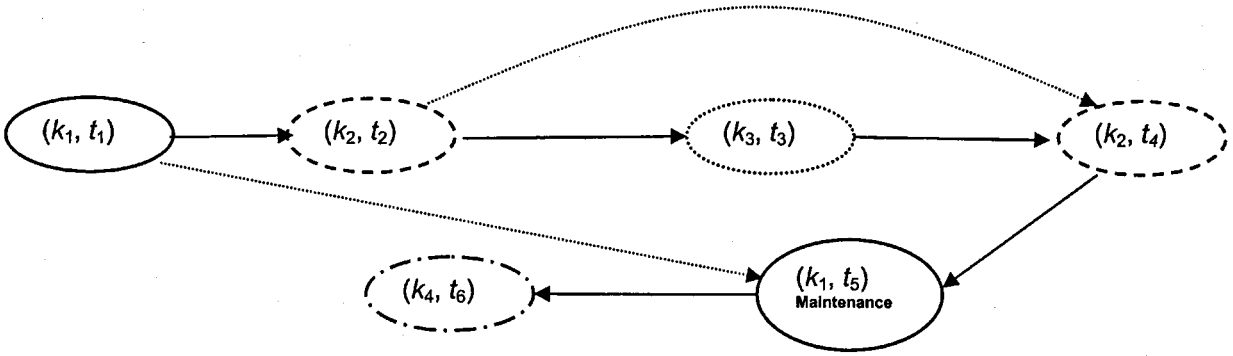


Figure 3.2 : Vols fictifs

L'utilisation de vols fictifs ne peut pas toujours assurer le respect des contraintes de maintenance pour tous les avions dans une fenêtre de temps précise et un aéroport précis. Supposons dans l'exemple de la Figure 3.2 que l'avion a soit dans l'aéroport k_2 à l'instant t_2 et que la perturbation provoque l'impossibilité d'effectuer le vol entre l'aéroport k_2 à partir de l'instant t_4 et l'aéroport k_1 (par exemple si la capacité d'atterrissage en k_1 est nulle après t_4). Dans ce cas l'avion ne pourra pas être à l'aéroport k_1 à temps pour sa maintenance. Il est alors nécessaire de considérer la *création* de vols réels. Dans notre exemple, deux vols peuvent être créés : le premier s'effectue de l'aéroport k_2 à partir de l'instant t_2 vers l'aéroport k_1 et le deuxième de l'aéroport k_3 à partir de l'instant t_3 vers l'aéroport k_1 (Figure 3.3).

Nous autorisons ainsi la modification des routes des avions avec une maintenance en annulant des circuits sur le trajet et/ou en faisant des "sauts" jusqu'à la destination de maintenance à temps. Un saut ne peut être effectué que si l'avion a a assuré les vols précédents en continu. Les sauts considérés sont de dimension 1 (i.e., un seul vol peut être créé pour chaque avion), de manière à éviter l'explosion combinatoire des cas possibles.

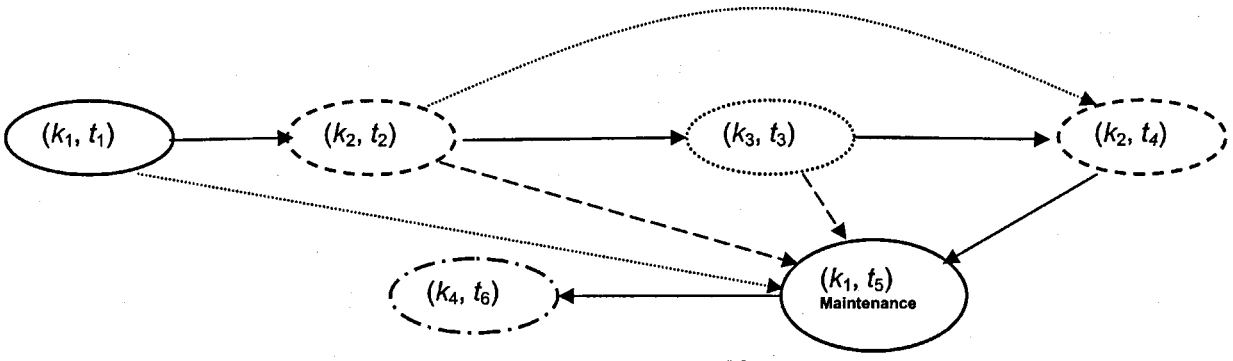


Figure 3.3 : Vols créés

Pour modéliser les contraintes de continuité des vols et des sauts pour les avions ayant une maintenance, nous introduisons les variables suivantes :

$u_{i,a}$: variable binaire égale à 1 si l'avion a effectue le vol existant i, 0 sinon.

$v_{i,a}$: variable binaire égale à 1 si l'avion a effectue le vol fictif i, 0 sinon.

$w_{i,a}$: variable binaire égale à 1 si l'avion a effectue le vol créé i, 0 sinon (il n'existe que pour les avions avec maintenance).

$r_{i,a}$: retard associé au vol i effectué éventuellement par l'avion a (variable continue).

$t_{i,a}$: instant de départ associé au vol créé i effectué éventuellement par l'avion a.

Nous présentons dans la Figure 3.4 le parcours de l'avion a avec les détails de chaque type de vol. Nous considérons d'abord les vols existants avec par exemple le vol (1, a) défini par l'aéroport de départ k_1, l'aéroport d'arrivée k_2, la date de départ T_{1a}^- et la date d'arrivée T_{1a}^+. Les deux variables affectées à ce vol sont u_{1a} qui indique la considération ou non de ce vol et la variable r_{1a} qui indique le retard pour ce vol. Un exemple de vol fictif est donné par le passage de (k_1, t_1) à (k_1, t_5). Ce vol est défini par l'aéroport de repos k_1, la date de début de ce repos T_{1a}^- et la date de fin de repos $T_{5a}^- - E_a$. Nous affectons une variable à ce vol v_{1a} qui indique la considération ou non de ce vol fictif. Un exemple de vol créé est donné par le passage de (k_2, t_2) à (k_1, t_5). Ce vol est défini par l'aéroport de départ k_2, l'aéroport d'arrivée k_1 (l'aéroport de maintenance), la date minimum de départ possible $T_{1a}^+ + E_a$ et la date d'arrivée maximale M_a (la date de début de maintenance). Nous affectons deux variables à ce vol : w_{2a} qui indique la création ou non de ce vol et t_{2a} qui indique la date de départ de ce vol s'il est créé.

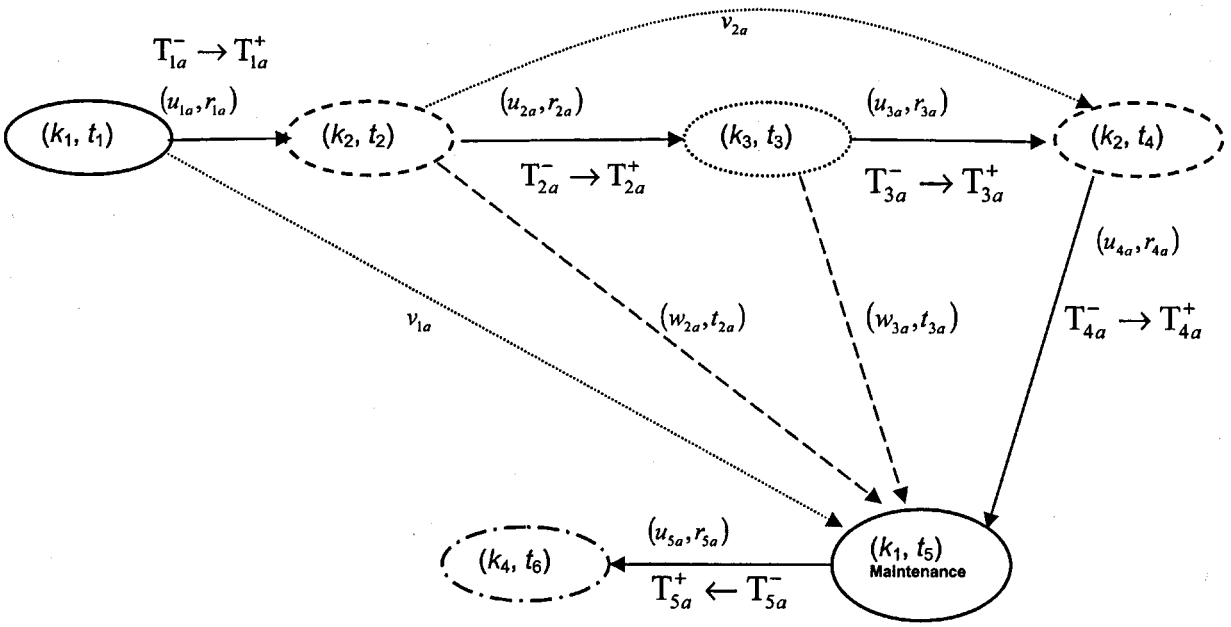


Figure 3.4 : Les différents vols du réseau

La nouvelle rotation d'un avion a est ainsi composée d'une série de vols consécutifs de trois types : existants (déjà programmés avant la perturbation et regroupés dans l'ensemble VE_a), fictifs où l'avion reste dans l'aéroport n'effectuant aucun vol réel (regroupés dans l'ensemble VF_a), et créés pour les avions avec maintenance (regroupés dans l'ensemble VC_a). Notre construction de nouvelles routes impose que $\text{Max}(|VF_a|, |VC_a|) \leq |VE_a|$ car les rotations alternatives sont obligatoirement plus courtes que l'initiale.

La modélisation des contraintes de continuité des vols et des sauts se définit alors comme suit :

$$(C-avion)\begin{cases} u_{i-,a} + v_{\hat{i},a} \geq u_{i,a} + v_{i,a} + w_{i,a} & a \in A, i \neq i_a^m \quad (1\text{-a}) \\ u_{i_a^m,a} + v_{\hat{i}_a^m,a} + \sum_{j=1}^{i_a^m} w_{j,a} = 1 & a \in A^{ma\,\text{int}} \quad (1\text{-b}) \\ r_{i,a} + \mathrm{T}_{i,a}^{+} + \mathrm{E}_a - L(1-u_{i,a}) \leq r_{j,a} + \mathrm{T}_{j,a}^{-} + L(1-u_{j,a}) & a \in A, i \prec j \quad (1\text{-c}) \\ r_{i-,a} + \mathrm{T}_{i-,a}^{+} + \mathrm{E}_a - \mathrm{L}(1\text{-}u_{i-,a}) \leq t_{i,a} + \mathrm{L}(1-w_{i,a}) & a \in A^{ma\,\text{int}}, i \prec i_a^m \quad (1\text{-d}) \\ r_{i,a} \leq \mathrm{M}_a - \mathrm{T}_{i,a}^{+} & a \in A^{ma\,\text{int}}, i \leq i_a^m \quad (1\text{-e}) \\ t_{i,a} + \mathrm{D}_{k^-(i,a),k^+(i_a^m,a)} \leq \mathrm{M}_a & a \in A^{ma\,\text{int}}, i \leq i_a^m \quad (1\text{-f}) \end{cases}$$

La contrainte (1-a) est une contrainte de flot qui précise que l'avion ne peut pas quitter un aéroport s'il n'a pas atterri dans cet aéroport ou s'il n'est pas déjà présent dans cet aéroport. Notons que toutes les variables $v_{i,a}$ ne sont pas forcément présentes dans la contrainte (1-a) puisque leur existence dépend du circuit présent dans la route initiale de l'avion a. De même pour la présence des variables $w_{i,a}$, sauf si l'avion associé a une maintenance et que le vol associé est programmé avant la maintenance. Ceci reste vrai pour toutes les contraintes du modèle impliquant ces variables. De plus, le vol fictif $v_{i,a}$ correspond dans (1-a) au circuit juste avant le vol i. La contrainte (1-b) est considérée seulement pour les avions qui ont une maintenance. Elle assure le respect de la maintenance en obligeant le dernier vol avant maintenance référencé par i_a^m pour l'avion a à avoir lieu, soit par le vol existant, soit par un vol fictif (noté $\hat{i}_a^m$, il correspond au circuit juste avant le vol $(i_a^m +)$), soit par un nouveau vol.

La contrainte (1-c) indique que chaque enchaînement de vols (i, j) pour un avion doit respecter le temps minimal de réengagement pour l'avion, à condition que le vol i soit programmé initialement avant le vol j. Autrement dit, la date d'atterrissage du vol i plus le temps de réengagement doit être postérieure à la date de décollage du vol j. Si un des deux vols est annulé alors la contrainte (1-c) est redondante car la constante L est suffisamment grande pour la relaxer. La contrainte (1-d) joue le même rôle que la contrainte (1-c) mais cette

fois entre les vols prévus et ceux éventuellement créés. De la même façon, un vol créé ne doit pas décoller sans respecter le temps minimum de réengagement pour l'avion.

La contrainte (1-e) fixe une borne supérieure sur le retard qui peut être considéré pour un vol d'un avion qui a une maintenance, la date d'atterrissage de tous les vols prévus avant la maintenance ne devant pas franchir cette date. La contrainte (1-f) a exactement le même rôle que (1-e) mais elle borne les dates d'atterrissage des vols éventuellement créés avec la durée du vol prise en compte. Rappelons que $D_{k^-(i,a),k^+(i_a^m,a)}$ est la durée du vol éventuellement créé de l'avion a entre l'aéroport de départ du vol i et l'aéroport de maintenance de a.

3.3.2 Contraintes sur les aéroports

Nous nous intéressons ici aux contraintes liées aux aéroports, c'est-à-dire les contraintes de capacité d'atterrissage et de décollage de chaque tranche horaire de chaque aéroport. La perturbation qui peut arriver pour les aéroports est la diminution des capacités d'atterrissage et de décollage qui nous obligent à annuler les vols prévus dans ces tranches et/ou à les décaler. La Figure 3.5 montre comment un vol peut être décalé et les différents cas associés.

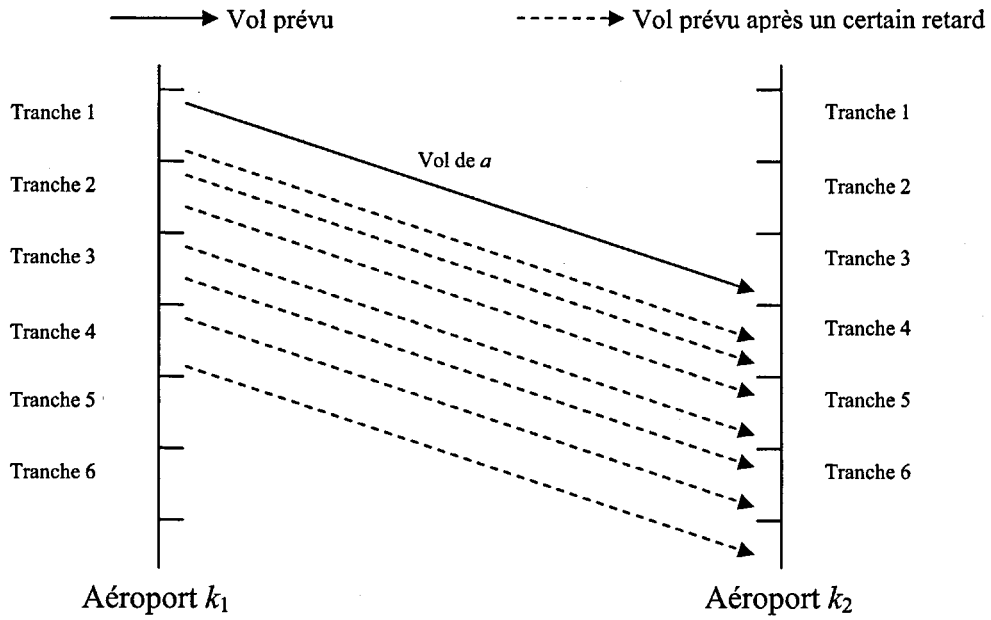


Figure 3.5 : Capacités des aéroports

D'après la Figure 3.5, l'avion a effectue un vol de l'aéroport k_1 à la première tranche horaire vers l'aéroport k_2 et arrive à la troisième tranche horaire dans le plan de vols initial. Après une perturbation qui réduit drastiquement la capacité de décollage de la tranche 1 de l'aéroport k_1, le décollage de certains avions dans cette tranche devient impossible. Dans ce cas, il est inévitable de supprimer le vol d'un avion ou de le retarder. Le même problème peut

se produire pour l'atterrissage. Retarder un vol nous oblige à prendre en considération la situation liée aux tranches suivantes. Remarquons que plusieurs cas peuvent alors se produire en fonction des capacités des deux aéroports aux différentes tranches et de la durée du vol entre k_1 et k_2. Ainsi par exemple si c'est la capacité d'atterrissage qui pose problème, il est peut être possible de décaler le vol en restant dans la même tranche de départ comme l'indique le premier trajet en pointillé. Il est par contre interdit d'avancer un vol. Pour modéliser ces contraintes, nous introduisons les variables suivantes :

$x^-_{i,a,k,h}$: variable binaire égale à 1 si le décollage du vol i de l'avion a s'effectue dans la $h^{ème}$ tranche de l'aéroport k.

$x^+_{i,a,k,h}$: variable binaire égale à 1 si l'atterrissage du vol i de l'avion a s'effectue dans la $h^{ème}$ tranche de l'aéroport k.

Les contraintes liées aux aéroports peuvent alors être modélisées comme suit :

$$C-aéroport\begin{cases}
\sum_{h=1}^{H_{k^-(i,a)}} T^-_{k^-(i,a),h} \times x^-_{i,a,k^-(i,a),h} \le r_{i,a} + T^-_{i,a} \le \sum_{h=1}^{H_{k^-(i,a)}} T^+_{k^-(i,a),h} \times x^-_{i,a,k^-(i,a),h} & a \in A, i \in VE_a \quad (1\text{-g}) \\
u_{i,a} = \sum_{h=1}^{H_{k^-(i,a)}} x^-_{i,a,k^-(i,a),h} & a \in A, i \in VE_a \quad (1\text{-h}) \\
\sum_{h=1}^{H_{k^+(i,a)}} T^-_{k^+(i,a),h} \times x^+_{i,a,k^+(i,a),h} \le r_{i,a} + T^+_{i,a} \le \sum_{h=1}^{H_{k^+(i,a)}} T^+_{k^+(i,a),h} \times x^+_{i,a,k^+(i,a),h} & a \in A, i \in VE_a \quad (1\text{-i}) \\
u_{i,a} = \sum_{h=1}^{H_{k^+(i,a)}} x^+_{i,a,k^+(i,a),h} & a \in A, i \in VE_a \quad (1\text{-j}) \\
\sum_{h=1}^{H_{k^-(i,a)}} T^-_{k^-(i,a),h} \times x^-_{i,a,k^-(i,a),h} \le t_{i,a} \le \sum_{h=1}^{H_{k^-(i,a)}} T^+_{k^-(i,a),h} \times x^-_{i,a,k^-(i,a),h} & a \in A, i \in VC_a \quad (1\text{-k}) \\
w_{i,a} = \sum_{h=1}^{H_{k^-(i,a)}} x^-_{i,a,k^-(i,a),h} & a \in A, i \in VC_a \quad (1\text{-l}) \\
\sum_{h=1}^{H_{k^+(i,a)}} T^-_{k^+(i,a),h} \times x^+_{i,a,k^+(i,a),h} \le t_{i,a} + D_{k^-(i,a),k^+(i^m_a,a)} \le \sum_{h=1}^{H_{k^+(i,a)}} T^+_{k^+(i,a),h} \times x^+_{i,a,k^+(i,a),h} & a \in A, i \in VC_a \quad (1\text{-m}) \\
w_{i,a} = \sum_{h=1}^{H_{k^+(i,a)}} x^+_{i,a,k^+(i,a),h} & a \in A, i \in VC_a \quad (1\text{-n}) \\
\sum_{i,a,k^-(i,a)=k} x^-_{i,a,k,h} \le C^-_{k,h} & k \in K, h \in H_k \quad (1\text{-o}) \\
\sum_{i,a,k^+(i,a)=k} x^+_{i,a,k,h} \le C^+_{k,h} & k \in K, h \in H_k \quad (1\text{-p})
\end{cases}$$

La contrainte (1-g) vérifie que chaque décollage est dans une tranche existante de l'aéroport de départ. Plus formellement, la date de décollage $r_{i,a} + T^-_{i,a}$ doit être dans une tranche $\left[T^-_{k^-(i,a),h}, T^+_{k^-(i,a),h}\right]$. Rappelons que $k^-(i,a)$ est l'aéroport de départ du vol i de l'avion a et h est le numéro de la tranche dans cet aéroport. L'unicité de la tranche choisie est assurée par la contrainte (1-h) qui exprime que si le vol est annulé alors aucune tranche n'est considérée pour le décollage. Les contraintes (1-i) et (1-j) ont les mêmes interprétations que les contraintes (1-g) et (1-h) pour assurer l'atterrissage d'un avion dans une tranche. Les contraintes (1-k) ... (1-n) ont les mêmes interprétations que les contraintes (1-g) ... (1-j) en considérant les vols créés au lieu des vols existants. Finalement les contraintes (1-o) et (1-p) assurent le respect des capacités de décollage et d'atterrissage respectivement associées à chaque tranche de chaque aéroport.

3.3.3 Contraintes sur les passagers

Les passagers ne sont pas vraiment soumis à des contraintes fortes car nous pouvons annuler tous les itinéraires (dans la période d'intervention) sans menacer la réalisabilité de la solution. Cependant l'annulation de l'itinéraire d'un groupe de passagers a un impact conséquent sur la valeur de la fonction objectif. Cette situation nous oblige à ajouter des contraintes de manière à assurer l'acheminement d'un nombre maximum de passagers à leur destination. Rappelons que pour assurer un itinéraire, aucun des vols associés ne doit être annulé et le temps de transit éventuel doit être au minimum de 30 minutes. La Figure 3.6 présente schématiquement une situation possible pour les passagers.

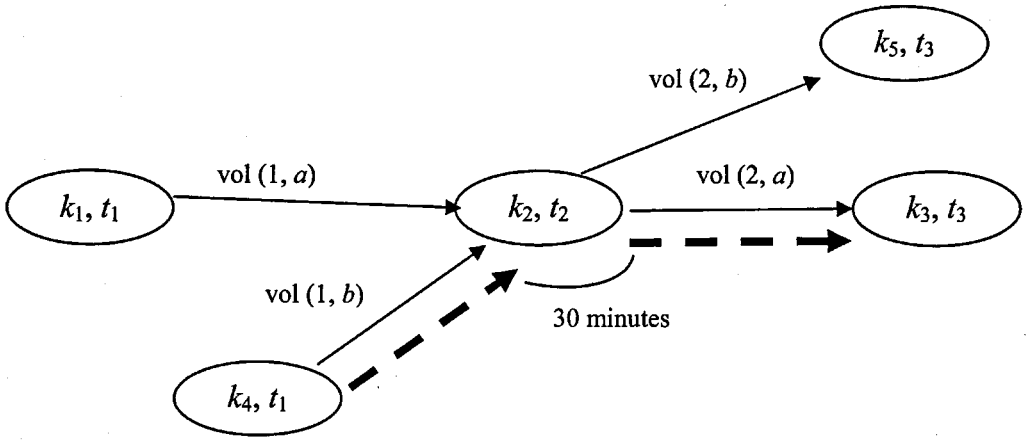


Figure 3.6 : Transit des passagers

La Figure 3.6 illustre le fait que si l'itinéraire initial d'un groupe de passagers est composé de plusieurs vols, par exemple le vol 1 de l'avion b puis le vol 2 de l'avion a, alors ces deux

vols ne doivent pas être annulés et le temps minimum entre la date d'atterrissage du premier vol et la date de décollage du deuxième vol doit être de plus de 30 minutes. Notons que dans cette première phase nous n'envisageons pas la réaffectation des passagers dans d'autres avions. Cela s'explique par le fait que l'annulation des itinéraires n'entraîne pas la non-réalisabilité de la solution et le fait que cela entraîne un éloignement de la situation initiale (en plus d'une complexité supplémentaire du modèle).

Pour assurer les contraintes de transit nous introduisons la variable suivante :

y_p : variable binaire égale à 1 si et seulement si le groupe de passagers p arrive à sa destination finale.

Les contraintes des passagers à respecter sont les suivantes :

$$(C\text{-}passagers)\begin{cases} |V_p| y_p \le \sum_{(i,a)\in V_p} u_{i,a} & \forall p \in P \quad (1\text{-q}) \\ r_{i,a} + \mathrm{T}^{+}_{i,a} + 30 \le r_{j,b} + \mathrm{T}^{-}_{j,b} + \mathrm{L}(1 - y_p) & \forall p \in P, \{(i,a),(j,b)\} \subseteq V_p, (i,a) \precsim (j,b) \quad (1\text{-r}) \end{cases}$$

La contrainte (1-q) indique qu'un itinéraire est annulé si au moins un vol de cet itinéraire est annulé, avec $|V_p|$ le nombre de vols dans l'itinéraire du groupe p (dans la Figure 3.6 $|V_p|$=2). La contrainte (1-r) est similaire à celle du transit des avions (1-c). Si l'itinéraire est composé de plusieurs vols, il faut respecter au moins 30 minutes de transit entre la date d'arrivée du groupe p ($r_{i,a} + \mathrm{T}^{+}_{i,a}$) et la date du prochain départ ($r_{j,b} + \mathrm{T}^{-}_{j,b}$).

L'ensemble des contraintes précédentes (1-a) à (1-r) n'est pas suffisant pour modéliser le problème car d'autres contraintes associées à la période d'intervention entrent en jeu. En effet avant et au-delà de cette période de temps, nous ne pouvons pas intervenir sur le plan de vols. Pour gérer ces problèmes, nous proposons d'ajouter d'autres contraintes.

3.3.4 Contraintes supplémentaires et bornes des variables

Avant la Période de Recouvrement : les vols arrivés ou déjà débutés en début de période de recouvrement (dont l'heure de départ indiquée dans le programme de vols initial est strictement antérieure au début de la période de recouvrement) ne peuvent pas être altérés. Plus formellement, si le départ prévu d'un vol est avant le début de la période, alors la variable de retard associée à ce vol doit être fixée à 0 et la variable associée au vol doit être fixée à 1, c'est-à-dire

$$(C-avant-recouvrement)\begin{cases} r_{i,a}=0 & \forall a\in A,\forall i\in VE_a : T^-_{i,a}<T^- \quad (1\text{-}s) \\ u_{i,a}=1 & \forall a\in A,\forall i\in VE_a : T^-_{i,a}<T^- \quad (1\text{-}t) \end{cases}$$

Annulation de vols : si un vol est annulé par les perturbations alors la variable associée à ce vol doit être fixée à 0, c'est-à-dire

$$(C-Annulation): u_{i,a}=0 \quad \forall a\in A,\forall i\in VE_a\cap VA \qquad (1\text{-}u)$$

avec *VA* l'ensemble des vols annulés par les perturbations avant la période de recouvrement.

Période d'indisponibilité d'un avion : si un avion est contraint à une période d'indisponibilité (hors maintenance) dans un intervalle de temps, alors toutes les variables d'atterrissage et de décollage associées à tous les vols de cet avion et aux tranches horaires incluses dans la période d'indisponibilité doivent êtres fixées à 0, c'est-à-dire

$$(C-Indisponibilité)\begin{cases} x^-_{i,a,k,h}=0 & \forall i,a,k,h:\left[T^-_{k^-(i,a),h},T^+_{k^-(i,a),h}\right]\subset\left[I^-_a,I^+_a\right] \quad (1\text{-}v) \\ x^+_{i,a,k,h}=0 & \forall i,a,k,h:\left[T^-_{k^+(i,a),h},T^+_{k^+(i,a),h}\right]\subset\left[I^-_a,I^+_a\right] \quad (1\text{-}w) \end{cases}$$

Distance maximale : si la distance entre deux aéroports dépasse la distance maximum qui peut être parcourue par un avion a (notée D_a^{max}), alors la variable associée à ce vol éventuel doit être fixée à 0, c'est-à-dire

$$(C-Dist-Max):\ D_{k^-(i,a),k^+(i,a)}w_{i,a}\le D_a^{max} \quad \forall a\in A,\forall i\in VE_a \qquad (1\text{-}x)$$

Notons que les contraintes considérées dans cette partie et les bornes des variables sont conservées durant tout le processus de résolution de notre modèle.

3.3.5 Fonction objectif

Notre objectif dans cette phase d'initialisation est de maximiser le nombre de passagers arrivant à destination tout en minimisant le retard total. Plus précisément, la fonction objectif considérée peut être formulée comme suit :

$$\text{Maximiser } OBJ=\alpha\sum_{p\in P}G_p y_p-\beta\sum_{i\in VE_a,a\in A}r_{i,a} \qquad (1\text{-y})$$

où α et β sont les coefficients de pondération et de normalisation des deux coûts dans la fonction objectif. Nous maximisons le nombre de passagers transportés en prenant en considération le retard de chaque vol. Notons qu'en pratique le retard a un poids très faible par rapport au transport des passagers, en raison du coût associé à l'annulation d'un groupe de passagers. L'ajout du retard dans l'objectif avec un faible poids nous permet cependant d'éviter des retards de vols non justifiés qui peuvent se produire lorsqu'un vol peut assurer

l'acheminement des passagers dans un intervalle, par exemple [11:30,11:50]. Dans ce cas nous préférons fixer l'atterrissage à 11:30 pour minimiser les retards.

3.3.6 Méthodes de résolution

Si nous ne tenions pas compte de contraintes sur le temps de calcul, et si l'objectif était uniquement d'obtenir une solution réalisable du problème, nous pourrions résoudre le programme linéaire en nombres entiers mixtes MIP suivant :

MIP Maximiser $OBJ = \alpha \sum_{p \in P} G_p y_p - \beta \sum_{i \in VE_a, a \in A} r_{i,a}$ (1-y)

$u_{i-,a} + v_{\hat{i},a} \geq u_{i,a} + v_{i,a} + w_{i,a}$ $a \in A, i \neq i_a^m$ (1-a)

$u_{i_a^m,a} + v_{\hat{i}_a^m,a} + \sum_{j=1}^{i_a^m} w_{j,a} = 1$ $a \in A^{ma\text{int}}$ (1-b)

$r_{j,a} + \mathrm{T}_{j,a}^{+} + E_a - L(1-u_{j,a}) \leq r_{i,a} + \mathrm{T}_{i,a}^{-} + L(1-u_{i,a})$ $a \in A, j \prec i$ (1-c)

$r_{i-,a} + \mathrm{T}_{i-,a}^{+} + E_a - \mathrm{L}(1-u_{i-,a}) \leq t_{i,a} + \mathrm{L}(1-w_{i,a})$ $a \in A^{ma\text{int}}, i \prec i_a^m$ (1-d)

$r_{i,a} \leq M_a - \mathrm{T}_{i,a}^{+}$ $a \in A^{ma\text{int}}, i \leq i_a^m$ (1-e)

$t_{i,a} + D_{k^-(i,a),k^+(i_a^m,a)} \leq \mathrm{M}_a$ $a \in A^{ma\text{int}}, i \leq i_a^m$ (1-f)

$$\sum_{h=1}^{H_{k^-(i,a)}} \mathrm{T}_{k^-(i,a),h}^{-} \times x_{i,a,k^-(i,a),h}^{-} \leq r_{i,a} + \mathrm{T}_{i,a}^{-} \leq \sum_{h=1}^{H_{k^-(i,a)}} \mathrm{T}_{k^-(i,a),h}^{+} \times x_{i,a,k^-(i,a),h}^{-}$$ $a \in A, i \in VE_a$ (1-g)

$$u_{i,a} = \sum_{h=1}^{H_{k^-(i,a)}} x_{i,a,k^-(i,a),h}^{-}$$ $a \in A, i \in VE_a$ (1-h)

$$\sum_{h=1}^{H_{k^+(i,a)}} \mathrm{T}_{k^+(i,a),h}^{-} \times x_{i,a,k^+(i,a),h}^{+} \leq r_{i,a} + \mathrm{T}_{i,a}^{+} \leq \sum_{h=1}^{H_{k^+(i,a)}} \mathrm{T}_{k^+(i,a),h}^{+} \times x_{i,a,k^+(i,a),h}^{+}$$ $a \in A, i \in VE_a$ (1-i)

$$u_{i,a} = \sum_{h=1}^{H_{k^+(i,a)}} x_{i,a,k^+(i,a),h}^{+}$$ $a \in A, i \in VE_a$ (1-j)

$$\sum_{h=1}^{H_{k^-(i,a)}} \mathrm{T}_{k^-(i,a),h}^{-} \times x_{i,a,k^-(i,a),h}^{-} \leq t_{i,a} \leq \sum_{h=1}^{H_{k^-(i,a)}} \mathrm{T}_{k^-(i,a),h}^{+} \times x_{i,a,k^-(i,a),h}^{-}$$ $a \in A, i \in VC_a$ (1-k)

$$w_{i,a} = \sum_{h=1}^{H_{k^-(i,a)}} x_{i,a,k^-(i,a),h}^{-}$$ $a \in A, i \in VC_a$ (1-l)

$$\sum_{h=1}^{H_{k^+(i,a)}} \mathrm{T}_{k^+(i,a),h}^{-} \times x_{i,a,k^+(i,a),h}^{+} \leq t_{i,a} + D_{k^-(i,a),k^+(i_a^m,a)} \leq \sum_{h=1}^{H_{k^+(i,a)}} \mathrm{T}_{k^+(i,a),h}^{+} \times x_{i,a,k^+(i,a),h}^{+}$$ $a \in A, i \in VC_a$ (1-m)

$$w_{i,a} = \sum_{h=1}^{H_{k^+(i,a)}} x_{i,a,k^+(i,a),h}^{+}$$ $a \in A, i \in VC_a$ (1-n)

$$\sum_{i,a:k^-(i,a)=k} x_{i,a,k,h}^{-} \leq \mathrm{C}_{k,h}^{-}$$ $k \in K, h \in H_k$ (1-o)

$$\sum_{i,a:k^+(i,a)=k} x^+_{i,a,k,h} \le C^+_{k,h} \qquad k \in K, h \in H_k \quad \text{(1-p)}$$

$$|V_p| y_p \le \sum_{(i,a)\in V_p} u_{i,a} \qquad p \in P \quad \text{(1-q)}$$

$$r_{i,a} + T^+_{i,a} + 30 \le r_{j,b} + T^-_{j,b} + L(1-y_p) \qquad p \in P, \{(i,a),(j,b)\} \subseteq V_p, (i,a) \tilde{\prec} (j,b) \quad \text{(1-r)}$$

$$r_{i,a} = 0 \qquad a \in A, i \in VE_a : T^-_{i,a} < T^- \quad \text{(1-s)}$$

$$u_{i,a} = 1 \qquad a \in A, i \in VE_a : T^-_{i,a} < T^- \quad \text{(1-t)}$$

$$u_{i,a} = 0 \qquad a \in A, i \in VE_a \cap VA \quad \text{(1-u)}$$

$$x^-_{i,a,k,h} = 0 \qquad i,a,k,h : \left[T^-_{k^-(i,a),h}, T^+_{k^-(i,a),h}\right] \subset \left[I^-_a, I^+_a\right] \quad \text{(1-v)}$$

$$x^+_{i,a,k,h} = 0 \qquad i,a,k,h : \left[T^-_{k^+(i,a),h}, T^+_{k^+(i,a),h}\right] \subset \left[I^-_a, I^+_a\right] \quad \text{(1-w)}$$

$$D_{k^-(i,a),k^+(i,a)} w_{i,a} \le D^{\max}_a \qquad a \in A, i \in VE_a \quad \text{(1-x)}$$

$$u_{i,a}, v_{j,a}, w_{l,a} \in \{0,1\} \qquad a \in A, i \in VE_a, j \in VF_a, l \in VC_a$$

$$r_{i,a}, t_{l,a} \ge 0 \qquad a \in A, i \in VE_a, l \in VC_a$$

$$x^-_{i,a,k^-(i,a),h}, x^+_{i,a,k^+(i,a),h} \in \{0,1\} \qquad a \in A, i \in VE_a \cup VC_a, h \in H_k$$

$$y_p \in \{0,1\} \qquad p \in P$$

Malheureusement, d'après nos expériences numériques, la résolution exacte par des techniques de séparations et évaluations du problème MIP ne garantit pas toujours de trouver une solution réalisable dans un délai raisonnable. Ceci est dû en particulier aux contraintes dures du problème : les contraintes de maintenance (1-b), les contraintes de capacités aéroportuaires (1-o) et (1-p), et au fait que les routes initiales des avions ne peuvent pas être modifiées. Nous proposons donc des relaxations partielles des contraintes de maintenance et des capacités aéroportuaires de manière à contourner ce problème.

3.3.6.1 Relaxation des contraintes de maintenance

La contrainte de maintenance d'un avion est nécessaire pour la sécurité des passagers, ce qui justifie le fait qu'elle ne puisse être violée. L'autorisation d'un saut direct d'un avion vers l'aéroport de maintenance dans la tranche horaire souhaitée n'est pas suffisante pour assurer la réalisabilité vis-à-vis de cette contrainte dure. Pour assurer au mieux la réalisabilité des contraintes de maintenance, nous proposons dans un premier temps de maximiser le nombre d'avions dont la maintenance est respectée. Pour cela nous introduisons les variables binaires suivantes :

z_a : variable binaire égale à 1 si l'avion a réalise sa maintenance, 0 sinon.

La contrainte de maintenance (1-b) est alors remplacée par

$$u_{i_a^m,a} + v_{i_a^m,a} + \sum_{j=1}^{i_a^m} w_{j,a} = z_a \qquad \forall a \in A^{maint} \qquad \text{(1-b')}.$$

Le modèle proposé MIP1 maximise alors le nombre d'avions respectant les contraintes de maintenance :

(MIP1) Maximiser $OBJ1 = \sum_{a \in A^{maint}} z_a$ sous les contraintes (1-a), (1-b'),..., (1- x).

Pour résoudre ce modèle, nous annulons tous les vols du problème, exceptés ceux qui sont fixés (programmés avant la période d'intervention ou programmés pour un avion avant sa maintenance) et nous ne relaxons aucune contrainte. L'annulation de ces vols libère des possibilités pour assurer la contrainte de maintenance pour chaque avion. La résolution de ce problème n'est pas très difficile car une grande partie du problème est annulée. Elle peut se faire en appliquant directement les procédures de résolution exacte à l'aide du solveur CPLEX.

Une solution optimale z^* du problème (MIP1) dont toutes les variables z_a^* ($a \in A^{maint}$) sont égales à 1 fournit une solution réalisable du problème (MIP). Au contraire si au moins une variable z_a^* est égale à 0, alors il existe au moins un avion a qui ne peut pas assurer sa maintenance. Si ce cas se produit, nous appliquons une procédure de réparation basée sur la programmation dynamique (voir Section 3.3.6.3). Dans le cas contraire nous fixons les variables z obtenues en résolvant (MIP1) et nous revenons ensuite au modèle d'origine (MIP).

3.3.6.2 Résolution itérative

La résolution exacte de (MIP) exige un temps d'exécution très important car le problème est NP-Difficile, et sa taille dépasse 400 000 variables et 100 000 contraintes d'après nos expérimentations.

Nous appliquons donc des techniques de pénalisations sur un sous-ensemble de contraintes dans l'objectif de trouver rapidement une bonne solution. Nous dressons tout d'abord un diagnostic afin d'identifier le bloc de contraintes qui entraîne la difficulté majeure du problème.

Nous remarquons que les contraintes (1-a), (1-b'), (1-c) et (1-d) sont des contraintes de flot et de temps. Dès lors, si nous supprimons toutes les autres contraintes et gardons uniquement ces contraintes, le problème réduit devient plus facile à traiter. De plus, les contraintes (1-e) et (1-f) représentent des bornes supérieures sur les variables continues et ne compliquent pas excessivement la résolution du problème. Les contraintes (1-q) et (1-r)

représentent le problème de flot des passagers qui n'a pas un grand impact sur la difficulté du problème global. Nous remarquons également que si nous considérons les contraintes (1-g),..., (1-n) sans les contraintes de capacité (1-o) et (1-p), alors l'ensemble des variables binaires (qui sont les plus nombreuses dans le problème) ne sont plus que des variables indicatives. Elles signalent alors uniquement les tranches d'atterrissage et de décollage associées aux vols modifiés dans une solution optimale. L'ajout des contraintes de capacité (1-o) et (1-p) rend les variables x^+ et x^- non plus indicatives mais décisionnelles, ce qui a un grand impact dans le modèle et a une forte influence sur les contraintes de flot sur les vols. Nous pouvons donc en conclure que les contraintes de capacité aéroportuaires (1-o) et (1-p) représentent les contraintes clés du problème.

Lors de la résolution de ce programme mathématique, nous considérons donc une relaxation partielle des contraintes aéroportuaires, en ajoutant des variables de tolérance de violation :

$s^-_{k,h}$: variable entière qui représente le nombre d'atterrissages autorisés au-delà de la capacité d'atterrissage de la tranche h de l'aéroport k.

$s^+_{k,h}$: variable entière qui représente le nombre de décollages autorisés au-delà de la capacité de décollage de la tranche h de l'aéroport k.

Les contraintes de capacité (1-o) et (1-p) après relaxation sont alors écrites comme suit :

$$\sum_{i,a:k^-(i,a)=k} x^-_{i,a,k,h} \le \mathrm{C}^-_{k,\mathrm{h}} + s^-_{k,h} \qquad k \in K, h \in H_k \qquad \text{(1-o')}$$

$$\sum_{i,a:k^+(i,a)=k} x^+_{i,a,k,h} \le \mathrm{C}^+_{k,\mathrm{h}} + s^+_{k,h} \qquad k \in K, h \in H_k \qquad \text{(1-p')}$$

Nous obtenons alors le modèle mathématique MIP2(Q) suivant :

MIP2(Q) Maximiser $OBJ2 = \alpha \sum_{p \in P} G_p y_p - \beta \sum_{i \in VE_a, a \in A} r_{i,a} - \sum_{k \in K, h \in H_k} Q s^-_{k,h} - \sum_{k \in K, h \in H_k} Q s^+_{k,h}$ (1-y)

$$u_{i-,a} + v_{\hat{i},a} \ge u_{i,a} + v_{i,a} + w_{i,a} \qquad a \in A, i \ne i^m_a \qquad \text{(1-a)}$$

$$u_{i^m_a,a} + v_{\hat{i}^m_a,a} + \sum_{j=1}^{i^m_a} w_{j,a} = z^*_a \qquad a \in A^{ma\,\mathrm{int}} \qquad \text{(1-b')}$$

$$r_{j,a} + \mathrm{T}^+_{j,a} + E_a - L(1-u_{j,a}) \le r_{i,a} + \mathrm{T}^-_{i,a} + L(1-u_{i,a}) \qquad a \in A, j \prec i \qquad \text{(1-c)}$$

$$r_{i-,a} + \mathrm{T}^+_{i-,a} + E_a - \mathrm{L}(1-u_{i-,a}) \le t_{i,a} + \mathrm{L}(1-w_{i,a}) \qquad a \in A^{ma\,\mathrm{int}}, i \prec i^m_a \qquad \text{(1-d)}$$

$$r_{i,a} \le M_a - \mathrm{T}^+_{i,a} \qquad a \in A^{ma\,\mathrm{int}}, i \le i^m_a \qquad \text{(1-e)}$$

$$t_{i,a} + D_{k^-(i,a),k^+(i^m_a,a)} \le \mathrm{M}_a \qquad a \in A^{ma\,\mathrm{int}}, i \le i^m_a \qquad \text{(1-f)}$$

$$\sum_{h=1}^{H_{k^-(i,a)}} \mathrm{T}^-_{k^-(i,a),h} \times x^-_{i,a,k^-(i,a),h} \le r_{i,a} + \mathrm{T}^-_{i,a} \le \sum_{h=1}^{H_{k^-(i,a)}} \mathrm{T}^+_{k^-(i,a),h} \times x^-_{i,a,k^-(i,a),h} \qquad a \in A, i \in VE_a \qquad \text{(1-g)}$$

$$u_{i,a} = \sum_{h=1}^{H_{k^-(i,a)}} x^-_{i,a,k^-(i,a),h} \qquad a \in A, i \in VE_a \qquad \text{(1-h)}$$

$$\sum_{h=1}^{H_{k^+(i,a)}} \mathrm{T}^-_{k^+(i,a),h} \times x^+_{i,a,k^+(i,a),h} \le r_{i,a} + \mathrm{T}^+_{i,a} \le \sum_{h=1}^{H_{k^+(i,a)}} \mathrm{T}^+_{k^+(i,a),h} \times x^+_{i,a,k^+(i,a),h} \qquad a \in A, i \in VE_a \qquad \text{(1-i)}$$

$$u_{i,a} = \sum_{h=1}^{H_{k^+(i,a)}} x^+_{i,a,k^+(i,a),h} \qquad a \in A, i \in VE_a \qquad \text{(1-j)}$$

$$\sum_{h=1}^{H_{k^-(i,a)}} \mathrm{T}^-_{k^-(i,a),h} \times x^-_{i,a,k^-(i,a),h} \le t_{i,a} \le \sum_{h=1}^{H_{k^-(i,a)}} \mathrm{T}^+_{k^-(i,a),h} \times x^-_{i,a,k^-(i,a),h} \qquad a \in A, i \in VC_a \qquad \text{(1-k)}$$

$$w_{i,a} = \sum_{h=1}^{H_{k^-(i,a)}} x^-_{i,a,k^-(i,a),h} \qquad a \in A, i \in VC_a \qquad \text{(1-l)}$$

$$\sum_{h=1}^{H_{k^+(i,a)}} \mathrm{T}^-_{k^+(i,a),h} \times x^+_{i,a,k^+(i,a),h} \le t_{i,a} + D_{k^-(i,a),k^+(i^m_a,a)} \le \sum_{h=1}^{H_{k^+(i,a)}} \mathrm{T}^+_{k^+(i,a),h} \times x^+_{i,a,k^+(i,a),h} \qquad a \in A, i \in VC_a \qquad \text{(1-m)}$$

$$w_{i,a} = \sum_{h=1}^{H_{k^+(i,a)}} x^+_{i,a,k^+(i,a),h} \qquad a \in A, i \in VC_a \qquad \text{(1-n)}$$

$$\sum_{i,a:k^-(i,a)=k} x^-_{i,a,k,h} \le \mathrm{C}^-_{k,h} + s^-_{k,h} \qquad k \in K, h \in H_k \qquad \text{(1-o')}$$

$$\sum_{i,a:k^+(i,a)=k} x^+_{i,a,k,h} \le \mathrm{C}^+_{k,h} + s^+_{k,h} \qquad k \in K, h \in H_k \qquad \text{(1-p)}$$

$$|V_p| y_p \le \sum_{(i,a) \in V_p} u_{i,a} \qquad p \in P \qquad \text{(1-q)}$$

$$r_{i,a} + \mathrm{T}^+_{i,a} + 30 \le r_{j,b} + \mathrm{T}^-_{j,b} + \mathrm{L}(1 - y_p) \qquad p \in P, \{(i,a),(j,b)\} \subseteq V_p, (i,a) \tilde{\prec} (j,b) \qquad \text{(1-r)}$$

$$r_{i,a} = 0 \qquad a \in A, i \in VE_a : T^-_{i,a} < T^- \qquad \text{(1-s)}$$

$$u_{i,a} = 1 \qquad a \in A, i \in VE_a : T^-_{i,a} < T^- \qquad \text{(1-t)}$$

$$u_{i,a} = 0 \qquad a \in A, i \in VE_a \cap VA \qquad \text{(1-u)}$$

$$x^-_{i,a,k,h} = 0 \qquad i,a,k,h : \left[\mathrm{T}^-_{k^-(i,a),h}, \mathrm{T}^+_{k^-(i,a),h}\right] \subset \left[\mathrm{I}^-_a, \mathrm{I}^+_a\right] \qquad \text{(1-v)}$$

$$x^+_{i,a,k,h} = 0 \qquad i,a,k,h : \left[\mathrm{T}^-_{k^+(i,a),h}, \mathrm{T}^+_{k^+(i,a),h}\right] \subset \left[\mathrm{I}^-_a, \mathrm{I}^+_a\right] \qquad \text{(1-w)}$$

$$\mathrm{D}_{k^-(i,a),k^+(i,a)} w_{i,a} \le \mathrm{D}^{\max}_a \qquad a \in A, i \in VE_a \qquad \text{(1-x)}$$

$$u_{i,a}, v_{j,a}, w_{l,a} \in \{0,1\} \qquad a \in A, i \in VE_a, j \in VF_a, l \in VC_a$$

$$r_{i,a}, t_{l,a} \ge 0 \qquad a \in A, i \in VE_a, l \in VC_a$$

$$x^-_{i,a,k^-(i,a),h}, x^+_{i,a,k^+(i,a),h} \in \{0,1\} \qquad a \in A, i \in VE_a \cup VC_a, h \in H_k$$

$$y_p \in \{0,1\} \qquad p \in P$$

$$s^-_{k,h}, s^+_{k,h} \ge 0 \qquad k \in K, h \in H_k$$

où Q est une constante importante qui pénalise la violation des contraintes de capacité des aéroports. Si $Q = 0$ (resp. Q tend vers ∞), alors les contraintes sont relâchées (resp. ne sont pas relâchées). La résolution de ce problème est basée sur une approche itérative dans laquelle nous augmentons la valeur de Q à chaque itération puis nous résolvons MIP2(Q). Nous arrêtons le processus si Q tend vers ∞ ou, en pratique, si nous dépassons un certain temps. Cette méthode de résolution permet au solveur de trouver à chaque itération une solution de départ qui permet d'accélérer la résolution à l'itération suivante ou au moins d'améliorer rapidement la solution courante.

Les expériences numériques préliminaires ont montré que cette démarche nous permet d'obtenir des solutions de bonne qualité en 2 minutes d'exécution maximum alors que la résolution brute et directe du problème (MIP) prend pour certaines instances plus de 6 heures d'exécution pour obtenir une solution réalisable (qui n'est pas forcément de bonne qualité).

Après avoir resserré au maximum les contraintes de capacité des aéroports, nous passons à une dernière étape dans laquelle nous fixons toutes les variables du problème, exceptées les variables concernant l'annulation et la création des vols (x^+, x^-, u, v, w). Nous éliminons également les variables s dans le but de réintégrer complètement les contraintes de capacité des aéroports. L'objectif de cette dernière étape est d'assurer la réalisabilité de la solution sans dégrader sa qualité. Le programme mathématique suivant est obtenu après fixation.

MIP3 Maximiser $OBJ3 = \sum_{p \in P} G_p y_p$

$$u_{i-,a} + v_{\hat{i},a} \geq u_{i,a} + v_{i,a} + w_{i,a} \qquad a \in A, i \neq i_a^m \qquad \text{(1-a)}$$

$$u_{i_a^m,a} + v_{\hat{i}_a^m,a} + \sum_{j=1}^{i_a^m} w_{j,a} = z_a^* \qquad a \in A^{maint} \qquad \text{(1-b')}$$

$$u_{i,a} = x^-_{i,a,k^-(i,a),h'} \qquad a \in A, i \in VE_a \qquad \text{(1-h)}$$

$$u_{i,a} = x^+_{i,a,k^+(i,a),h''} \qquad a \in A, i \in VE_a \qquad \text{(1-j)}$$

$$w_{i,a} = x^-_{i,a,k^-(i,a),\dot{h}} \qquad a \in A, i \in VC_a \qquad \text{(1-l)}$$

$$w_{i,a} = x^+_{i,a,k^+(i,a),\ddot{h}} \qquad a \in A, i \in VC_a \qquad \text{(1-n)}$$

$$\sum_{i,a:k^-(i,a)=k} x^-_{i,a,k,h} \leq C^-_{k,h} \qquad k \in K, h \in H_k \qquad \text{(1-o)}$$

$$\sum_{i,a:k^+(i,a)=k} x^+_{i,a,k,h} \leq C^+_{k,h} \qquad k \in K, h \in H_k \qquad \text{(1-p)}$$

$$|V_p| y_p \leq \sum_{(i,a) \in V_p} u_{i,a} \qquad p \in P \qquad \text{(1-q)}$$

$$r_{i,a} = 0 \qquad a \in A, i \in VE_a : T^-_{i,a} < T^- \qquad \text{(1-s)}$$

$$u_{i,a} = 1 \qquad a \in A, i \in VE_a : T^-_{i,a} < T^- \qquad \text{(1-t)}$$

$$u_{i,a} = 0 \qquad a \in A, i \in VE_a \cap VA \qquad \text{(1-u)}$$

$$x^-_{i,a,k,h} = 0 \qquad i,a,k,h : \left[T^-_{k^-(i,a),h}, T^+_{k^-(i,a),h}\right] \subset \left[I^-_a, I^+_a\right] \qquad \text{(1-v)}$$

$$x^+_{i,a,k,h} = 0 \qquad i,a,k,h : \left[T^-_{k^+(i,a),h}, T^+_{k^+(i,a),h}\right] \subset \left[I^-_a, I^+_a\right] \qquad \text{(1-w)}$$

$$D_{k^-(i,a),k^+(i,a)} w_{i,a} \leq D^{\max}_a \qquad a \in A, i \in VE_a \qquad \text{(1-x)}$$

$$u_{i,a}, v_{j,a}, w_{l,a} \in \{0,1\} \qquad a \in A, i \in VE_a, j \in VF_a, l \in VC_a$$

$$x^-_{i,a,k^-(i,a),h}, x^+_{i,a,k^+(i,a),h} \in \{0,1\} \qquad a \in A, i \in VE_a \cup VC_a$$

$$y_p \in \{0,1\} \qquad p \in P$$

La résolution du modèle MIP3 se fait à l'aide du solveur CPLEX avec un temps d'exécution limité à 2 minutes.

L'application de la première phase ne permet malheureusement pas toujours d'obtenir une solution réalisable du problème initial (pour 2 instances sur les 30 traitées). Dans ce cas nous proposons l'utilisation d'une procédure de réparation.

3.3.6.3 Procédure de réparation

Seules les contraintes de maintenance peuvent mener à une solution irréalisable dans laquelle au moins un avion est incapable d'atteindre à temps l'aéroport assigné pour sa maintenance sans modification importante du plan de vols initial. Si cela arrive, nous appliquons un algorithme basé sur la programmation dynamique pour trouver un chemin approprié pour atteindre à temps l'aéroport de maintenance pour chaque avion. L'objectif est d'assurer la maintenance de cet avion sans trop perturber son parcours initial.

Dans certains cas, il n'est pas possible de créer un vol direct à l'aéroport de maintenance à cause des capacités des aéroports (sinon le modèle de la première phase aurait su gérer le problème), ou si le vol est plus long que la longueur maximum autorisée pour l'avion. En conséquence, un nouveau chemin peut être nécessaire, ce qui modifie profondément la structure de la solution initiale.

L'algorithme emploie un graphe spatio-temporel dans lequel les sommets sont des couples (*aéroport*, *temps*) de situations et chaque arc représente la possibilité d'atteindre une situation à partir d'une autre. Dans ce graphe, nous calculons le chemin de meilleur coût entre la situation initiale et la situation finale associée au sommet (*aéroport de maintenance*, *date de maintenance*). D'un point de vue pratique, il n'est pas envisageable de réaliser un couplage (*aéroport*, *temps*) associé à chaque minute. Nous utilisons en conséquence des périodes (intervalles) d'une heure. L'approche utilisée est une heuristique car elle ne modifie pas les

vols des avions avec maintenance et ne considère que des périodes de temps. La qualité de la solution dépend ainsi de l'ordre de traitement des avions et de la précision de cette période.

3.3.6.3.1 Création du graphe

Le graphe est créé de façon dynamique. Nous créons seulement les sommets qui peuvent être atteints depuis le sommet initial en suivant un chemin valide (il peut donc y avoir un grand nombre de sommets). Les arcs représentent la possibilité d'aller d'un sommet (k, t) à un autre sommet (k', t'). Ils sont ajoutés dynamiquement et simultanément lors de la création des sommets. La possibilité d'atteindre un sommet est définie par la relation de récurrence suivante :

1. le sommet initial (début) est atteint.
2. un sommet (k, t) est atteint s'il est possible de créer un vol d'un autre sommet (k', t - $D_{k',k}$) à (k, t), où $D_{k',k}$ est la distance (en temps) entre l'aéroport k' et l'aéroport k.
3. un sommet (k, t) est atteint si (k, t-1) est atteint.

Lors de la construction, nous relaxons les contraintes de capacité (dans la relation 2) en autorisant l'annulation des vols d'avions sans maintenance. Cette modification est cependant prise en compte dans le coût de l'arc.

Soient |K| le nombre d'aéroports et |T| le nombre d'intervalles de temps considéré. Si le graphe est représenté par sa matrice d'adjacence, l'algorithme de construction nécessite O(|K| + |T|) en temps et en espace mémoire.

Nous associons deux valeurs à chaque arc : un temps (le temps de voyage si l'arc correspond à un vol, 0 s'il correspond à un avion non mobilisé) et un coût. Ce coût dépend du nombre de vols à annuler pour pouvoir créer un nouveau vol. Nous avons testé une autre version dans laquelle nous cherchions à minimiser le nombre de passagers avec un itinéraire annulé. La première version a fourni les meilleurs résultats en général. Cela peut s'expliquer par le fait que dans ce cas nous ne modifions pas trop la solution initiale.

3.3.6.3.2 Calcul du meilleur chemin

Etant donné que le graphe est spatio-temporel et que chaque arc relie un sommet (k, t) à un sommet (k', t') tels que $t' > t$, il ne peut contenir aucun circuit. L'algorithme de Bellman (1957) peut donc être appliqué avec les relations de récurrences suivantes :

- v(*aéroport de départ*, *date de départ*) = 0

- $v(k, t) = \underset{(k',t')\in N^-(k,t)}{Min} (v(k', t') + \delta)$

où δ est le coût lié à l'arc reliant (k, t) à (k', t') et $N^-(k, t)$ l'ensemble des prédécesseurs du sommet (k, t).

La méthode est donc linéaire par rapport au nombre d'aéroports et elle dépend de la longueur des intervalles de temps considéré.

Dans la première version étudiée (orientée coût), la valeur de δ correspond au nombre de vols qui doivent être annulés pour créer le vol qui assure la maintenance de l'avion ($\delta = 0, 1, 2,\ldots$). Dans la deuxième version (orientée temps), la valeur de δ est la durée du vol pour relier ces sommets. Les dates d'arrivée sont des informations cruciales, puisque nous employons des fenêtres de temps d'une heure. Cela entraîne plusieurs manières d'atteindre un sommet avec différentes dates d'arrivée, ce qui peut rendre certains vols réalisables ou non.

Quand l'algorithme de Bellman est appliqué, un seul coût peut être pris en considération. Dans notre cas chaque arc est marqué par deux valeurs (temps, coût). Nous avons aussi testé deux versions : la première considère seulement la valeur du chemin, la seconde considère le temps et ensuite la valeur du chemin selon un ordre lexicographique. La première version donne en général de meilleurs résultats en termes de coût sans garantie de réparer la solution. La deuxième version assure la validité de la solution, mais le nombre d'annulations de vols est en général plus grand. Nous avons opté pour cette deuxième version, puisque notre objectif est avant tout de réparer la solution.

S'il est possible d'atteindre le sommet final (aéroport de maintenance, date de début de maintenance), il est possible de trouver un chemin réalisable pour l'avion. Pour que le chemin soit généré, quelques vols doivent éventuellement être annulés. Si l'algorithme a le choix entre plusieurs vols, il annule en priorité ceux qui comportent le plus petit nombre de passagers.

Avant de décrire la seconde phase de notre approche, nous résumons dans l'Algorithme 3.1 les grandes étapes de la résolution du problème dans cette première partie. Nous commençons par la résolution du problème MIP1 afin de trouver une première solution Sol^0. Cette solution représente la meilleure situation possible pour laquelle un maximum d'avions assure leur maintenance. Dans le but de garder cette situation, nous fixons les variables associées aux contraintes de maintenances, $z = z^*$ (z^* représente le vecteur des variable z dans la solution Sol^0). Nous résolvons ensuite le problème MIP2(Q). Nous considérons à chaque itération k la solution précédente Sol^{k-1} comme une solution d'entrée pour accélérer le processus. À chaque itération nous augmentons arbitrairement la valeur de Q (à l'aide d'un

pas noté q = 100 ou 200). Nous avons considéré la constante importante *Q_Max* comme une borne supérieure pour la valeur de Q. La valeur de *Q_Max* est fixée en pratique à la plus grande capacité des avions. Rappelons que nous avons pénalisé la fonction objectif du problème MIP2(Q) de la façon suivante $-\sum_{k,h} Qs^{-}_{k,h} - \sum_{k,h} Qs^{+}_{k,h}$, où $Qs^{-}_{k,h}$ (resp. $Qs^{+}_{k,h}$) est le nombre d'atterrissages (resp. décollages) supplémentaires dans la tranche h de l'aéroport k multiplié par Q. Si $Q = Q_Max$, alors la quantité $Q_Max \times s^{-}_{k,h}$ (resp. $Q_Max \times s^{+}_{k,h}$) est une estimation du nombre de passagers qui ne vont pas assurer leur destination finale. En sortant de cette boucle itérative nous fixons les variables associées aux dates des vols (r, t) à (r^*, t^*) dans la dernière solution Sol^k. Cette fixation permet de relaxer les contraintes (1-c),..., (1-j) et (1-n), ce qui nous ramène à résoudre un problème de flot MIP3 plus facile. La résolution de MIP3 donne une solution Sol^* qui assure la réalisabilité de toutes les contraintes du problème sauf éventuellement celles de la maintenance. Dans ce cas, nous appliquons l'heuristique de réparation.

Algorithme 3.1 : La première phase

Sol^0 = ***résoudre*** (MIP1) ;
Fixer les variables z à z^* ;
$Q = 0$; $k = 0$;

tant que $Q < Q_Max$ **faire**
    Sol^k = ***résoudre*** (MIP 2(Q), Sol^{k-1}) ;
    $k = k + 1$;
    $Q = Q + q$;
fin tant que

Fixer les variables (r,t) à (r^*,t^*) ;
Sol^* = ***résoudre*** (MIP 3) ;
si Sol^* n'est pas réalisable **alors** Appliquer la procédure de ***réparation*** ; **fin si**

La Figure 3.7 illustre une partie du plan de vols associée à la solution réalisable obtenue à la fin de cette première phase.

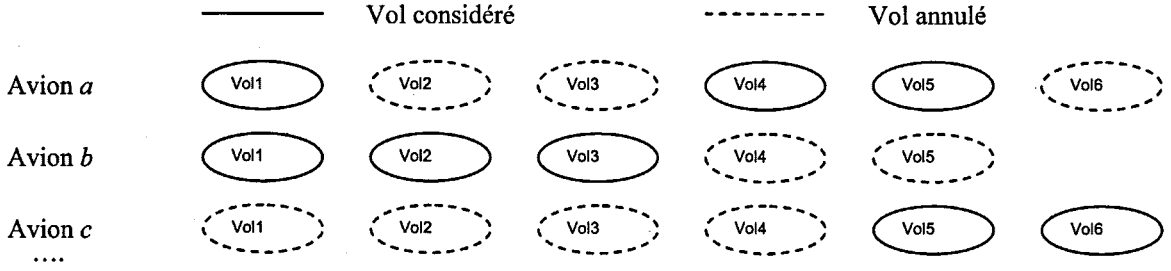


Figure 3.7 : Solution réalisable initiale

Cette figure montre que les avions peuvent être libres dans plusieurs intervalles en raison de l'annulation de vols. Ces intervalles, dits de *disponibilité*, seront notre zone de traitement privilégiée de la deuxième phase de l'algorithme où nous allons créer des vols pour assurer le transport de passagers n'ayant pas atteint leur destination.

3.4 Stratégie d'oscillation

La stratégie d'oscillation est étroitement liée aux origines de la recherche Tabou (Glover (1989), Glover (1990)) et fournit un moyen d'obtenir une interaction efficace entre l'intensification et la diversification sur le moyen terme jusqu'au long terme. Le processus consiste à approcher et à traverser de manière répétée un niveau critique à partir de différentes directions, ce qui crée un comportement oscillatoire. La stratégie d'oscillation consiste donc à définir un rythme alterné plus ou moins régulier pour traverser les niveaux critiques dans différentes directions. Toutes ces solutions critiques constituent la *zone prometteuse*. Notons que même si en général une stratégie d'oscillation autorise la visite de solutions non réalisables, nous nous limitons à l'espace des solutions réalisables, en particulier en raison de la difficulté de générer une solution réalisable du GPDA. La stratégie d'oscillation a été appliquée pour résoudre efficacement quelques problèmes classiques de la famille du sac à dos (voir Glover et Kochenberger (1996), Hanafi et Fréville (1998), Li et Curry (2005) par exemple). L'esquisse de notre algorithme d'oscillation peut être établie comme suit (Algorithme 3.2).

Algorithme 3.2 : Algorithme d'Oscillation

x : solution réalisable;
stop = Faux;

tant que Non Stop **faire**

// Phase Constructive

Etape 1.1 : Générer des routes admissibles pour les avions ;
Etape 1.2 : Générer des itinéraires admissibles pour les passagers ;
Etape 1.3 : Affecter des routes aux avions et des passagers aux vols simultanément ;

// Phase Destructive

Etape 2.1 : Supprimer des routes de certains avions selon un ou plusieurs critères;
Etape 2.2 : Affecter des passagers aux vols restants;

Mise à jour des paramètres de la stratégie d'oscillation;

fin tant que

Notons que dans cet algorithme les deux phases de construction et de destruction peuvent être permutées. Les étapes 1.1, 1.2, 2.1 sont gérées par des heuristiques et contrôlées par un modèle en nombres entiers mixtes dans les étapes d'affectation 1.3 et 2.2.

Le modèle utilisé dans les étapes d'affectation prend en entrée un ensemble de routes possibles pour les avions et un ensemble d'itinéraires possibles pour les passagers. Ce modèle peut être appliqué dans les deux phases (constructive et destructive). Dans la phase constructive, il considère les places vides dans les vols existants et l'ensemble des places dans les vols potentiellement créés. Il s'agit alors de prendre en compte simultanément les avions et les passagers. Dans la phase destructive, il ne considère que les places vides dans les vols existants. Les heuristiques de génération de routes et d'itinéraires reposent sur des recherches arborescentes tronquées, de type "meilleur d'abord". Elles prennent en entrée une situation globale courante, un groupe (avions ou passagers), et cherche à générer un ensemble de routes/itinéraires pour les membres de ce groupe, de leur position courante à l'aéroport de destination (placement final ou aéroport de disponibilité pour un avion).

L'heuristique de suppression de routes pour les avions et d'itinéraires pour les passagers repose sur le principe dual de la génération. Comme nous le verrons dans la suite, nous éliminons surtout des sous-chemins de routes qui sont les moins remplis par rapport aux capacités des avions. Nous donnons une priorité à la suppression de circuits pour maintenir la réalisabilité de la solution.

Dans la partie suivante, nous présentons plus en détails les différentes étapes de l'algorithme d'oscillation. Nous commençons par les heuristiques de génération.

3.4.1 Heuristiques constructives

Nous avons développé deux familles d'heuristiques. La première cherche à créer de nouveaux vols pour les avions afin de satisfaire des demandes supplémentaires de passagers et éventuellement d'assurer les contraintes de positionnement final des avions. La deuxième cherche à générer des itinéraires pour les passagers à partir d'un ensemble de vols.

3.4.1.1 Génération des routes pour les avions

Notre heuristique de création de vols cherche à générer un ensemble de routes possibles pour un avion dans un intervalle (ou tranche) de disponibilité. Un intervalle de disponibilité est un intervalle de temps où l'avion n'est pas mobilisé et peut effectuer des vols. Dans notre algorithme de génération de vols, nous avons distingué deux types d'intervalle de

disponibilité : les intervalles "intérieures" qui se situent entre deux vols successifs déjà existants (par exemple dans la Figure 3.7, l'avion *a* a une tranche de disponibilité intérieure entre les vols 1 et 4), et l'intervalle "final" qui se situe après le dernier vol existant (dans la Figure 3.7 l'avion *a* a une tranche de disponibilité finale après le vol 5). Dans ce dernier cas, nous cherchons éventuellement à rejoindre l'aéroport de positionnement final de l'avion, si nous cherchons à minimiser les coûts associés aux contraintes de positionnement. Cette décision dépend de la stratégie globale sur la préférence donnée à ces contraintes ou à celles liées au nombre de passagers arrivant à destination. Ces deux types de tranches conduisent à deux types de problèmes légèrement différents : dans le premier cas, nous cherchons à créer une tournée pour l'avion. Dans le deuxième cas, il s'agit d'une tournée "ouverte" où la destination finale n'est pas précisée. Le premier cas est un peu plus délicat à gérer, car il nécessite de s'assurer qu'il est toujours possible d'atteindre l'aéroport de retour dans une fenêtre de temps donnée pour assurer la continuité des vols.

L'heuristique de création de tournées fonctionne selon un critère glouton. Nous avons testé plusieurs critères pour décider de l'ajout d'un vol : aéroports les moins visités, se rapprocher du positionnement final (s'il existe), vol le plus court, aéroports les moins surchargés, vol le plus demandé par les passagers, vol qui part au plus tôt. Le critère le plus efficace en pratique est lié aux passagers qui peuvent emprunter le vol créé.

Initialement, l'avion est dans la position de départ spécifiée par la tranche de disponibilité considérée. L'heuristique est initialisée avec une tournée vide. Nous cherchons ensuite à ajouter itérativement un vol à la fin de la tournée courante. Chaque prise de décision (ajout d'un vol) dépend d'une procédure de "vote" pour décider de la prochaine destination. Chaque passager présent dans l'aéroport courant au moment considéré, ainsi que tous ceux qui peuvent atteindre l'aéroport courant à l'aide d'un vol déjà existant, participent au vote. Des poids différents sont affectés aux passagers selon qu'ils sont sur un itinéraire retour ou qu'ils doivent emprunter un vol pour arriver à l'aéroport courant. Les destinations sont classées par ordre décroissant des votes, et la première qui satisfait les différentes contraintes (capacité de décollage et d'atterrissage des aéroports, distance franchissable, durée du vol avant la maintenance) est choisie. Le choix des stratégies d'exploration est d'autant plus critique qu'il est possible qu'aucune solution ne soit trouvée à la fin de la méthode, si les capacités des aéroports ou les tranches de disponibilité de l'avion sont trop contraignantes.

L'algorithme suivant décrit l'heuristique de création d'une tournée sur une tranche de disponibilité. Dans cet algorithme nous utilisons deux critères d'arrêt qui sont le nombre maximum de nœuds à explorer et le nombre maximum de solutions à générer.

Algorithme 3.3 : Procédure de création de tournées

```
// Entrée
    a ∈ A            : un avion ayant une période de disponibilité.
    [Disp⁻, Disp⁺]   : un intervalle de disponibilité de a.
    k1               : l'aéroport d'origine de l'avion a.
    k2               : l'aéroport de destination de a.
    Max_n            : le nombre maximum de nœuds à explorer.
    Max_s            : le nombre maximum de routes générées.
    Liste_routes     : la liste des solutions (liste de tournées).
    List             : une liste de routes partielles.
// Initialisation
    Liste_routes := Ø ;
    List = Ø ;
    pour chaque aéroport y atteignable depuis k1 après Disp⁻ et avant Disp⁺ en un seul vol faire
        si k2 n'existe pas ou y = k2 alors ajouter (k1, y) à Liste_routes ; fin si
        ajouter (k1, y) à List;
    fin pour
    cpt = 0 ;
// Recherche pas à pas
    tant que cpt < Max_n et | Liste_routes |< Max_s et List ≠ Ø faire
        cpt = cpt +1 ;
        retirer la meilleure solution partielle s de List ;
        soit x l'aéroport courant d'arrivée de s ;
        pour chaque aéroport y atteignable depuis x en un seul vol avant Disp⁺ faire
            si k2 n'existe pas ou y = k2 alors ajouter s ∪ (x, y) à Liste_routes ; fin si
            ajouter s ∪ (x, y) à List;
        fin pour
    fin tant que
retourner Liste_routes ;
```

3.4.1.2 Génération des itinéraires pour les passagers

L'heuristique de génération d'itinéraires a pour but de produire un ensemble de chemins possibles pour réaffecter un groupe de passagers donné, éventuellement en divisant ce groupe en plusieurs sous-groupes. Le principe général de l'heuristique est de chercher les itinéraires de longueur l empruntant les vols et permettant d'acheminer les passagers du lieu où ils se trouvent vers leur destination finale. Pour chaque groupe de passagers, les itinéraires sont examinés dans l'ordre d'arrivée au plus tôt. L'algorithme de génération d'itinéraires repose sur le même principe que celui de génération de vols. Une des différences est qu'ici il n'est

envisageable d'emprunter que des vols ayant des places libres, alors que dans la génération de routes il est envisageable d'aller vers n'importe quel autre aéroport à partir d'un aéroport donné (en respectant tout de même les contraintes du problème). Le nombre de solutions possibles pour les itinéraires est par conséquent beaucoup moins grand que celui de la génération de routes. De plus, pour les avions nous autorisons la visite d'un aéroport plusieurs fois, ce qui n'est pas le cas pour les passagers. Finalement, nous devons élaguer tous les itinéraires qui dépassent le retard maximum autorisé.

L'heuristique prend en entrée l'ensemble des groupes de passagers n'ayant pas atteint leur destination finale. Chaque groupe est caractérisé par sa position courante, sa date de disponibilité, ainsi que la situation courante (vols existants, affectations déjà effectuées). La méthode possède plusieurs paramètres qui déterminent l'ordre dans lequel les groupes de passagers sont considérés, la manière de générer les itinéraires, et le nombre potentiel de passagers du groupe pouvant être affectés. En ce qui concerne l'ordre des groupes, nous avons testé plusieurs possibilités : disponibilité au plus tôt, nombre de passagers dans le groupe, itinéraire déjà commencé ou non. En pratique, les meilleurs résultats sont obtenus en considérant le nombre de passagers, avec un poids plus important pour les passagers en correspondance.

Les différents itinéraires possibles pour un groupe donné sont générés de la façon suivante. La file de solutions est initialisée avec tous les itinéraires possibles de longueur un (un seul vol) vers n'importe quelle destination à partir de sa position courante. A chaque étape, l'algorithme sélectionne la solution partielle la plus *profitable* et cherche à l'étendre en lui ajoutant un vol. Chaque itinéraire partiel ainsi obtenu est placé dans l'ensemble des solutions s'il arrive à destination. Sinon il est placé dans la file de priorité et l'algorithme continue. L'algorithme s'arrête lorsque l'un des critères d'arrêt spécifiés est satisfait (nombre de solutions, nombre d'état explorés, ...). L'efficacité de la méthode dépend en grande partie de la pertinence des différents critères de choix. Nous avons testé plusieurs stratégies : itinéraire contenant les aéroports les moins visités d'abord, ceux qui se rapprochent le plus de la destination finale, minimum d'attente dans les aéroports. Se rapprocher le plus possible de la destination finale est souvent la meilleure stratégie (ce qui peut se comprendre car il est inutile de trop s'éloigner sous peine de devoir ajouter plusieurs autres vols à l'itinéraire).

3.4.1.3 Modèle de création des routes et d'affectation des passagers

La phase d'affection vise à trouver une solution de bonne qualité qui combine au mieux la création de nouvelles routes pour les avions et l'affectation des passagers. Pour prendre en considération les avions et les passagers simultanément, nous avons modélisé le problème sous forme d'un sac à dos multidimensionnel en variables binaires et entières. Les variables binaires correspondent aux choix des routes et les variables entières aux nombres de passagers affectés dans chaque groupe. Ce modèle prend en entrée un ensemble d'itinéraires possibles pour les passagers et un ensemble de routes possibles pour les avions. Les contraintes de capacités aéroportuaires ainsi que les capacités des cabines des avions sont prises en compte. Pour décrire plus en détails le modèle proposé, nous introduisons d'abord un ensemble de notations.

R_a : ensemble des routes potentielles considérées pour l'avion a.

R'_a : ensemble des routes fixées pour l'avion a.

$I_{i,a}$: ensemble des vols dans la route i de l'avion a.

S_p : ensemble des itinéraires potentiels pour le groupe de passagers p.

$V_{s,p}$: ensemble des vols dans l'itinéraire s du groupe p.

$C_{j,i,a}$: capacité résiduelle de l'avion a associé au vol j de la route i.

$c^1_{j,p}$: coût du groupe p par rapport à l'itinéraire j égal au coût d'annulation d'un passager du groupe p moins le coût du retard éventuel (si un itinéraire est choisi).

$c^2_{i,a}$: coût de création des vols de l'avion a dans la route i.

Nous donnons dans la Figure 3.8 un exemple de routes possibles pour un avion a donné dans un intervalle de disponibilité donné. Supposons que l'avion a a un intervalle de disponibilité dans l'aéroport k_2 de la tranche T1 jusqu'à la tranche T8. L'ensemble R_a est l'ensemble des routes qui peuvent être parcourues par l'avion a comme le montre le graphique. L'avion a est disponible à l'aéroport k_2 dans l'intervalle [T1, T8]. Il peut par exemple emprunter les vols de (k_2, t_1) vers (k_3, t_3), puis de (k_3, t_3) vers (k_4, t_5) et terminer son chemin par le vol de (k_4, t_6) vers (k_2, t_8). L'ensemble des vols précédents forme une route possible qui appartient à l'ensemble R_a. Il peut également prendre les deux vols ((k_2, t_1) → (k_1, t_3)) puis ((k_1, t_4) → (k_2, t_7)) (nous remarquons que dans ce dernier vol l'avion sera à l'aéroport k_2 avant la tranche T_8). Cet avion peut aussi emprunter d'autres chemins si le réseau le permet.

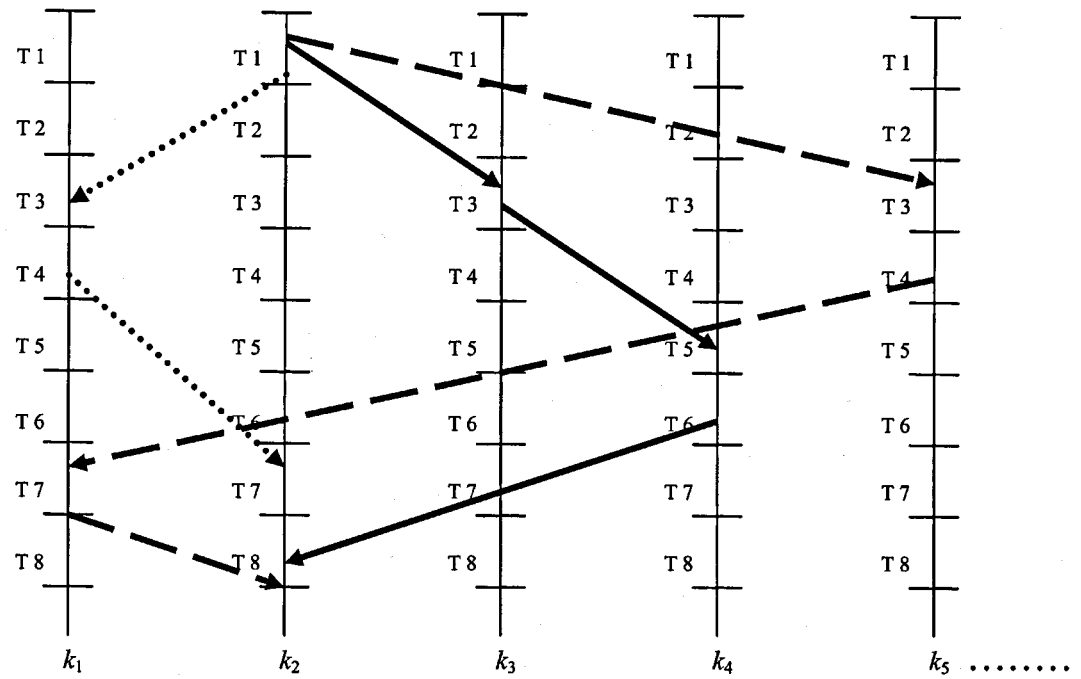


Figure 3.8 : Génération des chemins d'avions

Pour modéliser notre problème nous introduisons les variables suivantes :

$x_{i,a}$: variable binaire égale à 1 si l'avion a emprunte la route i, 0 sinon.

$z_{j,i,a}$: variable binaire égale à 1 si l'avion a emprunte le vol j de la route i, 0 sinon.

$y_{s,p}$: variable entière égale au nombre de passagers du groupe p qui empruntent l'itinéraire s.

$f_{s,p,j,i,a}$: variable entière égale au nombre de passagers du groupe p qui empruntent le vol j du chemin i de l'avion a sur l'itinéraire s.

L'ensemble des contraintes du modèle s'établit alors comme suit :

$$x_{i,a} = z_{i,j,a} \qquad a \in A,\ i \in R_a \text{ et } j \in I_{i,a} \qquad \text{(2-a)}$$

$$\sum_{i \in R_a} x_{i,a} \leq 1 \qquad a \in A \qquad \text{(2-b)}$$

$$x_{i,a} = 1 \qquad a \in A,\ i \in R'_a \qquad \text{(2-c)}$$

$$\sum_{a \in A} \sum_{i \in R_a} \sum_{j \in I_{i,a}} z_{j,i,a} \leq C^{-}_{k,h} \qquad k \in K,\ h \in H \qquad \text{(2-d)}$$

$$\sum_{a \in A} \sum_{i \in R_a} \sum_{j \in I_{i,a}} z_{j,i,a} \leq C^{+}_{k,h} \qquad k \in K,\ h \in H \qquad \text{(2-e)}$$

$$y_{s,p} = f_{s,p,j,i,a} \qquad s \in S_p,\ p \in P \text{ et } (i,j,a) \in V_{s,p} \qquad \text{(2-f)}$$

$$\sum_{s \in S_p} y_{s,p} \leq G_p \qquad p \in P \qquad \text{(2-g)}$$

$$\sum_{p\in P}\sum_{s\in S_p} f_{s,p,j,i,a} \le C_{j,i,a} z_{j,i,a} \qquad a \in A,\ i \in R_a \text{ et } j \in I_{i,a} \qquad \text{(2-h)}$$

$$x_{i,a}, z_{j,i,a} \in \{0, 1\}, y_{s,p}, f_{s,p,j,i,a} \in N \qquad a \in A,\ i \in R'_a \cup R_a,\ p \in P,\ s \in S_p,\ j \in I_{i,a}$$

Les contraintes (2-a) expriment le fait que si l'avion emprunte un chemin alors il doit parcourir tous les vols de ce chemin. Les contraintes (2-b) indiquent que chaque avion ne peut prendre au maximum qu'un seul chemin parmi les chemins générés dans un même intervalle de disponibilité. Les contraintes (2-c) assurent que les chemins existants sont conservés et fixés (ces chemins peuvent être supprimés complètement ou partiellement dans la partie destructive de la méthode). Les contraintes (2-d) indiquent que l'ensemble des vols qui atterrissent dans une tranche horaire ne doivent pas dépasser la capacité d'atterrissage de cette tranche. Les contraintes (2-e) ont le même rôle pour les capacités de décollage. Les contraintes (2-f) assurent que l'ensemble des passagers affectés à un itinéraire est le même sur chaque vol composant cet itinéraire. Les contraintes (2-g) expriment que la somme de tous les passagers du groupe p affectés dans tous les itinéraires ne doit pas dépasser le nombre de passagers du groupe. Les contraintes (2-h) assurent le respect des contraintes de capacité des avions. Si le vol associé n'est pas créé dans cette itération, cela signifie qu'il existe et qu'il a déjà des passagers à transporter. Nous déduisons alors la capacité résiduelle du vol.

L'objectif du modèle est obtenu à partir des coûts définis pour le problème global, et vise à maximiser le gain associé aux passagers pour lesquels il est possible de trouver un itinéraire moins le coût de retard engendré par cet itinéraire :

$$\text{Maximiser} \sum_{p\in P}\sum_{j\in S_p} c^1_{j,p} y_{j,p} - \sum_{a\in A}\sum_{i\in R_a} c^2_{i,a} x_{i,a}$$

Nous remarquons que nous pouvons réduire la taille du problème précédent en remplaçant les variables $z_{j,i,a}$ par $x_{i,a}$ et $f_{s,p,j,i,a}$ par $y_{s,p}$. Plus formellement, nous obtenons alors grâce aux contraintes d'égalité (2-a) et (2-f) le modèle mathématique réduit suivant :

$$\text{Maximiser} \sum_{p\in P}\sum_{j\in S_p} c^1_{j,p} y_{j,p} - \sum_{a\in A}\sum_{i\in R_a} c^2_{i,a} x_{i,a}$$

$$\text{s.c.} \quad \sum_{i\in R_a} x_{i,a} \le 1 \qquad a \in A$$

$$x_{i,a} = 1 \qquad a \in A,\ i \in R'_a$$

$$\sum_{a\in A}\sum_{i\in R_a} x_{i,a} \le C^-_{k,h} \qquad k \in K,\ h \in H$$

$$\sum_{a \in A} \sum_{i \in R_a} x_{i,a} \leq C^{+}_{k,h} \qquad k \in K, h \in H$$

$$\sum_{j \in S_p} y_{j,p} \leq G_p \qquad p \in P$$

$$\sum_{s \in S_p} \sum_{p \in P} y_{s,p} \leq C_{j,i,a} x_{i,a} \qquad a \in A, i \in R_a \text{ et } j \in I_{i,a}$$

$$x_{i,a} \in \{0, 1\}, y_{j,p} \in N \qquad a \in A, i \in R'_a \cup R_a, p \in P, j \in S_p$$

La taille de ce problème NP-Difficile dépend principalement du nombre de chemins générés pour les avions et du nombre d'itinéraires générés pour les passagers. Il est indispensable d'avoir une résolution rapide de ce problème pour pouvoir évaluer le maximum de chemins et d'itinéraires possibles. Dans le cadre de notre étude, ce modèle mathématique est résolu à l'aide du logiciel CPLEX avec un temps limité.

3.4.2 Heuristique destructive

L'idée générale de la suppression de vols vient de la constatation que les avions ne sont pas nécessairement *pleins* dans la situation initiale ou après la création et l'affectation des passagers. Il se peut que nous trouvions des vols à moitié vides par exemple. Cette situation entraîne l'idée de supprimer certains vols pour :

- libérer des capacités des aéroports, et ainsi laisser de la place pour la création de vols.
- augmenter le taux de remplissage des autres vols après réaffectation des passagers dans ces vols. Nous considérons ici qu'un vol transportant peu de passagers par rapport à sa capacité est "inefficace".

Devant la difficulté d'obtenir une solution réalisable du problème, le processus de suppression assure le maintien de la réalisabilité de la solution. Pour cela, il faut donc respecter les contraintes de continuité des vols : nous ne supprimons que des circuits effectués par des avions. De plus, nous ne considérons pas les circuits qui contiennent un sommet de maintenance (*aéroport de maintenance, date de maintenance*). Nous avons retenu comme critère de sélection des vols inefficaces le taux de remplissage des vols du circuit. Il est cependant possible d'utiliser d'autres critères tels que le nombre de passagers transportés, le nombre de places des avions ou le type de vols du circuit. Nous présentons dans la Figure 3.9 un exemple de suppression d'un circuit pour un avion.

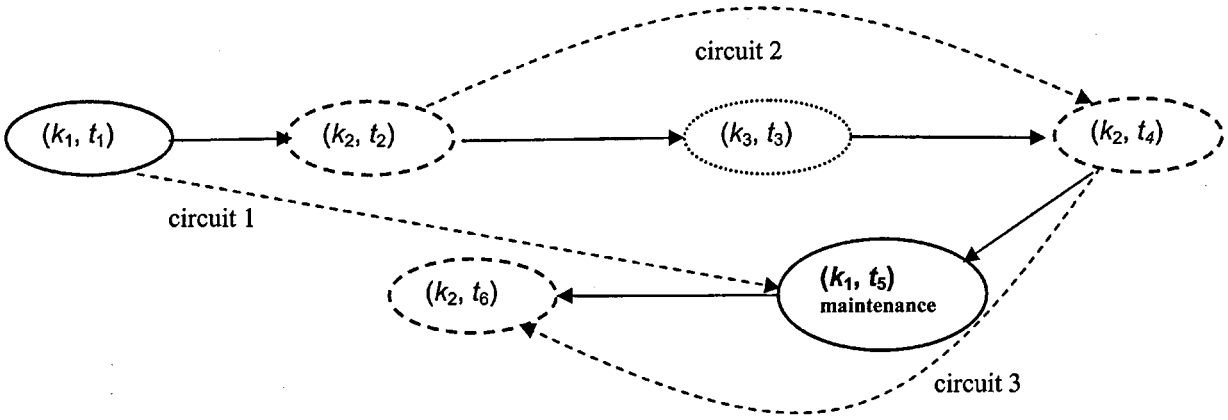


Figure 3.9 : Suppression des circuits d'un avion

Nous remarquons dans la Figure 3.9 que l'avion a effectue trois circuits durant son parcours.

Circuit 1 : $(k_1, t_1) \rightarrow (k_2, t_2) \rightarrow (k_3, t_3) \rightarrow (k_2, t_4) \rightarrow (k_1, t_5)$.

Circuit 2 : $(k_2, t_2) \rightarrow (k_3, t_3) \rightarrow (k_2, t_4)$.

Circuit 3 : $(k_2, t_4) \rightarrow (k_1, t_5) \rightarrow (k_2, t_6)$.

Parmi les trois circuits existants nous ignorons le circuit 3 car si l'avion a est à (k_2, t_4) et que nous supprimons les vols vers (k_1, t_5) et (k_2, t_6) alors il résulte une solution qui ne réalise pas la contrainte de maintenance de l'avion a. Le choix de la suppression se fait alors entre les circuits 1 et 2. À titre d'exemple le taux de remplissage pour le circuit 2 est calculé par la formule suivante :

$$\left(\frac{NBP_a\left((k_2,t_2) \rightarrow (k_3,t_3)\right)}{C_a} + \frac{NBP_a\left((k_3,t_3) \rightarrow (k_2,t_4)\right)}{C_a} \right) \Big/ 2$$

avec $NBP_a\left((k,t) \rightarrow (k',t')\right)$ le nombre de passagers dans le vol allant de k à k' entre t et t' par l'avion a, et C_a la capacité de l'avion a. Il est important de préciser que nous n'autorisons la suppression que si sa combinaison avec l'affectation des passagers améliore la valeur de la solution courante.

La procédure de suppression se fait en respectant les deux étapes suivantes : (1) les circuits sont triés dans l'ordre croissant de leur taux de remplissage ; (2) la suppression de chaque circuit est évaluée en termes de réalisabilité et en termes d'amélioration de la fonction objectif. Notons qu'en pratique nous limitons le nombre de circuits évalués et que nous utilisons un seuil pour le taux minimum de remplissage.

3.5 Résultats numériques

Nous présentons dans cette section une synthèse des résultats obtenus lors du challenge. La plupart de ces résultats sont consultables sur le site internet du challenge. L'ordinateur

utilisé possède un processeur de 2Ghz et 2 Gb de RAM. Le temps d'exécution est limité à 10 minutes pour l'évaluation.

Il y avait trois ensembles d'instances A, B et X. L'ensemble A contient 10 instances et a été utilisé pour la phase de qualification du challenge réunissant au départ 29 équipes de différentes nationalités. Les ensembles B et X contiennent respectivement 10 et 8 instances, et ont été utilisées pour départager les 11 équipes finalistes.

Le Tableau 3.1 présente les résultats numériques concernant les instances *A*. Il donne par colonne, le nom de l'instance, le nombre d'aéroports, le nombre d'avions, le nombre d'itinéraires. Le champ "Réal" indique si la solution trouvée par le modèle mathématique dans la première phase est réalisable ou non ; le champ "Q (*Qualif*)" indique la meilleure valeur trouvée par toutes les équipes finalistes lors de la qualification (nous n'avons pas à notre disposition la valeur finale des autres équipes pour ces instances) ; le champ "QI (*Qualité Initiale*)" donne la valeur obtenue par la première phase de notre algorithme ; et le champ "QF (*Qualité Finale*)" indique notre valeur finale. Les résultats dans ces trois derniers champs sont donnés en €. Une valeur en gras dans les deux dernières colonnes indique l'obtention de la meilleure valeur.

Instance	Aéroports	Avions	Itinéraires	Réal.	Q (€)	QI (€)	QF (€)
A01	35	85	1943	MIP	23 789.0	28 155.6	**-11 059.1**
A02	35	85	1943	MIP	83 515.0	126 684.7	90 429.4
A03	35	85	1943	MIP	131 694.6	**131 694.6**	**79 647.9**
A04	35	85	1943	MIP	108 081.9	523 678.8	**36 379.5**
A05	35	85	3959	MIP	3 717376.3	8 092 977.5	**1 332420.6**
A06	35	85	1872	MIP	44 305.0	71 355.2	61 752.9
A07	35	85	1872	MIP	202 247.7	245 696.7	**157 931.3**
A08	35	85	1872	MIP	329 521.8	**329 521.8**	**199 834.3**
A09	35	85	1872	MIP	187 144.4	1 096 303.8	**128 526.4**
A10	35	85	3773	MIP	7 210166.9	17166321.7	**3570361.9**

Tableau 3.1 : Résultats pour les instances A

Notre approche a été classée seconde lors de la phase de qualification par rapport aux valeurs fournies dans le champ QI. Le Tableau 3.1 montre que l'utilisation de la programmation mathématique sans la procédure de réparation était suffisante pour obtenir une solution réalisable pour toutes les instances du groupe A. La dernière colonne montre que l'amélioration obtenue en appliquant la stratégie d'oscillation est considérable, notamment pour les plus grandes instances A05 et A10. Cette stratégie nous a même permis d'obtenir des coûts négatifs (profit supplémentaire pour la compagnie) comme dans le cas de l'instance A01. La version complète de notre algorithme fournit la meilleure valeur pour 8 instances sur les 10 de l'ensemble A.

Nous donnons dans le Tableau 3.2 les résultats obtenus lors de la finale du challenge. Le Tableau 3.2 présente par colonne, le nom de l'instance, le nombre d'aéroports, le nombre d'avions, le nombre d'itinéraires, la phase ayant permis d'obtenir une solution réalisable, la qualité de la solution obtenue dans la première phase, et la qualité de notre solution finale. Notons que la qualité des solutions est relative aux meilleures solutions obtenues par les finalistes. Cette qualité est calculée par la formule $(zw(I) - z(I)) / (zw(I) - zb(I))$, où $zb(I)$ (resp. $zw(I)$) représente la valeur de la meilleure (resp. la plus mauvaise) solution pour l'instance I, et $z(I)$ représente le coût de la solution trouvée par notre approche. Notons que si une méthode ne fournit pas de solution ou si la solution retournée est irréalisable, la valeur $zw(I)$ considérée est égale à deux fois la plus mauvaise valeur trouvée par les autres équipes. Une valeur de 1.0 dans une des deux dernières colonnes signifie donc que notre approche obtient la meilleure valeur parmi les finalistes.

Instance	Aéroports	Avions	Itinéraires	Réalisable	QI	QF
B01	44	255	11214	MIP	0.80	0.89
B02	44	255	11214	Réparation	0.47	0.71
B03	44	255	11214	MIP	0.83	0.90
B04	44	255	11214	MIP	0.82	0.90
B05	44	255	11214	MIP	0.86	0.96
B06	44	255	11565	MIP	0.76	0.91
B07	44	255	11565	Réparation	0.26	0.73
B08	44	255	11565	MIP	0.76	0.89
B09	44	255	11565	MIP	0.77	0.90
B10	44	255	11565	MIP	0.82	**1.00**
XA01	35	85	1943	MIP	0.89	**1.00**
XA02	35	85	3959	MIP	0.95	**1.00**
XA03	35	85	1872	MIP	0.80	**1.00**
XA04	35	85	3773	MIP	0.20	**1.00**
XB01	44	255	11214	MIP	0.92	0.96
XB02	44	255	11214	MIP	0.94	0.99
XB03	44	255	11565	MIP	0.89	0.96
XB04	44	255	11565	MIP	0.91	**1.00**

Tableau 3.2 : Résultats pour les instances B et X

Le Tableau 3.2 montre que notre approche obtient des résultats intéressants, en particulier pour les instances de l'ensemble X qui étaient inconnues par les finalistes. Les instances XA01 à XA04 ont été produites à partir de l'ensemble des instances A. Pour l'ensemble B les résultats moyens obtenus pour les instances B02 et B07 sont étroitement liés à la présence d'une combinaison de contraintes de maintenance empêchant le modèle de la première phase de trouver une solution réalisable. Notre algorithme doit alors s'éloigner du programme de vols initial pour chercher une solution réalisable à l'aide de la procédure de réparation. Un

autre élément qui peut expliquer le comportement de notre méthode sur l'ensemble B est la recherche partielle qui n'est pas suffisante pour produire de "bonnes" routes dans la phase constructive. Toutefois, nous pouvons noter que pour les instances les plus difficiles (celles dans les capacités d'aéroports sont les plus perturbées, à savoir B05 et B10) notre méthode obtient des résultats très encourageants.

Dans le tableau suivant, publié sur le site du ce challenge, nous présentons le score de chacune des 9 équipes qualifiées pour la phase finale. Ce tableau donne le classement final obtenu sur la base du score normalisé moyen calculé comme expliqué précédemment.

Classement	1	**2**	3	4	5	6	7	8	9
Catégorie	Senior	**Senior**	Senior	Junior	Junior	Senior	Senior	Junior	Junior
Score	95.90	**92.73**	74.26	72.01	70.62	70.31	64.02	42.02	20.48

Tableau 3.3 : Score final des 9 premières équipes qualifiées

Nous remarquons que notre heuristique, classée en deuxième position et en catégorie senior, a un score très proche de l'équipe classée en première position. L'écart avec les autres équipes et par contre nettement plus important.

3.6 Conclusions

La gestion de perturbations dans le domaine aérien a fait l'objet de nombreux travaux ces dernières décennies. L'importance de ce problème repose sur l'énorme enjeu économique qu'il représente. Pour résoudre celui-ci, nous avons choisi une méthode hybride basée sur une stratégie d'oscillation qui utilise des heuristiques et des méthodes exactes. Après avoir produit une première solution réalisable à l'aide d'un programme linéaire en nombres mixtes et une heuristique, nous alternons entre des phases constructives et destructives pour améliorer cette solution en ajoutant et en supprimant des vols et des itinéraires. La création de vols et l'affectation des passagers aux vols sont contrôlées par un modèle linéaire en nombre entiers tandis que la génération des vols et des itinéraires potentiels est assurée par des méthodes heuristiques. Notre approche a obtenu des résultats intéressants dans un contexte difficile de challenge (ressources et temps d'exécution limités). La deuxième place obtenue au classement général confirme que l'hybridation entre méthodes exactes et approchées est efficace pour des problèmes NP-Difficiles de grande taille.

Notre méthode peut encore être améliorée en intégrant la mémoire adaptative dans le processus de recherche et en révisant les paramètres de la génération de routes pour les avions et des itinéraires pour les passagers. De plus, un renforcement de la procédure de suppression peut nous permettre d'améliorer le comportement oscillatoire de notre méthode. Finalement,

l'exploitation d'informations duales pour la génération des itinéraires ou la permission de visiter des solutions non réalisables durant l'algorithme d'oscillation sont d'autres éléments possibles à explorer.

Bibliographie :

Aickelin, U. et Dowsland, K. (2000). Exploiting Problem Structure in a Genetic Algorithm Approach to a Nurse Rostering Problem. *Journal of Scheduling*, **3**: 139-153.

Akbar, M.M., Ergun, O. et Punnen, A.P. (2001). Heuristic solutions for the multiple-choice multi-dimensional knapsack problem. *Dans: International Conference on Computer Science, San Francisco, USA.*

Akbar, M.M., Rahman, M.S., Kaykobad, M., Manning, E.G. et Shoja, G.C. (2006). Solving the multidimensional multiple-choice knapsack problem by constructing convex hulls. *Computers & Operations Research*, **33**(5): 1259–1273.

Anandalingam, G. et White, D. J. (1990). A solution for the linear static Stackelberg problem using penalty function. *IEEE Transactions Automatic Control*, **35**: 1170-1173.

Andersson, T. (2006).Solving the flight perturbation problem with metaheuristics. *Journal of Heuristics*, **12**: 37-53.

Arguello, M., Bard, J. et Yu, G. (1997). A GRASP for aircraft routing in response to groundings and delays. *Journal of Combinatorial Optimization*, **1**(3): 211-228.

Armstrong, R.D., Kung, D.S., Sinha, P. et Zoltners, A.A. (1983). A computational study of a multiple-choice knapsack algorithm. *ACM Transactions on Mathematical Software*, **9**(2): 184-198.

Audet, C., Hansen, P., Jaumard, B. et Savard, G. (1997). Links between linear bilevel and mixed 0-1 programming problems. *Journal of Optimization Theory and Applications*, **93**: 273–300.

Audet, C., Savard, G. et Zghal, W. (2007). New branch-and-cut algorithm for bilevel linear programming. *Journal of Optimization Theory and Applications*, **134**(2): 353-370.

Balas, E. et Martin, C.H. (1980). Pivot and Complement - a Heuristic for Zero-One Programming. *Management Science*, **26**(1): 86-96.

Balas, E. et Zemel, E. (1980). An Algorithm for Large Zero-One Knapsack Problems. *Operations Research*, **28**: 1130-1154.

Bard, J.F. et Moore, J.T. (1990). A branch and bound algorithm for the bilevel programming problem. *SIAM Journal on Scientific and Statistical Computing*, **11**: 281–292.

Bard, J.F. et Moore, J.T. (1992). An Algorithm for the Discrete Bilevel Programming Problem. *Naval Research Logistics*, **39**: 419-435.

Basnet, C. et Wilson, J. (2005). Heuristics for determining the number of warehouses for storing non-compatible products. *International Transactions in Operational Research*, **12**(5): 527-538.

Bellman, R. (1957). Dynamic Programming, *Princeton University Press, Princeton, NJ, dover paperback (2003) edition.*

Bialas, W. et Karwan, M. (1984). Two-level linear programming. *Management Science*, **30**: 1004-1020.

Bialas, W., Karwan, M. et Shaw, J. (1980). A parametric complementary pivot approach for two-level linear programming. *Technical Report 80-2, Operations Research Program, State University of New York at Buffalo.*

Blum, C., Blesa Aguilera, M.J., Roli, A. et Sampels, M. (2008). Hybrid metaheuristics: an emerging approach to optimization. *Studies in computational intelligence*, vol. **114**, *Springer.*

Boussier, S. (2008). Etude de Resolution Search pour la programmation linéaire en variables binaires. *Ph.D Université Montpellier II.*

Bracken, J. et McGill, J. (1973). Mathematical programs with optimization problems in the constraints. *Operations Research*, **21**: 37–44.

Bratu, S. et Barnhart, C. (2006). Flight operations recovery: New approaches considering passenger recovery. *Journal of Scheduling*, **9**: 279-298.

Brotcorne, L., Hanafi, S. et Mansi R. (2009). A dynamic programming algorithm for the bilevel knapsack problem. *Operations Research Letters*, **37**(3): 215–218.

Cao, J. et Kanafani, A. (1997a). Real-time decision support for integration of airline flight cancellations and delays part 1: Mathematical formulation. *Transportation Planning and Technology*, **20**: 183-199.

Cao, J. et Kanafani, A. (1997b). Real-time decision support for integration of airline flight cancellations and delays part 2: Algorithm and computational experiments. *Transportation Planning and Technology*, **20**: 201-217.

Calamai, P., et Vicente, L. (1993). Generating Linear and Linear-Quadratic Bilevel Programming Problems. *SIAM Journal on Scientific and Statistical Computing*. **14**: 770–782.

Chen, L., Khan, S., Li, K.F. et Manning, E.G. (1999). Building an adaptive multimedia system using the utility model. *Parallel and Distributed Processing, Lecture Notes in Computer Science* 1586, *Springer Verlag (InternationalWorkshop on Parallel and Distributed Realtime Systems (WPDRTS 7), ACM/IEEE/US Navy Surface Warfare Center)*, 289-298.

Cherfi, N. et Hifi, M. (2008). A column generation method for the multiple-choice multi-dimensional knapsack problem. *Computational Optimization and Applications*, DOI 10.1007/s10589-008-9184-7.

Chocolaad, C.A. (1998). Solving geometric knapsack problems using tabu search. AFIT/GOR/ENS/98M-05, *Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio.*

Chrétienne, P., Picouleau, C. (1995). Scheduling with communication delays a survey, *dans: P. Chrétienne, E.G. Coffman, J.K. Lenstra, Z. Liu (Eds.), Scheduling Theory and its Applications, John Wiley Ltd.*, New York, 65-89.

Clausen, J., Larsen, A., Larsen, J. et Rezanova, N. J. (2009). Disruption management in the airline industry: concepts, models and methods. *Computers & Operations Research*, doi: 10.1016/j.cor.2009.03.027.

Colson, B., Marcotte, P. et Savard, G. (2005). Bilevel Programming, A survey, *4OR*, **3**: 87-107.

Colson, B., Marcotte, P. et Savard, G. (2007). An overview of bilevel optimization. *Annals of Operations Research*, **153**(1): 235–256.

CPLEX. Using the cplex callable library and cplex mixed integer programming, (2008).

Dempe, S. (1996). Discrete bilevel optimization problems. *Technical report, Universität Leipzig, Wirtschaftswissenschaftliche Fakultät, Institut für Wirtschaftsinformatik.*

Dempe, S. (2002). Foundation of Bilevel Programming. *Kluwer Academic Publishers.*

Dempe, S. et Richter, K. (2000). Bilevel Programming with Knapsack Constraint. *Central European Newspaper of Operations Research*, **8**: 93-107.

DeNegre, S.T. et Ralphs, T.K. (2008). A branch-and-cut algorithm for integer bilevel linear programs. *Operations Research and Cyber-Infrastructure*, **47**: 65–78.

Detienne, B., Péridy, L., Pinson, E. et Rivreau, D. (2009). Cut generation for an employee timetabling problem. *European Journal of Operational Research*, **197**: 1178-1184.

Dyer, M.E. (1980). Calculating Surrogate Constraints. *Mathematical Programming*, **19**: 255-278.

Dyer, M.E., Riha, W.O. et Walker, J. (1995). A hybrid dynamic programming/branch-and-bound algorithm for the multiple-choice knapsack problem. *Journal of Computational and Applied Mathematics*, **58**: 43-54.

Edmunds, T. et Bard, J. (1991). Algorithms for Nonlinear Bilevel Mathematical Programming. *IEEE Transaction on systems, Man and Cybernetics*, **21**: 83-89.

Fayard, D. et Plateau, G. (1981). An algorithm for the Solution of the 0-1 Knapsack Problem. *Computing*, **28**(3): 269-287.

Feng, Y. (2001). Resource allocation in Ka-band satellite systems. *Master's thesis, University of Maryland. Technical Report for the Institute for Systems Research.*

Fischetti, M. et Lodi, A. (2003). Local Branching. *Mathematical Programming*, **98**: 23–47.

Fortuny-Amat, J. et McCarl, B. (1981). A Representation and Economic Interpretation of a Two-Level Programming Problem. *Journal of the Operational Research Society*, **32**: 783-792.

Fréville, A. et Hanafi, S. (2005). The Multidimensional 0-1 Knapsack Problem – Bounds and Computational Aspects. Annals of Operations Research, 139(1): 195–227.

Fréville, A. et Plateau, G. (1986). Heuristics and reduction methods for multiple constraints 0-1 linear programming problems. *European Journal of Operational Research*, **24**: 206–215.

Fréville, A. et Plateau, G. (1993). An exact search for the solution of the surrogate dual of the 0-1 bidimensional knapsack problem, *European Journal of Operational Research*, **68**: 413-421.

Gabteni, S. et Grönkvist, M. (2008). Combining column generation and constraint programming to solve the tail assignment problem. *Annals of Operations Research*, doi:10.1007/s10479-008-0379-1.

Gavish, B. et Pirkul, H. (1982). Allocation of databases and processors in a distributed data processing. Management of Distributed Data Processing (J. Akola, ed.). North-Holland: Amsterdam, pp. 215–231.

Gavish, B. et Pirkul, H. (1985). Efficient algorithms for solving multiconstraint zeo-one knapsack problems to optimality. *Mathematical Programming*, **31**: 78–105.

Geoffrion, A.M. (1974). Lagrangean relaxation for integer programming. *Mathematical Programming Study*, **2**: 82-114.

Glover, F. (1965). A Multiphase-Dual Algorithm for the Zero-One Integer Programming Problem. *Operations Research*, **13**: 879-919.

Glover, F. (1975). Surrogate Constraint Duality in Mathematical Programming. *Operations Research*, **23**(3): 434-451.

Glover, F. (1989). Tabu Search-Part I. ORSA, *Journal on Computing*, **1**: 190-206.

Glover, F. (1990). Tabu Search-Part II. ORSA, *Journal on Computing*, **2**: 4-32.

Glover, F. (2006). Parametric tabu-search for mixed integer programs. *Computers & Operations Research*, **33**: 2449-2494.

Glover, F. et Kochenberger, G. (1996). Critical event tabu search for knapsack problems. *Dans: I.H. Osman and J.P. Kelly (eds.), Meta Heuristics: Theory and Applications*, pages 407-427, *Kluwer, Boston.*

Glover, F. et Laguna, M. (1997). Tabu search. *Kluwer Academic Publishers.*

Gondran, M. et Minoux, M. (1995). Graphes et Algorithmes. *Eyrolles, troisième édition.*

Goodman, M.D., Dowsland, K.A. et Thompson, J.M. (2007). A GRASP-knapsack hybrid for a nurse-scheduling problem. *Journal of Heuristics.*

Gopalan, R. et Talluri, K. (1998a). Mathematical models in the airline schedule planning: a survey. *Annals of the Operational Research*, **76**: 155-185.

Gopalan, R. et Talluri, K. (1998b). The aircraft maintenance routing problem. *Operations Research*, **46**: 260-271.

Greenberg, H. et Pierskalla, W. (1970). Surrogate mathematical programs. *Operations Research*, **18**: 924–939.

Guignard, M. et Kim, S. (1987). Lagrangian decomposition: a model yielding stronger langrangian bounds. *Mathematical programming*, **39**: 215–228.

Hanafi, S. et Fréville, A. (1998). An efficient tabu search approach for the 0-1 multidimensional knapsack problem. *European Journal of Operational Research*, **106**: 659-675.

Hanafi, S. et Glover, F. (2007). Exploiting Nested Inequalities and Surrogate Constraints. *European Journal of Operational Research*, **179**(1): 50-63.

Hanafi, S., Mansi, R. et Brotcorne, L. (2008). Integer Programming Formulation of the Bilevel Knapsack Problem. *9èmes journées d'analyse numérique et d'optimisation (JANO9), Maroc*. 17-19.

Hanafi, S., Mansi, R. et Wilbaut, C. (2009). Iterative Relaxation-Based Heuristics for the Multiple-choice Multidimensional Knapsack Problem. *M.J. Blesa et al. (Eds.): HM 2009, Lecture Notes in Computer Science* 5818, 73–83.

Hanafi, S. et Wilbaut, C. (2006). Heuristiques convergentes basées sur des relaxations. *Conférence ROADEF* 2006, *Lille (France)*, février.

Hanafi, S. et Wilbaut, C. (2009). Improved convergent heuristics for the 0-1 multidimensional knapsack problem. *Annals of Operations Research*, doi: 10.1007/s10479-009-0546-z.

Hanafi, S., Wilbaut, C., Mansi, R. et Clautiaux, F. (2009). Stratégie d'oscillation pour la gestion de perturbations dans le domaine aérien. Conférence *ROADEF* 2009, *Nancy (France)*, 393-394.

Hane, C., Barnhart, C., Johnson, E., Marsten, R., Nemhauser, G. et Sigismondi, G. (1995). The fleet assignment problem: solving a large-scale integer program. *Mathematical Programming*, **70**: 211-232.

Hansen, P. (1986). The steepest ascent mildest descent heuristic for combinatorial programming. *Presented at the Congress on Numerical Methods in Combinatorial Optimization, Capri, Italy.*

Hansen, P., Jaumard, B. et Savard, G. (1992). New Branch and Bound Rules for Linear Bilevel Programming. *Siam Journal on Scientific and Statistical Computing*, **13**: 1194-1217.

Hifi, M., Michrafy, M. et Sbihi, A. (2004). Heuristic algorithms for the multiple-choice multidimensional knapsack problem. *Journal of the Operational Research Society*, **55**(12): 1323-1332.

Hifi, M., Michrafy, M. et Sbihi, A. (2006). A reactive local search-based algorithm for the multiple-choice multi-dimensional knapsack problem. *Computational Optimization and Applications*, **33**: 271-285.

Hiremath, C. (2008). New Heuristic And Metaheuristic Approaches Applied To The Multiple-choice Multidimensional Knapsack Problem. *Ph.D., Department of Biomedical, Industrial and Human Factors Engineering, Wright State University.*

Jarrah, A.I., Yu, G., Krishnamurthy, N. et Rakhit, A. (1993). A decision support framework for airline flight cancellations and delays. *Transportation Science*, **27**: 266-280.

Júdice, J.J. et Faustino, A. (1992). A sequential LCP method for bilevel linear programming. *Annals of Operations Research*, **34**: 89–106.

Kalashnikov, V.V. et Rios-Mercado, R.Z. (2002). A penalty-function approach to a mixed-integer bilevel programming problem. *Rapport de recherche, Universidad Autónoma de Nuevo León, Mexico.*

Kellerer, H., Pferschy, U. et Pisinger, D. (2004). Knapsack Problems. Springer.

Khan, S. (1998). Quality adaptation in a multi-session adaptive multimedia system: model and architecture. PhD thesis, *Department of Electronical and Computer Engineering, Uiversity of Victoria.*

Khan, S., Li, K.F. et Manning, E.G. (1997). The Utility Model for Adaptive Multimedia System. *In: International Workshop on Multimedia Modeling*, pp. 111–126.

Khan, S., Li, K.F., Manning, E.G. et Akbar, M. (2002). Solving the knapsack problem for adaptive multimedia systems. *Studia Informatica, Special Issue on Combinatorial Problems,* **2**(2): 157–178.

Köppe, M., Queyranne, M. et Ryan, C.T. (2009). A parametric integer programming algorithm for bilevel mixed integer programs. *The Smithsonian/NASA Astrophysics Data System, eprint arXiv/*0907.1298.

Lardeux, A., Knippel, G. et Geffard, G. (2003). Efficient algorithms for solving the 2-layered network design problem. *In: Proceedings of the International Network Optimization Conference, Evry, France*, pp. 367–372.

Lettovsky, L., Johnson, E.L. et Nemhauser, G.L. (2000). Airline crew recovering. *Transportation Science.* **34**: 337–348.

Li, V.C. et Curry, G.L. (2005). Solving multidimensional knapsack problems with generalized upper bound constraints using critical event tabu search. *Computers & Operations Research*, **32**(4): 825–848.

Li, V.C., Curry, G.L. et Boyd, E.A. (2004). Towards the real time solution of strike force asset allocation problems. *Computers & Operations Research*, **31**(2): 273–291.

Loridan, P. et Morgan, J. (1989). New results on approximate solutions in two-level optimization. *Optimixation*, **20**: 819-836.

Loridan, P. et Morgan, J. (1996). Weak via strong Stackelberg problem: New results. *Journal of Global Optimization*, **8**: 263-287.

Love, M., Sorensen, K., Larsen, J. et Clausen, J. (2001). Using heuristics to solve the dedicated aircraft recovery problem. *Rapport de recherche IMM-TR-2001-18, Department of Informatics and Mathematical Modelling, Technical University of Denmark*, Kgs. Lyngbvy.

Luo, Z. Q., Pang, J. S. et Ralph, D. (1996). Mathematical Programs with Equilibrium Constraints. *Cambridge University Press, New York, NY.*

Mansi, R., Brotcorne, L. et Hanafi, S. (2008). Reformulation of the Bilevel Knapsack Problem. *ECCO XXI, 21, Croatie*, 42-43.

Mansi, R., Wilbaut, C., Hanafi, S. et Clautiaux, F. (2009). Oscillation Strategy for Disruption Management in the Airline Industry. *MIC* 2009: *the VIII Metaheuristics International Conference, Hambourg (Allemagne), juillet.*

Martello, S., Pisinger, D. et Toth, P. (1999). Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science,* **45**: 414-424.

Martello, S. et Toth, P. (1990). Knapsack Problems: Algorithms and Computer Implementations, *Wiley – Interscience Series in Discrete Mathematics and Optimization, J.Wiley & Sons, New-York.*

Moore, J.T. et Bard, J.F. (1990). The Mixed Integer Linear Bilevel Programming Problem. *Operations Research*, **38**: 911-921.

Moser, M., Jokanovic, D.P. et Shiratori, N. (1997). An algorithm for the multidimensional multiple-choice knapsack problem. *IEICE Transactions in Fundamentals*, **E80-A**(3): 582-589.

Osorio, M.A., Glover, F. et Hammer, P. (2002). Cutting and surrogate constraint analysis for improved multidimensional knapsack solutions. *Annals of Operations Research*, **117**(1): 71-93.

Parra-Hernandez, R. et Dimopoulos, N. (2002). A new heuristic for solving the multi-choice multidimensional knapsack problem. *IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans*, **35**(5): 708–717.

Pirkul, H. (1987). Efficient algorithms for the capacitated concentrator location problem. *Computers & Operations Research*, **14**(3): 197-208.

Pisinger, D. (1995). A minimal algorithm for the multiple-choice knapsack problem. *European Journal of Operational Research*, **83**(2): 394–410.

Pisinger, D. (2001). Budgeting with bounded multiple-choice constraints. *European Journal of Operational Research*, **129**: 471-480.

Raidl, G.R. (1999). The multiple container packing problem: A genetic algorithm approach with weighted codings. *ACM SIGAPP Applied Computing Review, ACM Press*, **7**(2): 22-31.

Rardin, R.L. et Uzsoy, R. (2001). Experimental evaluation of heuristic optimization algorithms: A tutorial. *Journal of Heuristics*, **7**(3): 261-304.

Razzazi, M.R. et Ghasemi, T. (2008). An Exact Algorithm for the Multiple-Choice Multidimensional Knapsack Based on the Core. Advances in Computer Science and Engineering, Communications in Computer and Information Science, **6**: 275-282. ISBN 978-3-540-89985-3 (Online).

Romaine, J.M. (1999). Solving the multidimensional multiple knapsack problem with packing constraints using tabu search. *Rapport de recherche* AFIT/GOR/ENS/99M-15, *Air Force Institute of Technology, Wright-Patterson Air Force Base, Ohio.*

Saunders, R.M. et Schinzinger, R. (1970). A Shrinking Boundary Algorithm for Discrete System Models. *IEEE Transactions On Systems Science and Cybernetics*, ssc-**6**(2): 133-140.

Sbihi, A. (2007). A best first search exact algorithm for the Multiple-choice Multidimensional Knapsack Problem. *Journal of Combinatorial Optimization*, **13**(4): 337–351.

Shimizu, K., Ishisuka, Y. et Bard, J.F. (1997). Nondifferentiable and Mathematical Programming. *Kluwer Academic Publishers.*

Siarry, P. et Dreyfus, G. (1989). La méthode du recuit simulé : théorie et applications. *Ouvrage de synthèse, éditeur : ESPCI - IDSET, 10 rue Vauquelin.*

Sinha, P. et Zoltners, A.A. (1979). The multiple-choice knapsack problem. *Operations Research*, **27**(3): 503–515.

Soyster, A.L., Lev, B. et Slivka, W. (1978). Zero–one programming with many variables and few constraints. *European Journal of Operational Research*, **2**(3): 195-201.

Spielberg, K. et Guignard, M. (2000). A Sequential (Quasi) Hot Start Method for BB (0,1) Mixed Integer Programming. *Mathematical Programming Symposium, Atlanta.*

Spielman, D. et Teng, S.H. (2001). Smoothed analysis of algorithms: why the simplex algorithm usually takes polynomial time. *Proceedings of the Thirty-Third Annual ACM Symposium on Theory of Computing*, ACM, 296–305.

Sriram, C. et Haghani, A. (2003). An optimization model for aircraft maintenance scheduling and re-assignement. *Transportation Research Part A: Policy and Practice*, **37**: 29-48.

Stackelberg, H.V. (1952). The Theory of the Market Economy. *Oxford University Press.*

Talluri, K.T. et Van Ryzin, G.J. (2004). The Theory and Practice of Revenue Management. *Kluwer Academic Publishers.*

Teodorovic, D. et Guberinic, S. (1984). Optimal dispatching strategy on an airline network after a schedule. *Journal of Operations Research*, **15**: 178-182.

Teodorovic, D. et Stojkovic, G. (1990). Model for operational daily airline scheduling. Transportation Planning and Technology, **14**: 273-285.

Teodorovic, D. et Stojkovic, G. (1995). Model to reduce airline schedule disturbances. *Journal of Transportation Engineering*, **121**(4): 324-331.

Thengvall, B., Bard, J. et Yu, G. (2000). Balancing User Preferences for Aircraft Schedule Recovery During Irregular Operations. *IIE Transactions*, **32**(3): 181-193.

Thengvall, B., Yu, G. et Bard, J. (2001). Multiple fleet aircraft schedule recovery following hub closures. *Transportation Research, A*, **35**: 289-308.

Thirwani, D. et Arora, S.R. (1997). An algorithm for the integer linear fractional bilevel programming problem. *Optimization*, **39**: 53–67.

Tillman, I.A., Hwang, C.L. et Kuo, W. (1980). Optimization of System Reliability. *New York: Marcel Dekker.*

Toth, P. (1980). Dynamic programming algorithms for the zero-one knapsack problem. *Computing*, **25**: 29–45.

Toyoda, Y. (1975). A simplified algorithm for obtaining approximate solutions to zero-one programming problems. *Management Science*, **21**(12): 1417–1427.

Trick, M. (2003). A dynamic programming approach for consistency and propagations for knapsack constraints. *Annals of Operations Research*, **118**(1): 73–84.

Vicente, L.N., Savard, G. et Judice, J. (1994). Descent Approaches for Quadratic Bilevel Programming. *Journal of Optimization Theory and Applications,* **81**: 319-399.

Vicente, L.N., Savard, G. et Judice, J. (1996). Discrete linear bilevel programming problem. *Journal of Optimization Theory and Applications*, **89**(3): 597-614.

Vimont, Y., Boussier, S. et Vasquez, M. (2008). Reduced costs propagation in an efficient implicit enumeration for the 01 multidimensional knapsack problem. *Journal of Combinatorial Optimization*, **15**(2): 165-178.

Voudouris, C. et Tsang, E.P.K. (1995). Guided local search, *Rapport de recherche* CMS-247, *Department of Computer Science, University of Essex*, August.

Watson, R.K. (2001). Packet Networks and optimal admission and upgrade of service level agreements: applying the utility model. *M.A.Sc. Thesis, Department of ECE, University of Victoria, Canada.*

Weide, O., Ryana, D. et Ehrgott, M. (2009). An iterative approach to robust and integrated aircraft routing and crew scheduling. *Computers & Operations Research*, doi:10.1016/j.cor.2009.03.024.

Wen, U. et Yang, Y. (1990). Algorithms for solving the mixed integer two-level linear programming problem. *Computers & Operations Research*, **17**(2): 133–142.

Wilbaut, C. et Hanafi, S. (2009). New convergent heuristics for 0–1 mixed integer programming. *European Journal of Operational Research*, **195**: 62–74.

Yan, S. et Lin, C.G. (1997). Airline scheduling for the temporary closure of airports. *Transportation Science*, **31**(1): 72-82.

Yan, S. et Tu, Y. (1997). Multifleet routing and multistop flight scheduling for schedule perturbation. *European Journal of Operational Research*, **103**: 155-169.

Yan, S. et Yang, D.H. (1996). A decision support framework for handling schedule perturbation. Transportations Research, B, **30**(6): 405-419.

Yu, G. (1998). Operations Research in the Airline Industry. *Kluwer Academic Publishers, Dordrecht.*

Yu, G. et Qi, X. (2004). Disruption management: framework, models and applications. *World Scientific Publishing Company.*